Multiple Classifier Systems in Offline Cursive Handwriting Recognition

Inauguraldissertation
der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

Simon Günter

von Aarwangen

Leiter der Arbeit:	Prof. Dr. H. Bunke
	Institut für Informatik
	und angewandte Mathematik,
	Universität Bern

Multiple Classifier Systems in Offline Cursive Handwriting Recognition

Inauguraldissertation
der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

Simon Günter

von Aarwangen

Leiter der Arbeit:	Prof. Dr. H. Bunke Institut für Informatik und angewandte Mathematik, Universität Bern

Von der Philosophisch-naturwissenschaftlichen Fakultät angenommen.

Bern, den 22.1.2004	Der Dekan:
	Prof. Dr. G. Jäger

Abstract

The thesis addresses the application of multiple classifier systems (MCS) methods in the domain of handwriting recognition. To evaluate the MCS methods two different state-of-the-art handwritten word recognizers are used. Both recognizers are based on Hidden Markov Models (HMMs) and process handwritten words in the same manner: First the word image is normalized in respect to slant, skew and height. Then a sequence of feature vectors is extracted from the image. Finally the HMMs are used to determine the most likely word class from a set of candidate word classes. As the performance of every MCS method depends on the performance of the classifiers it is applied to, the optimization of the recognizers is also addressed in the thesis.
There are mainly two kind of MCS methods: The ensemble methods which create several classifiers out of one base classifier and the combination schemes which combine the output of several classifiers to a single output. Both kind of methods are addressed in the thesis. Also the optimal values of the parameters of the MCS methods, such as the number of classifiers created by the ensemble methods, are evaluated in the thesis.
There are two objectives of the thesis. The first objective is to the design new MCS methods and the second objective is to evaluate some classic and the new MCS methods in large scale experiments. Four new categories of ensemble methods and two new combination schemes are proposed in the thesis. In addition the adaption of some classic combination schemes for the use with HMM classifiers is addressed. All methods are tested in a handwritten word recognition task where the recognizer, given an image of a word, has to choose a word from 3997 possible word classes.

Acknowledgment

The work for the thesis was performed during my employment as a research and lecture assistant at the Institute for Computer Science and Applied Mathematics (IAM) of the University of Bern. The research was partly funded by the Swiss National Science Foundation (Nr. 20-52087.97). Additional funding was provided by the Swiss National Science Foundation NCCR program "Interactive Multimodal Information Management (IM)2" in the Individual Project "Scene Analysis".
I owe many thanks to Prof. Dr. Horst Bunke for supervising this thesis and orienting the work. Prof. Dr. Bunke provided me with much extremely useful information, including, but not limited to, research results of the literature, and was always open to my ideas and views. For his role as second examinator I would like to thank Prof. Dr. Hanspeter Bieri. I also would like to thank Andrew and Jennifer for proof-reading this thesis.
Many colleagues have contributed in different ways to this thesis. In particular I would like to thank Dr. Urs-Victor Marti for providing the handwritten word recognizer used in the experiments of this thesis and Matthias Zimmermann for the provision of the handwritten word samples from the IAM database, which he maintained and upgraded. I'm also indebted to Christophe Iringer for the support of the computational infrastructure, especially the Linux cluster. Many thanks go to Dr. Stefan Fischer, Dr. Xiaoyi Jiang, Tamas Varga and Andreas Schlapbach for helpful discussions. Last but by no means least I want to thank Susanne Thüler for administrative support.
Finally, I would like to thank my family who supported me during my studies.

Contents

Chapter 1

Introduction

1.1 Handwriting recognition

The task of handwriting recognition is the transcription of handwritten data into a digital format. The goal is to process handwritten data electronically with the same or nearly the same accuracy as humans. By doing the processing with computers a large amount of data can be transcripted at a high speed. The field of handwriting recognition is divided into the sub-fields of on-line and off-line recognition. In on-line recognition special devices are used to track the movement of the pen and the temporal information is recorded. In off-line recognition an image of the handwritten text is scanned and recorded. In general off-line recognition is considered the more difficult task, because of the lack of temporal information. It is possible to construct the image of the handwriting using the information of the movement of the pen, however it is not possible to reconstruct the information of the movement of the pen using only the image. In this thesis the task of off-line handwriting recognition is considered.

There are already many commercial applications where off-line handwriting recognition systems are used, such as automatic address reading [44, 88] and bank check processing [40]. Other possible applications of off-line handwriting recognition are: The automatic reading of personal notes and communications, and the transcription of handwritten archives into digital libraries. The performance of state-of-the-art systems is in general still too low for those two applications and so there is a need to improve their performance.

The field of off-line handwriting recognition has been a topic of intensive research for many years now. At first, only the recognition of isolated handwritten digits and characters was investigated [91]. Later the recognition of whole words [85] was addressed. The main problem when recognizing whole words is segmenting the words into their individual characters. There are two main approaches used in solving this problem. In the first a set of candidate character boundaries are determined. Normally many more boundaries are found than actually exist in the word. Then the dynamic programming technique is used to determine which of those candidates are the real boundaries and which characters comprise the word. The problem with this approach is that the set of boundary candidates must contain all real boundaries. In addition, the dynamic programming will be very slow if too many candidate boundaries are used. The second segmentation approach is to use a Hidden Markov Model (HMM) recognizer

[75] which implicitly segments the word during recognition. In the last few years the HMM based approach has become dominant [44, 65, 98]. Therefore the work in this PhD thesis focuses on HMM recognizers. Good summaries of the state of the art in off-line handwriting recognition are given in [10, 94].

In an HMM recognizer each class of the recognition problem normally has a corresponding HMM, so for example, there are ten HMMs for a digit recognizer. Often in word recognition characters are modeled instead of the words. The word models are then constructed by concatenating the corresponding character models. The main reason for not using HMM for whole words is the small number of data samples available per word. In addition the assumption that the instances of the same characters are similar and that the shape of the character instances is not too strongly influenced by the context in the word seems to be reasonable. To construct the word HMMs the recognizer needs to know which words should be recognized. The list of recognizable words, the vocabulary, is application dependent and the output of the classifier can only be one of these words. The problem that the true class of the test word is not in the vocabulary was addressed in [93, 96]. In this thesis the vocabulary always contains the true classes of all test patterns. Most of the systems reported in the literature to date consider constrained recognition problems based on small vocabularies from specific domains, e.g. the recognition of handwritten check amounts [40] or postal addresses [44]. Free handwriting recognition, without domain specific constraints and large vocabularies, which will be considered in this thesis has been addressed only recently [47, 65, 96, 105].

In some recent studies [64, 65, 93, 96] the recognition of a whole line of text was considered. These studies assumed that each line consists of a sequence of words from the vocabulary. To prune the search space of all possible word sequence and to increase the performance of the system a language model is applied. A language model defines the probability of a word appearing giving the precedent words. In the most simple case, the uni-gram model, only the word itself is considered. The n-gram model considers also the $n-1$ last words. To compute those probabilities a large collection of text, the corpus, is used. In [105] the recognition of whole sentences of handwritten text was examined and the recognition performance was increased by incorporating a stochastic grammar which was automatically generated using the corpus. In the actual work only the task of word recognition is addressed and no language model is applied.

For the design of a recognizer often a large set of data is needed. This set of data is called the training data and the process of setting the free parameters of a recognizer is called training. A training pattern is labeled when the true class of the pattern is known. The true class of an unlabeled pattern is unknown. In off-line handwriting recognition we normally use labeled data. There are many databases of handwritten text known from the literature [26], e.g. CEDAR [33], NIST [100], the “Senior”-database [84] and the IAM database [66]. The IAM database which was used for this work is described in Section 2.4.

An off-line handwriting recognizer which is trained using only the data of a single writer and then used to classify the patterns of the same writer is called a single writer system. Normally a recognizer should be able to classify data of several writers and so it is trained and tested using words written by many people. Such systems are denoted as multi writer systems. We can sub-classify multi writer systems into writer dependent (WD) and writer independent (WI) systems. In WD systems the training words and the words used for the recognition task were written by the same people. In WI systems the test patterns were written by people who didn’t contribute to the training data. In applications we normally use writer independent systems. For example, in the application of address readers, there are a lot of people who write addresses who didn't contribute to the training data. In some applications it may be useful to adapt a multi writer system to the data of a single writer. Such an application is the transcription of personal notes. In [95] methods to achieve this adaption were studied.

1.2 Multiple classifier systems

In pattern recognition problems the correct classes of input patters should be determined. A system which solves a pattern recognition problem is denoted as a classifier. In this work, where off-line handwriting recognition is addressed, the term classifier is used as a synonym for recognizer. A classifier may output a single class, a ranked list of classes or it may reject the pattern. Often a score value is output with each class. Sometimes this score value is an estimation of the probability of the class being correct, but normally holds less information. In case of rejection the classifier is unable to classify the pattern with any high degree of confidence. In this thesis most experiments were conducted using classifiers which do not reject patterns.

In early studies of pattern recognition only one classifier was used to solve a classification problem. The research domain of multiple classifier systems (MCS) examines how several classifiers can be applied together to obtain better classification systems. It was shown that the MCS methods may improve the recognition performance in difficult pattern recognition problems [51, 52]. MCS methods may also be used to increase the speed of the system [13, 57] or to reduce the time taken for the design of the classification system. The second goal may be achieved by combining several suboptimal classifiers. As will be shown later in this thesis such combination may lead to similar or better performances than a single optimized classifier. There are two main issues in MCS research. The first issue is how the individual classifiers are created or designed, and the second is how those classifiers are combined.

Often several classifiers with different architectures already exist that solve the same classification problem. This is because the first step in the design of a classic pattern recognition system is to find the best classifier for the problem. Therefore classifiers with different architectures were tested. Additionally several research groups used different approaches to solve the same problem, so that each of these groups designed its own different classifier. Another approach to obtain a set of classifiers is to produce a large number of classifiers and then to reduce the set [45]. Recently a number of methods that automatically generate several classifiers when given a single classifier have been proposed in the field of machine learning [16]. These methods are called ensemble methods and the single classifier input to these methods is denoted as the base classifier. The different classifiers are produced by modifying the training set [8, 21], the used feature set [28], the input data by injecting randomness [17], or the parameters and architecture of the base classifier [73]. A summary of ensemble methods is given in [16] and the methods applied in this work are described in Chapter 4.

There are three basic structures of classifier combination: serial, parallel and conditional. In a serial combination the output of a classifier is the input of the next classifier. Normally a classifier in a serial combination reduces the set of candidate classes so that the final classifier output is only one class. Serial combination of classifiers was applied in [24, 77]. In conditional

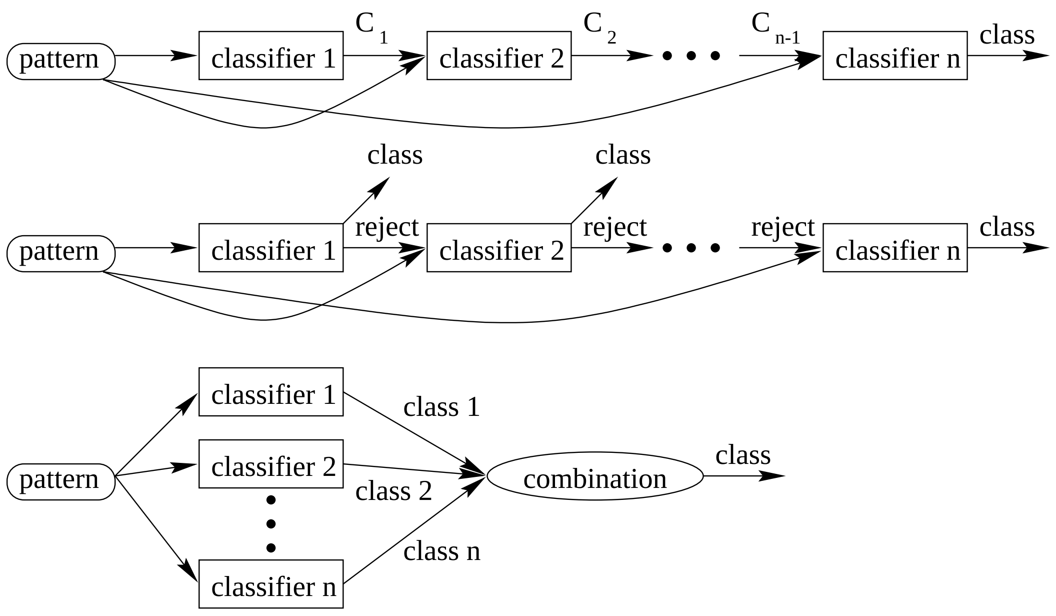


Figure 1.1: The different structures of classifier combination: Serial (top), conditional (center) and parallel (bottom).

combination some classifiers are only applied if the output of other classifiers meets a certain condition. In an often used conditional classifier combination the second classifier classifies only the pattern rejected by the first classifier [13, 57]. The second classifier deals with the "difficult" patterns and is normally more complex and slower than the first. In a parallel combination the output of all classifiers is combined in a last step to a single output. This combination [49, 91] is very popular, because each classifier that solves the classification problem may be used (for the serial and conditional combination normally special classifiers are used). In addition classifiers produced by an ensemble method are normally combined in parallel. In Figure 1.1 the different structures of combinations of classifiers are illustrated.

A combination that is a mixture of the above mentioned basic structures is denoted as hybrid combination. Such a combination was for example applied in [27]. In this thesis we focus on parallel combination, because it is the most popular one and because it is the combination used for classifiers created by ensemble methods.

There are many ways to combine in parallel the results of a set of classifiers, depending on the type of the classifiers' output [19, 90]. If the output is only the best ranked class then majority voting can be applied. More sophisticated voting schemes also look at the probability of the classification error for a specific class (Bayesian Combination Rule [49]) and the dependencies between the classifiers (Behavior-Knowledge Space [31]). Some classifiers have a ranked list of classes as output. For them often Borda count [29] or related methods are used. Some classifiers also generate a score value for each class. In this case the sum, product, maximum, minimum, or the median of the scores of all classifiers can be calculated and the class with the highest value is regarded as the combined result [49]. It is also possible to first weight each classifier according to its individual performance and then apply a combination rule [30]. Another approach is to use the score values output by the individual classifiers as input for a trainable classifier, e.g. a neural-network [99], which acts as the combiner (this method may be regarded as a hybrid combination). There are many papers in the literature which compare these combination schemes on several problems. In [1] the schemes are compared depending on the degree of perturbation imposed on the input patterns. Under small perturbations the product and minimum rule were best, whereas for large perturbations median and sum rule were superior. In [59] a system is presented, which perturbates the output of the classifiers when tested on the training set, and then chooses the combination scheme that leads to the best recognition performance for a wide range of perturbation parameters. More details on the topic of combination methods are given in Chapter 5.

1.3 MCS in handwriting recognition

Handwriting recognition is one of the most difficult problems in pattern recognition and it has been shown that MCS methods may increase the recognition performance for such difficult problems. So it is straightforward to apply MCS methods to the problem of handwriting recognition. There are many previously published studies of such applications. In most cases optimized individual classifiers have been combined. A combination of three classifiers using the sum rule is described in [62] for the problem of handwritten numerals recognition. Often specialized combination structures for specific tasks are used. In [5] a multilayer perceptron is trained to classify handwritten numerals. For each pair of numerals that is likely to be confused a support vector machine is trained. If the two best classes output by the multilayer perceptron are the digits of such a pair, the specialized classifier is invoked. Otherwise the best class of the multilayer perceptron is output. In [61] a rule-based classifier is serially combined with four Hopfield neural networks, which are only used when the first classifier rejects the input. The task of the first neural network is to select one of the other three as the final classifier. This system was also used for recognition of handwritten digits.

For cursive handwritten word recognition, Hidden Markov Models have become a standard tool [40, 44, 46, 65]. Consequently, combinations of HMMs with other classifiers have been proposed. In [68] an HMM classifier is combined with a pattern matching classifier. A holistic classifier to reduce the vocabulary for an HMM classifier is described in [46]. Another example of combining an HMM with a classifier of another type is given in [7], where the calculation of the observation probabilities in an HMM is accomplished by a neural network. Surprisingly, the combination of several HMM classifiers is rare. In [6] a pseudo 2-dimensional HMM and a 1-dimensional HMM were combined with the sum rule. A special way to combine HMM classifiers is described in [60, 97] which is similar to the one described in Section 5.4. This combination takes advantage of the HMM architecture of word recognizers.

Although the popularity of multiple classifier systems in handwritten recognition has grown significantly, not a great deal of work on the use of ensemble methods for HMM classifiers has been reported in the literature. An exception is [71], where Bagging was applied in an isolated handwritten characters recognition task.

1.4 Goal and structure of the thesis

The goal of this thesis is to evaluate classic and new MCS methods in handwriting recognition problems. A significant part of this thesis addresses the use of ensemble methods for HMM classifiers in handwriting recognition. The author hopes that this work will help to increase our knowledge of this little researched area. Not only the classic ensemble methods, like Bagging and Boosting, are evaluated, but also some other methods that are introduced in this work. The produced ensembles were combined by several classic and some newly developed combination schemes.

The structure of this thesis is as follows. In Chapter 2 the handwritten word classifiers and the handwritten text database used in the experiments are introduced. In the following chapter some methods to optimize those classifiers are described. These methods were applied in the experiments to produce state-of-the-art base classifiers. Chapter 4 contains the description of classic and new ensemble methods used in the experiments. In Chapter 5 the topic of parallel classifier combination for the problem of handwritten word recognition is addressed. Some classic and some new combination methods are discussed in detail. In Chapter 6 the results of initial experiments are given. To goal of these experiments is to find an optimal base classifier and to select some well performing ensemble methods and combination schemes. In addition the best values of the free parameters of the ensemble methods, for example the number of classifiers in the ensemble, are determined. In Chapter 7 the results of the best ensemble methods and combination schemes using an optimized base classifier are presented. Conclusions of the results are then drawn in Chapter 8 and possible future work is discussed.

Chapter 2

Handwritten word recognizers and database

In this chapter, the handwritten word recognizers and the database used in the experiments are introduced. Two different Hidden Markov Model classifiers were used in the experiments of this thesis. These classifiers are described in detail in Sections 2.2 and 2.3. In Section 2.1 a description of the common characteristics of the two classifiers is given. The IAM database is described in Section 2.4.

2.1 HMM classifiers for handwriting recognition

HMM based handwritten word recognizers normally consist of three main modules (see Fig. 2.1): the preprocessing, where noise reduction and normalization take place, the feature extraction, where the image of a handwritten word is transformed into a sequence of numerical feature vectors, and the recognizer, which converts these sequences of feature vectors into a word class.

The first step in the processing chain, the preprocessing, is mainly concerned with text image normalization. The goal of the different normalization steps is to produce a uniform image of the writing with less variation of the same character or word across different writers. The aim of feature extraction is to derive a sequence of feature vectors which describe the writing in such a way that different characters and words can be distinguished, but which do not contain too

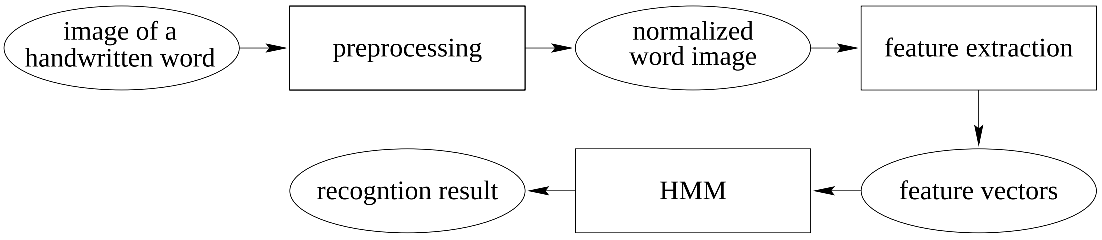


Figure 2.1: System overview

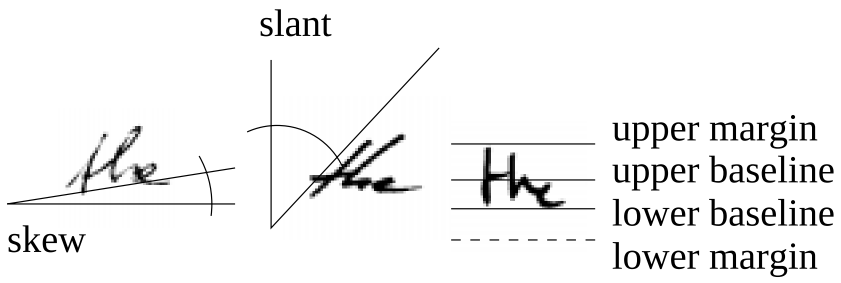


Figure 2.2: Normalized features of the handwriting: Skew, slant and baseline positions

much redundant information. At the core of the recognition procedure is an HMM. It receives a sequence of feature vectors as input and outputs a word class. Instead of a sequence of numerical feature vector a sequence of integers, which corresponds to classes of input values, is input to some HMM classifiers. Such HMM classifiers are denoted as discrete HMM classifiers, while the classifiers that use numerical feature vectors are denoted as continuous HMM classifiers. A semi-continuous HMM classifier type for which the numerical feature vectors are clustered, and the same input class is assigned to all vectors of a resulting cluster does also exist. Gaussian mixtures are often used to model the distributions of the feature vectors of the clusters. The probability of outputting a feature vector of a cluster is stored in the states of HMMs. The feature extraction depends on the type of HMM classifier, of course. As both classifiers that are used in the experiments are continuous HMM classifiers, only this type of HMM classifier is described in Subsection 2.1.3. A more detailed description of the general properties of HMM classifiers for handwriting recognition is given in [64, 65].

2.1.1 Preprocessing

Each person has a different writing style with its own characteristics. This fact makes the recognition of handwriting complicated. To reduce variations in the handwritten texts as much as possible, a number of preprocessing operations are applied. The input for these preprocessing operations are images of text lines or words extracted from a database. The normalized features of the handwriting are normally the skew, the slant and the positions of the baselines. Figure 2.2 illustrate the meaning of these terms. An example of these normalization operations is shown in Fig. 2.3.
The following normalization steps are therefore executed:

- Skew Correction: The word is horizontally aligned, i.e. rotated, so that the baseline is parallel to the x-axis of the image.

- Slant Correction: By applying a shear transformation, the writing's slant is transformed into an upright position.

- Line Positioning: The word 's total extension in a vertical direction is normalized to a standard value. Moreover, applying a vertical scaling operation the location of the upper and lower baseline are adjusted to a standard position. In both systems used in this thesis

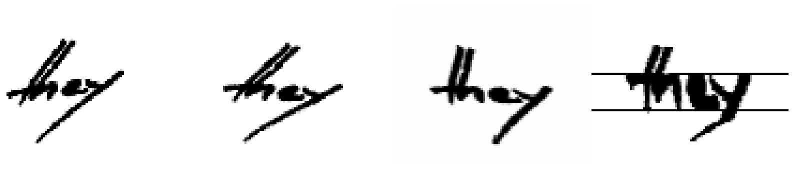


Figure 2.3: Preprocessing of the images. From left to right: original, skew corrected, slant corrected and positioned. The two horizontal lines in the picture on the far right are the two baselines.

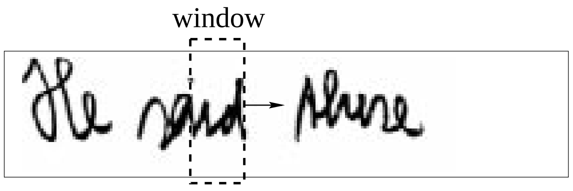


Figure 2.4: Illustration of the sliding window technique. A window is moved from left to right and features are calculated for each position of the window.

the normalized height of the parts between the upper margin and the upper baseline, between the upper baseline and the lower baseline and between the lower baseline and the lower margin is 40 pixels.

It should be noted that all preprocessing steps were executed on whole text lines of the underlying database (compare Section 2.4), despite the fact that the recognition task is carried out for words. So the quality of the preprocessing may be higher than in the case when only preprocessing the words alone, because more data is available.

2.1.2 Feature extraction

The HMM classifiers take a sequence of numerical feature vectors as input. In the feature extraction a sequence of some informative key values (the features) are extracted from the image. The HMM classifiers used in the experiments apply a so-called sliding window approach. A window is moved from left to right over the image and at each position a feature vector is extracted. The height of this window is equal to the height of the image. The width of the window for the two classifiers is 1 and 16 pixels respectively. After the extraction of a feature vector, the window is shifted only one pixel to the right. Therefore the number of extracted feature vectors is equal to the width of the image. A graphical representation of this sliding window technique is shown in Fig. 2.4.
The two classifiers described in Sections 2.2 and 2.3 differ mainly in the features that are extracted from the sliding window.

2.1.3 Hidden Markov models

Hidden Markov models (HMMs) are widely used in the field of pattern recognition. Their original application was in speech recognition [76]. Because of the similarities between speech and cursive

handwriting recognition, HMMs have become very popular in handwriting recognition as well [55].
When using HMMs for a classification problem, an individual HMM is designed for each pattern class. For each observation sequence, i.e. for each sequence of feature vectors, the likelihood that this sequence was produced by an HMM of a class can be calculated. The class whose HMM achieved the highest likelihood is considered as the class that produced the actual sequence of observations.
An HMM consists of a set of states, and transition probabilities between those states. One or several of the states are defined as final states. For each state a likelihood value for each possible observation is defined. To define this likelihood value a probability distribution of the feature vectors is used. A valid sequence of states for a observation sequence $o_{seq} = o_1, o_2, \ldots, o_n$ is $s_{seq} = s_1, s_2, \ldots, s_n$ where s_n is a final state. Note that the number of states in s_{seq} is the same as the number of observations in o_{seq}. The likelihood of the sequence of states s_{seq} is the product of the likelihood of observing o_i in state s_i for all observations, multiplied by the probabilities of the transitions from state s_i to s_{i+1} for all $i \in \{1, \ldots, n-1\}$. There are two possibilities for defining the likelihood of an observation sequence o_{seq} for a given HMM. Either the highest likelihood of all possible state sequences is used (Viterbi recognition), or the sum of the likelihood of all possible state sequences is considered as the likelihood of the observation sequence (Baum-Welch recognition). Both systems described in the next sections use the first possibility.
In word recognition systems with a small vocabulary, it is possible to build an individual HMM for each word. But for large vocabularies this method no longer works because of a lack of sufficient training data. Therefore, in our systems, an HMM is built for each character. The use of character models allows us to share training data. Each instance of a character in the training set has an impact on the corresponding HMM, which in turn leads to a better parameter estimation. By using character instead of word models, we assume that all the instances of a character are similar, and are not (strongly) dependent on the context of the instance within the word. It is obvious that this assumption does not hold, but the drawback of loosing the context information of the character was shown to have a smaller impact on the recognition performance than the advantage of the improved parameter estimation. To model entire words, the character models are concatenated with each other. Thus a recognition network is obtained (see Fig. 2.5). Note that this network doesn't include any contextual knowledge on the character level, i.e., the model of a character is independent of its left and right neighbor.
To achieve high recognition rates, the character HMMs have to be adapted to the problem. In particular the number of states, the possible transitions and the type of output probability distributions have to be selected. Because of the left to right direction of writing, a linear transition topology has been chosen for the character models of both systems used in the experiments. In a linear topology only the same or the succeeding state can be reached from each state. A graphical representation of an HMM with linear topology and 14 states is shown in Fig. 2.6. Because of the continuous nature of the features extracted by the classifiers, described in the following two sections, probability distributions for the features are used. The probability distributions of all states are assumed to be Gaussian mixtures, i.e. weighted sums of Gaussians. It is a well known fact that any distribution may be approximated by Gaussian mixtures. The individual Gaussian distributions are assumed to have diagonal covariance matrices. This assumption reduces the

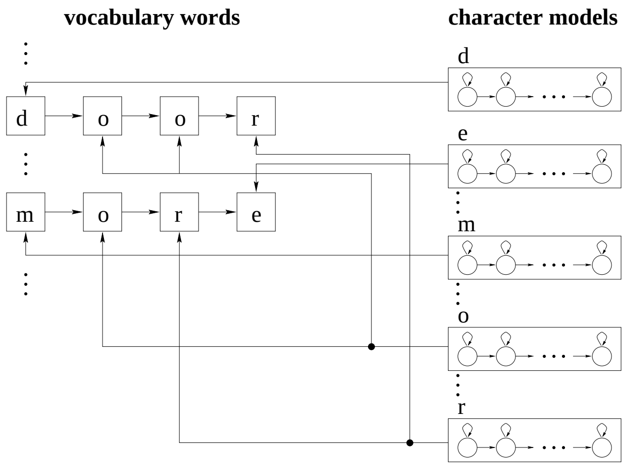


Figure 2.5: Concatenation of character models yields the word models

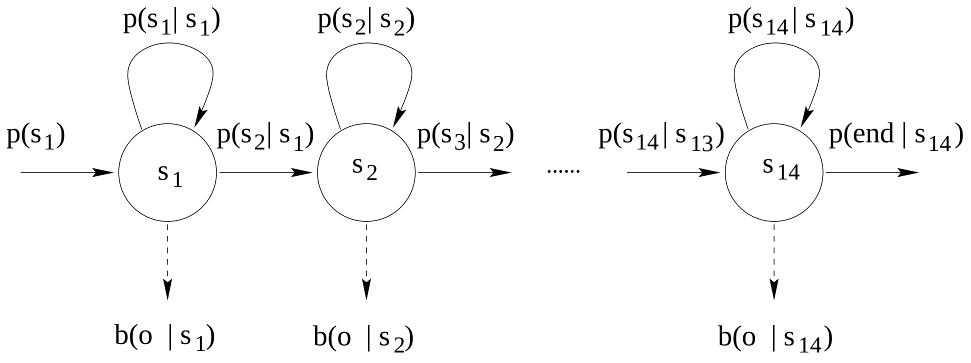


Figure 2.6: HMM with linear topology

number of free parameters from $n \cdot (n+1)$ to $2 \cdot n$ with n being the number of features, because no covariance terms must be calculated. Yet results based on this assumption are only exact when the features are statistically independent. This is not the case for the systems used in the experiments, but the negative impact of only calculating approximations of the exact values is smaller than the positive effect of getting better estimations for the free parameters.

To set the free parameters of the HMMs, the transitions probabilities, and the parameters of the Gaussian distributions, the Baum-Welch training [76] is used. Baum-Welch training is a version of the Expectation-Maximization technique [14] and works with labeled training data. The product of the likelihood values for the correct word HMMs of the training patterns is guaranteed to increase in each iteration of the algorithm, yet the recognition rate of the classifier may decrease when using too many iterations. This is because the HMMs are overfit to the training data. As mentioned before, the recognition is performed using the Viterbi algorithm [76]. For each word in the vocabulary the likelihood is calculated that a given feature vector sequence was produced by the HMM of the word. The word corresponding to the HMM with the highest likelihood is the output of the HMM classifier. Normally the calculated likelihood is returned by the classifier. Often the words corresponding to the HMMs with the n best likelihoods are output, producing a n-best result list. For more details on the topic of HMMs see [76], for example.

Both HMM classifiers described in the next two sections were implemented using the Hidden Markov Model Toolkit (HTK) [102], which was originally developed for speech recognition.

2.2 Geometric features based recognizer

The system described in this section was designed by Dr. Urs-Victor Marti [64, 65]. To extract the features from the image, a sliding window approach is used, where the width of the window is one pixel, i.e. the window contains one image column. Nine geometrical features are extracted from this column. The first three features are the weight of the window (i.e. the average grey value), its center of gravity, and the second order moment of the window. This set characterizes the window from the global point of view. It includes information about the amount of pixels and their position within the window. The other features represent additional information about the writing. Features four and five define the position of the upper and the lower contour in the window. The next two features, numbers six and seven, give the orientation of the upper and the lower contour in the window by the gradient of the contour at the window's position[1]. For feature number eight, the number of black-white transitions in the vertical direction is used. Finally, feature number nine gives the average grey value between the upper and lower contour. It must be noted that features four to eight are calculated on binary images, while the other features are calculated on grey value images. In the implementation, this was addressed by using grey value images, and doing an implicit binarization using a threshold of 200 for features 4 to 8. This means that for these features, pixels with grey values between 201 and 255 are regarded as white pixels, and all others are regarded as black pixels. In summary, the output of the feature extraction phase is a sequence of 9-dimensional feature vectors.

In the base system used in this thesis the probability distributions of the states are single Gaussians. Most character HMMs consist of 14 states, similar to the original system of Dr. Urs-Victor Marti [64, 65]. Exceptions are the HMMs corresponding to the the comma and the period where two states were used. The number 14 for the quantity of states has been found empirically [64]. This rather high number can be explained by the fact that the sliding window used for feature extraction is only one pixel wide, and that many different writing styles are present in the used database. The training of the system consists of 4 iterations of the Baum-Welch algorithm, and only one Gaussian is used in the Gaussian mixture distributions of the states. This number of 4 iterations was found to be optimal for the above described system for a large range of vocabulary and training set sizes. Several optimization methods for the above described system are introduced in Chapter 3.

[1]To calculate the contour direction, the next image column is also considered. This means that for these two features the window width is actually two.

2.3 Sub-sampling features based recognizer

The feature extraction of the system described in this section is based on the system designed by Dr. Alessandro Vinciarelli [93, 96]. All the other parts of the system were adopted from the system of Dr. Urs-Victor Marti (see last section). Therefore the system used in the experiments in this thesis differs from the one introduced in [93, 96] and may therefore produce different results. There is, for example, another preprocessing mechanism and another HMM architecture (the system in [93, 96] used Bakis HMMs). In addition, the optimization methods described in Chapter 3 were not applied in [93, 96].
As in the system of the last section, a sliding window approach is used for the feature extraction, but for the actual system the window has a width of 16 pixels. Given the content of this window the following steps are applied:

1. All connected components[2] are determined. Components which have no pixels between the upper and lower baselines are removed. The goal of this step is to eliminate horizontal strokes belonging to neighboring characters, as in the top stroke of the letter "T".

2. The part of the window above the upper most black pixel and below the lower most black pixel are removed. Therefore the height of the window may decrease.

3. The window is partitioned into 16 cells arranged in a 4×4 grid. Each cell always has a width of 4 pixels, but the height of the cells may vary.

4. The number of black pixels in each cell divided by the total number of black pixels in the window is then used as a feature.

Because there are 16 cells per window, the output of the feature extraction phase is a sequence of 16-dimensional feature vectors. Those 16 features are strongly correlated, which has a negative impact on the performance of the recognizer. In [93, 96] some transformations were applied on the extracted feature vectors to solve this problem. For the system used in this thesis, the Karhunen-Loeve transformation [48], a linear Principal Component Analysis, was applied. This transformation is described in Section 3.4.

2.4 Handwritten text database

All the handwritten text data used to train and test the recognizers originate from the IAM database [66]. The IAM data has been collected at the Institute of Computer Science and Applied Mathematics at the University of Bern during the last five years. The raw data consists of 1,500 forms of handwritten English text from over 500 writers. These 1,500 forms include over 10,000 handwritten lines and 100,000 word instances. The forms were scanned with a resolution of 300 dpi. A description file in XML format for each form of handwritten text also exists. The description, together with some programs, allows the extraction of individual lines and words from the forms, and provides the normalization parameters, i.e. the slant and skew angles, and

[2]Connected components are connected regions of black pixels. Here each pixel with a lower grey value then 200 is regarded as black.

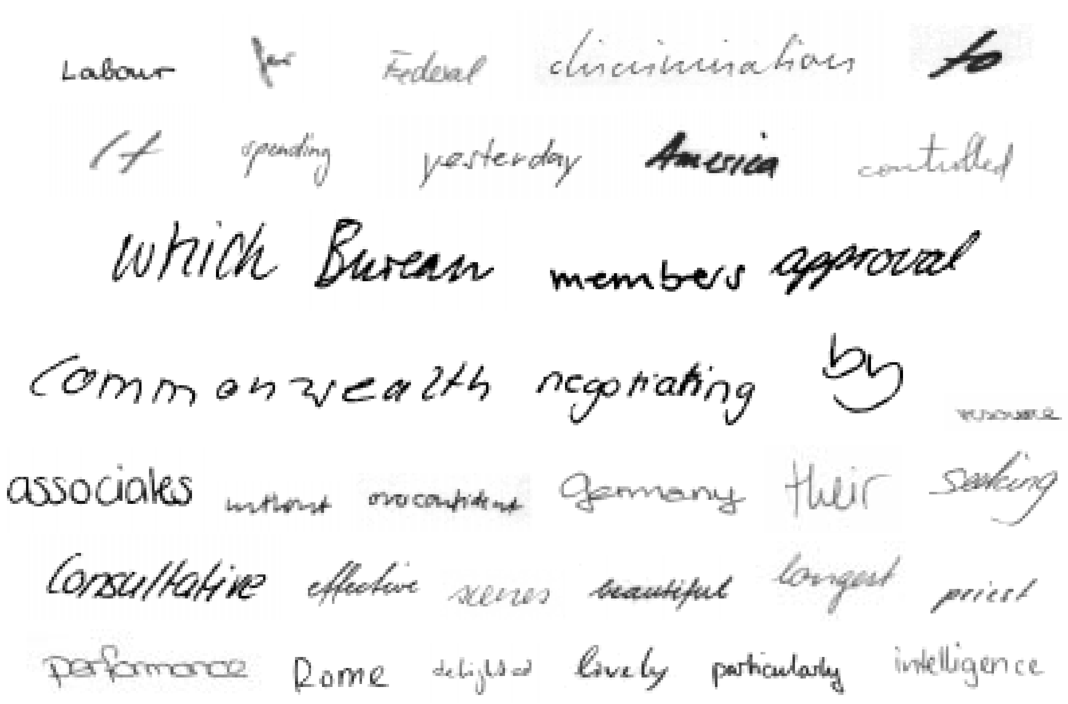


Figure 2.7: Sample of words from the IAM database

the baseline positions of the text lines. The semi-automatic line segmentation procedure applied to obtain the word boundaries is described in [104]. This procedure is not always successful. If a line is mis-segmented, which happens in roughly 10 % of all cases, the position of the words in that line are undefined and the individual words can not be extracted.
The transcriptions of the texts of the forms were adopted from the LOB corpus, which is available electronically. The LOB corpus contains text from different domains, e.g. reports, press reviews and books. Each form of the database has an identification tag. The first letter of this tag indicates the domain of the original text. The IAM database contains forms from 15 domains. In the experiments in this thesis only words from 5 domains were used, so the full potential of the database was not exploited. The reason for only using a part of the available data was the limited computational resources (the time complexity of the HMM classifiers described in the two last sections is rather high).
The IAM database is freely available to other researchers upon request[3]. A small sample of words from the IAM database is shown in Figure 2.7.

[3]see http://www.iam.unibe.ch/~zimmerma/iamdb/iamdb.html for details or contact office@iam.unibe.ch

Chapter 3

Classifier optimization methods

This chapter contains descriptions of several methods to optimize the handwritten word recognizers introduced in Chapter 2. In Section 3.1 methods to optimize the number of states of the HMMs are presented. Section 3.2 contains descriptions of methods to optimize the training procedure and to establish the optimal number of Gaussians used in the Gaussian mixtures distributions. In the following section some methods to select a "good" subset of all features are presented. The quality of the feature set is defined as the performance of the classifier which uses only those features to recognize words, or an estimation of its performance. Section 3.4 discusses which method for transforming the feature vector is most suitable for the considered application. The goal of the transformation is to achieve an increased performance of the classifier using the transformed features. Finally a rejection mechanism is presented in Section 3.5. The aim of the rejection mechanism is to increase the accuracy of the classifiers. Most optimization methods introduced in this chapter would also work for HMM classifiers other than the two described in Chapter 2.

3.1 State number optimization

The base systems of the classifiers described in Chapter 2 use the same number of states for almost all HMMs. The optimal number of states was determined simply by testing the system with different values for this parameter. For the system of Dr. Urs-Victor Marti, best performance was achieved when using 14 states per HMM [64, 65]. By using the same number of states for all HMMs, we assume that the optimal number of states is the same for all characters. Unfortunately but naturally this assumption is flawed. Consider for example the two letters "u" and "w". The letter "w" is basically two "u"s fused together, so the optimal number of states for "w" should be roughly double that of the optimal number of states for "u". The approach using the same number of states for all HMMs is denoted as *const-state*.
In [103] two approaches used to to determine the optimal state numbers of the individual HMMs were presented. Both methods set the state number of a character HMM based on the distribution of the lengths of the feature vector sequences corresponding to this character. For the classifiers described in Chapter 2 the length of the feature vector sequence is equal to the width of the image of the character in pixels. So we need to calculate the distribution of the widths

of images of the character. This was done by training the system using a very low number of 8 states per HMM[1] and testing it in "forced alignment" mode on the training set. In the "forced alignment" mode the recognizer knows the correct class label of each input word and has only to provide the optimal alignment of the sequence of feature vectors with the HMM states. From this optimal alignment, the character boundaries and the width of the character images can be derived. Although the system has a suboptimal architecture (due to the low fixed number of states) it is still able to find good estimates of the character boundaries, because the correct labels are given to the recognizer. Please note that since the HMMs have eight states, eight is the minimum measured width of a character. The two methods described in [103] to set the state numbers as follows:

- **Bakis method**: The Bakis method [4] sets the number of states of each HMM to the average width of the corresponding character images multiplied by a constant f. The theory of the Bakis method is that the optimal number of the HMMs is always the same fraction of the average width of the corresponding character images. The optimal value for f must be determined by experiment.
- **Quantile method**: In the Quantile method the number of states of an HMM is set to the highest number with the following property: The fraction of images of the corresponding character that have a smaller width than this state number is smaller than a constant q. This means that at most a fraction of q of all images are too small for the model. The theory of the method is that the models should be as large as possible, so that all the structure of the data can be modeled, but not so large that the model is too large for a significant part of the training data. The optimal fraction q of images that can not be modeled is assumed to be the same for all characters. Also for this method the optimal value of the only parameter, q, must be determined.

In Figure 3.1 an example of a character image width distribution is shown where the widths of a total of 10 character images were recorded. The average character width is 11.2, so using the Bakis method with $f = 0.54$ we obtain a (rounded) state number of 6. For the Quantile method we obtain a state number of 8 if q is smaller than 0.1. If q is between 0.1 and 0.3, the Quantile method gives a state number of 10.
Normally the Quantile method produces larger HMMs than the Bakis method. This is because the HMMs produced by the Quantile method always have state numbers which are larger than two values. One of this values is 8, the fixed state number of HMMs used by the system to extract the character boundaries. The other value is the smallest estimated width of all images of the character. There is no such limit for the state numbers in the Bakis method, e.g. by setting f to 0 all state numbers of the characters are 0 (the resulting system would of course be completely useless).

3.2 Training method optimization

The training of the system introduced in [64, 65] is quite simple. The single Gaussian distributions of the states were initialized by the global mean and variance of the features and a fixed

[1]For special characters, like punctuation, the state number was set even lower.

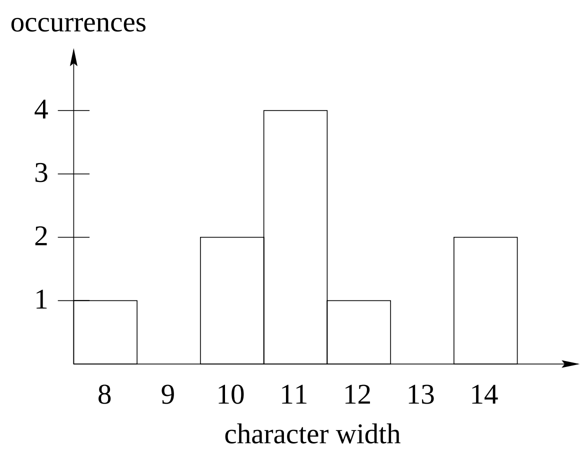


Figure 3.1: Example of a width distribution of character images

number of 4 training iterations of the Baum-Welch algorithm was applied. The number of 4 training iterations has shown to be the optimal value for many recognition tasks with different training and test sets and different vocabulary sizes. When using less than 4 iterations poorly trained models were produced, but if more than 4 training iterations were used, the models overfit the training data. If Gaussian mixture distributions are used, the initialization of the parameters is no longer straightforward. A possible initialization would be to set the variance and the mean of the Gaussians to the global value and to randomly distort the mean of each Gaussian. It was observed that this initialization lead to poorly estimated distributions, because the models "got stuck" in local maxima of the search space. So the standard approach to train HMM classifiers is as follows: Firstly a classifier with single Gaussian distributions is trained with the optimal number of training iterations. Then the number of Gaussians in the distributions is gradually increased. The classifier using single Gaussian distributions is already quite a good performing classifier and by gradually increasing the number of Gaussians the classifier is gradually refined. This approach does not tend to "get stuck" in local maxima.

To increase the number of Gaussians in a Gaussian mixture distribution the splitting procedure is applied. The splitting procedure replaces the Gaussian with the highest weight by two new Gaussians. The means of the two new Gaussians are distorted versions of the mean of the original Gaussian. In the Hidden Markov Model Toolkit (HTK) [102] the length of the distortion vector is 0.2 times the standard deviation of the original Gaussian. The variance of the two new Gaussians is set to the variance of the removed Gaussian. Because the sum of the weights of all Gaussians must be 1, the weights of the new Gaussians is set to half the weight of the original Gaussian. If the number of Gaussians is to be increased by more than one then several sequential splitting procedures are executed.

Based on the above mentioned standard approach to training classifiers with Gaussian mixtures, three strategies to find the optimal training method are proposed. They all use a validation set to measure the quality of the produced classifiers and they differ in the number of training iterations applied between increasing the number of Gaussians, i.e. they differ in the way the system is adapted to the new situation after increasing the number of Gaussians. The three strategies are as follows:

1. The algorithm underlying the first strategy is shown in Table 3.1. First the classifier using single Gaussian distributions is trained with the optimal number of training iterations. Then the number of Gaussian mixtures is iteratively increased. After each increase of the number of Gaussians a fixed number, C, of training iterations is performed. When the desired number of Gaussians per distribution is reached, the classifier is tested on the validation set immediately after each training cycle to find the optimal number of training iterations for the final system, i.e. the system with the desired number of Gaussians. The system is tested up to MAXITER training iterations. The training is then continued until the performance of the system doesn't increase any longer. The motivation of this approach is that the system should adapt itself to the new configuration after each increase of the number of Gaussians. For this adaption a fixed number of training cycles is executed.

2. The second strategy is identical to the first, when C=0. This means that the number of Gaussian mixtures is increased up to the desired number without any training iterations after an increase of the number of Gaussians. The second part, i.e. establishing the optimal number of training iterations for the system with the desired number of Gaussians, is the same as in the first strategy. In this approach it is assumed that it is not necessary to adapt the system immediately after each increase of the number of Gaussians.

3. The algorithm of the third strategy is shown in Table 3.2. In this strategy the optimal number of training iterations is searched after each increase of the number of Gaussians, which means that the system is optimized immediately after a new Gaussian has been added. It is assumed that the optimal number of training iterations is not smaller than MINITER. Also here the system is trained with more than MAXITER iterations if the performance of the system keeps increasing. The algorithm adds Gaussians until the minimal number of Gaussians GAUSSIANS are produced. More Gaussians are added until the performance of the system no longer increases.

In the first two strategies the desired number of Gaussians, GAUSSIANS, must be defined. To establish the optimal training method the algorithms of these two strategies must be executed with a suitable range of values for GAUSSIANS, which is application dependent. In the third strategy the constant GAUSSIANS is the minimal number of Gaussians for the classifier. If this value is too low, it may occur that the best solution isn't found, because the algorithm stops in a local maximum. If the value of GAUSSIANS is set appropriately, the third strategy must only be applied once to find the optimal training method. The constants MINITER and MAXITER are also application dependent, and good values for them must be found empirically. If one wants to be certain of not missing the optimal number of training iterations, a low value for MINITER and a high value for MAXITER should be used.
In all three strategies the number of Gaussians is increased for all states of all HMMs at the same time. So the number of Gaussians is optimized globally for all states. To avoid strong overfitting effects, the HTK splitting procedure prohibits the splitting of a Gaussian when the estimation of the parameters of that Gaussian was done using too few feature vectors [102].
The three strategies introduced above search for the optimal training method. There are two possibilities in using this training method to train the final system:

```
$B$ is the optimized base classifier with single Gaussian distributions;
for($i = 2$;$i <$GAUSSIANS;$i$++)
    increase number of Gaussians of $B$ by 1;
    train $B$ with $C$ iterations;
increase number of Gaussians of $B$ by 1;
$B'$ is a copy of $B$;
for($i = 1$;($i \leq$MAXITER) or ($best\_iterations = i - 1$);$i$++)
    train $B'$ with 1 iteration;
    test $B'$ on the validation set;
    if($B'$ is the best tested classifier yet)
        $best\_iterations = i$;
train $B$ with $best\_iterations$ iterations;
return $B$ and the used training method of $B$;
```

Table 3.1: First strategy to find the optimal training method

```
$B$ is the optimized base classifier with single Gaussian distributions;
for($i = 2$;($i \leq$GAUSSIANS) or (performance of $B$ increased);$i$++)
    increase number of Gaussians of $B$ by 1;
    $B'$ is a copy of $B$;
    for($j$=MINITER;($j \leq$MAXITER) or ($best\_iterations = j - 1$);$j$++)
        train $B'$ with 1 iteration;
        test $B'$ on the validation set;
        if($B'$ is the best tested classifier with $i$ Gaussians)
            $best\_iterations = j$;
    train $B$ with $best\_iterations$ iterations;
return best tested classifier and its training method;
```

Table 3.2: Third strategy to find the optimal training method

1. The final system is trained with the optimal training method on the training set only. In this case the final system is the classifier returned by the algorithms given in Tables 3.1 and 3.2.

2. The final system is trained with the optimal training method on the union of the training and validation set.

The second possibility has the advantage that more training data is available and it is a well known fact that systems trained on larger datasets normally outperform systems trained on smaller ones. Yet there may be a bias in the sense that the derived parameter values are only optimal for the original training set. Consider for example a k-nearest neighbor classifier. Assume the training and validation set have roughly the same size and the validation set is used to determine the best number of k. This number corresponds to an "optimal" region around test patterns where training patterns are considered for the classification. By combining training and validation sets the density of patterns is doubled and this region will be much smaller. To

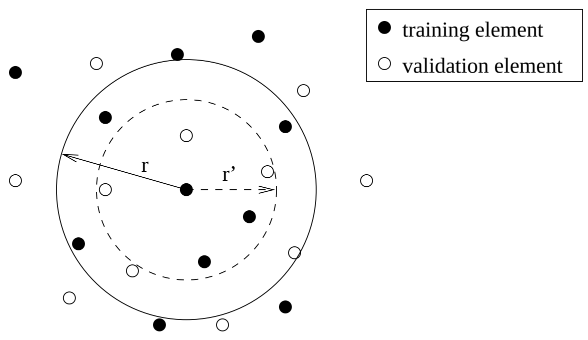


Figure 3.2: Different regions where elements are considered in a 5-Nearest Neighbor classifier. When only the training elements are used, the radius of the circle containing the considered elements is r. If also the validation elements are used then this radius becomes r'.

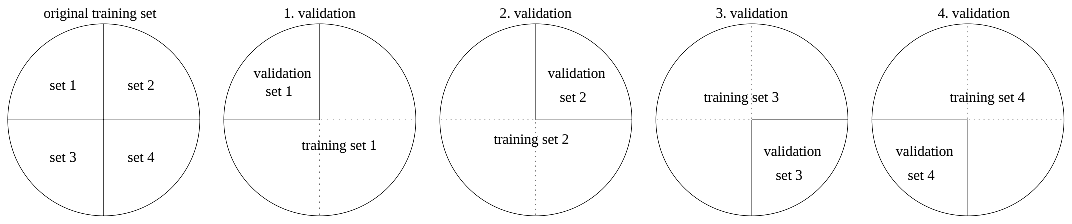


Figure 3.3: 4-fold cross-validation

have regions with a size similar to the one used in the validation process, a larger value of k must be used. This situation is illustrated in Fig. 3.2.
In order to measure the performance of a system to optimize some parameter values, cross-validation may be used instead of splitting the whole data set into a training and a validation set. In cross-validation the training set is randomly split into n equally sized partitions. Then the system is trained on $n-1$ partitions and the performance of the system is measured on the remaining partition. Each partition is once used as a validation set and the final result is the average over all n tests. A cross-validation with $n = 4$ is depicted in Figure 3.3. Because each element of the original training set is included once into a validation set, the training of the final system always follows the second possibility mentioned above when using cross-validation.

3.3 Feature selection

In the feature extraction process as much information as possible should be gathered from the input pattern. So it would seem to be reasonable to produce large feature vectors. Yet the use of feature vectors of too high a dimensionality leads to the so-called *Curse of Dimensionality* [48]. This phenomena can be explained through the following example [43]. Consider a space partitioned into regularly arranged cells. By increasing the dimension, the number of cells increases exponentially. In a modeling problem the distribution of the data is estimated and this can only be done reliably if the space occupied by the data is well sampled. As the number of

cells increases exponentially, also the number of patterns necessary for good sampling increases with the number of dimensions. So the use of too many features leads to poorly estimated distributions.
A way to overcome the *Curse of Dimensionality* is to start with a set of candidate features and to reduce them. It may also be appropriate to transform the feature vector before the reduction. This approach is discussed in the next section. Many ways exist to reduce a given set of features [50], and they all need an objective function to validate the quality of the considered subset of the features. In Subsection 3.3.1 approaches using a validation set are described. Objective functions based on the analysis of the trained Hidden Markov Models are discussed in Subsection 3.3.2 and in Subsection 3.3.3 an approach is introduced where the results of the different feature sets are simulated.

3.3.1 Feature selection using validation set

To validate the quality of a feature set, often a validation set is used on which the recognition rate is measured. With this objective function the selection of the best subset of features becomes a search problem. An optimal approach is to calculate the objective function for all possible subsets. This approach will be denoted as exhaustive search in the following and is only feasible for a rather small number of features. Branch & Bound algorithms [69] use a priori knowledge to reduce the search space. However, there is often no such a priori knowledge available and a suboptimal approach, such as Sequential Forward/Backward Search [15], Floating Search [74] or Oscillating Search [86], is used.
In this thesis we use a modified version of the “plus 1 - take away 1” search algorithm [74]. The original algorithm is changed so that a feature is only added or removed from the working feature set if the objective function is increased by that action. The “backward” version of the used search algorithm is given in Table 3.3 and will be simply denoted as the backward feature search in the following chapters. The previously mentioned objective function f is the performance of the system on the validation set using the considered feature set s. As in the floating search method [74] the feature to be added or removed is not fixed and there are conditional inclusions and exclusions of features depending on the values of the objective function. In contrast to the floating search the objective function of the working feature set does always increase. This leads to fewer search steps but may lead the search into a local minimum. However, because we only have few features in the systems described in Chapter 2 the risk of such a stop is rather small. Another difference to the floating search described in [74] is that in each step one feature is added or removed at most. There is also a “forward” version of the method shown in Table 3.3. The only difference to the “backward” version is that s is initially set equal to the empty set, i.e. the search is started with an empty set of features. This version of the algorithm will be denoted as the forward feature search in the following.

3.3.2 Feature selection using HMM parameters

A trained HMM contains a large amount of information which may be used to determine how much a single feature contributes to the recognition process, i.e. how important a feature is. This may be done by analyzing the estimated variance and mean values of the distributions

Input: The set of all features s_{all} and an objective function f.
Output: A good-performing feature subset s.

```
$s$ is a set of features and initially equal to $s_{all}$;
while($f(s)$ increased since last loop)
    bestperformance:= $f(s)$;
    bestset:= $s$;
    for all features $i$ in $s$
        $s_{new}$:= $s$ without feature $i$;
        if($f(s_{new}) >$ bestperformance)
            bestperformance:= $f(s_{new})$;
            bestset:= $s_{new}$;
    $s$:= bestset;
    for all features $i$ not in $s$
        $s_{new}$:= $s \cup \{$feature $i\}$;
        if($f(s_{new}) >$ bestperformance)
            bestperformance:= $f(s_{new})$;
            bestset:= $s_{new}$;
    $s$:= bestset;
return $s$;
```

Table 3.3: Backward feature search algorithm used in the experiments

corresponding to that feature (note that there are no covariance terms in the Gaussian distributions used in the classifiers described in Chapter 2). To keep the descriptions of the quality measures as simple as possible only two states s_1 and s_2 of a HMM and only single Gaussian distributions are considered in the following. The mean values of a feature f of these states are denoted as $\mu_f(s_1)$ and $\mu_f(s_2)$ and the variances as $\sigma_f(s_1)$ and $\sigma_f(s_2)$. If the difference of the mean values $\mu_f(s_1) - \mu_f(s_2)$ of the distributions of the two states is much larger than the two corresponding standard deviations $\sqrt{\sigma_f(s_1)}$ and $\sqrt{\sigma_f(s_2)}$ for a feature f, then the distributions are very different. In this case a clear assignment of a feature vector to one of the two states is possible, i.e. the feature allows the identification of the state which produced the feature vector. In the recognition process the sequence of states that produced the feature vector sequence is searched. A clear assignment of a feature vector to a state is very favorable for this search, so a feature is important for the recognition process when the difference of its means are large relative to its standard deviations. Each feature quality measure applied in this thesis is based on the above mentioned considerations.
The following feature quality measures were applied:

- **Fisher criterion**: For all pairs of states of the HMMs (s_i, s_j) the term

$$fisher_{ij} = \frac{(\mu_f(s_i) - \mu_f(s_j))^2}{\sigma_f(s_i) + \sigma_f(s_j)} \tag{3.1}$$

is calculated. The quality of the feature f is then defined as the sum of the terms of all possible pairs of different states which is the same as $\sum_i \sum_{j \neq i}(fischer_{ij})$. In the Fisher criterion all states are compared with each other regardless of whether they belong to the same HMM or whether they are at the same position in the HMMs. The application of the Fisher criterion to HMMs was introduced in [53].

- **Local**: This measure is specially designed for HMMs with linear topology. The linear topology is one of the most prominent HMM architectures used in handwriting recognition. Instead of comparing all pairs of different states, as in the Fisher criterion, only neighbor states, i.e. states which are neighbors in the linear structure of the HMM, are compared. This means that the same term as in equation 3.1 is used, yet the sum is only taken over all pairs of states that are neighbors. The motivation of this measure as follows. From each state in an HMM with linear topology there is only a transition to itself and to the next state. So for the assignment of a feature vector to a state it must only be established whether it belongs to the actual state or to the next state and only the distributions of such neighbor states must be different to select the correct assignment. Because this measure takes the linear topology of the HMMs into account, it may produce superior results than the Fisher criterion. As a drawback of the Local measure it must be noted that states of different HMMs are not compared.

- **Root Local**: This is the same as the Local measure with one modification. Instead of the term calculated in equation 3.1 the square root of this value is used. By applying a square root operation to the individual terms, large values are not that strongly weighted as when using the Local measure, and so the Root Local Measure may be more robust to outliers.

Not all HMMs have the same importance for the recognition system. Consider for example the HMMs for the two characters "e" and "X". The character "X" is so rare that it is almost irrelevant if the corresponding HMM is well trained. A poor HMM for the letter "e" may significantly lower the performance of the system. To take this consideration into account, the above introduced measure are weighted by the occurrences of the characters in the training set. In case of the Fisher criterion the term of equation 3.1 is multiplied by the product of the number of occurrences of the two characters to which the two states belong. In the case of the two other measurements the compared states always belong to the same HMM and so term of equation 3.1 is only multiplied by the number of occurrences of the character corresponding to that HMM.
When using Gaussian mixture distributions instead of single Gaussian distributions the term of equation 3.1 is calculated for each pair of Gaussians where the first Gaussian belongs to the first state s_1 and the second Gaussian to the second state s_2. The final value of $fisher_{ij}$ is then the weighted sum of the terms for all those pairs of Gaussians. The weight of a pair of Gaussians is the product of the weights of the Gaussians in the corresponding distributions.
Using one of the above described measures, the features may now be sorted according to their quality. The final selected feature set contains the m features with the highest quality. The best choice for m is not obvious. One way to choose m is to search for large gaps in the vector containing the ordered quality measures (e.g. if this vector is [1.0,0.9,0.2,0.1] a good choice for m may be 2). In this thesis we applied a different approach. For all possible choices of m, we tested the resulting feature set on a validation set. The value of m which leads to the highest

recognition rate on the validation set is selected. Alternatively the test of the feature sets could also be done on the training set.

3.3.3 Feature selection using simulations

In this Subsection a method is introduced which simulates the recognition results of classifiers using different features sets. The simulation is done for all elements of a given validation set and is based on two simplifications. The simulation often produces inexact results, yet it is much faster than the calculation of the correct results.
It was empirically observed that two HMM classifiers using different feature sets but the same HMM topology often have a similar (or identical) optimal assignment of the features vectors to the states of the HMM of the correct class, even when the outputs of the two classifiers differ from each other. This optimal assignment is returned by the Viterbi algorithm (compare Subsection 2.1.3) and will be denoted as an optimal path.
Based on the above mentioned observation the first simplification is the following: It is assumed that the optimal path for a given input sequence and a given HMM is the same for all feature sets. The assumption will be only approximatively true in most applications, but the optimal paths of the HMMs must be calculated once only for all feature sets. Ideally, the optimal paths are calculated with the classifier using the full set of features, because it is reasonable to assume that this set already produces rather good results. For all other feature sets only the score for this path must be re-calculated for each HMM. A similar simplification was proposed in [12].
It was also observed that the HMM with the highest score given one feature set occurs often among the HMMs with very high scores using another feature set. This observation leads to another simplification: We assume that the output of the classifier using any feature set is among the classes with the N best score values obtained with the full set of features. Using this simplification only the scores of HMMs that are in the N-best list of the test with the classifier using the full set of features needs to be calculated. A pattern is regarded as correct if the HMM corresponding to the correct class has a higher score than any other HMM in this list[2].
The two simplifications mentioned above allow us to very quickly calculate simulations of the recognition results for any feature set. Those simulations may be used to calculate estimations of the recognition rate of the classifier for those features. The method requires the testing of the classifier using the full set of features on the validation set producing an N-best list for each pattern. Note that the estimated recognition rate is usually different from the real one and that the quality of the estimation depends on the value of N.
The above described method may be used to calculate an estimated recognition rate for any set of features. So the method can be used in conjunction with classic feature selection methods by using the estimated instead of the real recognition rate as the objective function (compare Subsection 3.3.1). Because of the high speed of the method, the estimated recognition rate was calculated for all possible feature sets, i.e. an exhaustive search was carried out.
In this thesis we used two approaches to select the final feature set from the obtained results:

1. The feature set with the highest estimated recognition rate is selected.

[2]In the experiments N denotes the number of incorrect HMMs compared to the correct HMM, i.e. a $(N+1)$-best list is used.

2. For each feature set size the classifier using the feature set with the highest estimate recognition rate is tested on the validation set. The feature set that achieves the highest real recognition rate on the validation set is selected.

The motivation for the second approach is that the estimated recognition rate depends strongly on the size of the feature set. Especially for small values of N small feature sets often obtain values that are too large. By validating the best feature sets for each feature set size the negative impact of this effect can be abolished.

3.4 Feature transformation

Instead of the selection of a subset of features, like in Section 3.3, the quality of the feature vector may also be increased by transformation[3]. The literature presents several methods to transform the feature vector (see e.g. [48] for details). In our work we used a linear Principal Component Analysis (PCA), the Karhunen-Loeve transformation [48], which we simply denote as PCA. There are three reasons why PCA is an appropriate transformation method for the systems described in Section 2. Firstly, PCA as a linear transformation method is very simple. Secondly, the components of the transformed feature vector are statistically independent. Such independence was implicitly assumed by using diagonal covariance matrices of the Gaussian distributions, and by using such transformed feature vectors the assumption is no longer flawed. Another reason for using PCA is that the feature components are ordered according to their importance. By removing the last n components, the dimensionality of the feature vector may be reduced without losing too much information. The optimal value for n must be empirically determined for each application. Note that the application of feature selection methods to feature vectors transformed by PCA is not very promising, because the non relevant part of the feature vector, i.e. the last components of the feature vectors, are already determined by the PCA. On the other hand it is also not promising to apply PCA to a feature set which was found by a feature selection method, because the most irrelevant features will have already been removed and the dimensionality of the feature vector is already low. In the system of Dr. Alessandro Vinciarelli described in Subsection 2.3 PCA was always applied, because the features used in this system are strongly correlated. For details of the topic of PCA see for example [48, 93] .

3.5 Rejection method

In some applications it is useful to allow the rejection of patterns for which the classifier gives a result with a low confidence. To express the quality of such a system the three terms recognition rate, rejection rate and accuracy are used. The *recognition rate* is the fraction of patterns which are classified correctly and the *rejection rate* is the fraction of patterns which are rejected by the classifier. The *accuracy* of the classifier is the fraction of non-rejected patterns which are classified correctly. In some applications a high accuracy is needed, while a rather high rejection rate is tolerated. Consider for example the application of bank check reading. Even if 20 % of

[3]Feature selection may also be regarded as a transformation, namely a projection

all checks are rejected, the automatic reading is economic, because only a fifth of all checks must be read by a human. It is, of course, very important that the case of a misread remains rare.
As mentioned above, the rejection is based on a confidence measure. The systems described in Chapter 2 output a score value which could be used as this measure. Unfortunately the score value is an unnormalized likelihood value and no strong correlation of the correctness of a result and the score value was observed, even when some normalization methods were applied. In this thesis we use one of the confidence measures introduced in [67] and which is called the dictionary dependent anti-model criterion with duration coefficient. It is based on the score values of the best score $score_{best}$ and the second best score $score_{2nd\ best}$ for each pattern. Therefore a 2-best list of each test pattern must be calculated when using this confidence measure. The confidence of a result for a pattern x is then defined as:

$$conf(x) = \frac{score_{best}(x) - score_{2nd\ best}(x)}{length\ of\ x} \tag{3.2}$$

Here the length of a pattern x is the length of the feature vector sequence corresponding to this pattern. The values of confidence measure calculated using equation 3.2 strongly depend on the used classifiers, so it is reasonable to normalize it. The normalized confidence measure of a pattern is defined as:

$$conf'(x) = \frac{score_{best}(x) - score_{2nd\ best}(x)}{length\ of\ x} \cdot \sum_{y \in T} \left(\frac{length\ of\ y}{(score_{best}(y) - score_{2nd\ best}(y)) \cdot |T|} \right) \tag{3.3}$$

where T is the training set and so the average is calculated over all patterns of the training set[4]. When using the confidence measure of equation 3.3 for several classifiers (compare Subsection 5.3.4 and Section 5.5), the average of the confidence was calculated over all training patterns and classifiers in the experiments. Yet it may also be reasonable to calculate the average for each classifier alone and to normalize the confidence values with the averages corresponding to the actual classifier.
A pattern x is rejected if its confidence value is smaller than a fixed threshold t. The threshold t controls how many patterns are rejected. High values for t lead to high rejection rates, but should also increase the accuracy of the classifier.

[4]In the implementation of the system the average was calculated on the patterns of the test set, because a 2-best recognition on the whole training set is very time consuming. If the test set is large enough and similar to the training set this should lead to similar averages. Note that only score values of the test patterns, and not their class labels are used.

Chapter 4

Ensemble methods

This chapter contains the description of classic and new ensemble methods used in the experiments. In Section 4.1 some properties of well performing ensembles will be discussed and an explanation for using ensembles instead of single classifiers is given. The following section includes descriptions of the classic ensemble methods used in the experiments. The next section presents methods which partition the training set and train the individual classifiers on one or more partitions. In Section 4.4 new ensemble methods based on AdaBoost, one of the classic ensemble methods, will be introduced. The following section contains descriptions of ensemble methods which use feature selection algorithms to select the feature sets for the individual classifiers of the ensemble. In Section 4.6 ensemble methods which work with several base classifiers will be introduced.

4.1 Properties of ensembles of classifiers

In [16] three reasons why an ensemble method may work better than a single classifier are given. These reason are enumerated as follows and illustrated in Figure 4.1:

1. A classifier may be regarded as a point in the space of all possible classifiers. When only little training data is available, many different classifiers that have good performances on the training set may be trained. But only few of them will be good for the recognition of other patterns. It was shown that by combining several of these classifiers, e.g. by voting, we are likely to find a solution that has better generalization properties. (A solution has a high generalization property if the performance on the training set is similar to the recognition rate of new patterns).

2. Many learning algorithms, including the Baum-Welch algorithm, work by performing some form of local search that may get stuck in local maxima of the search space. It may be that the optimal classifier will never be found, because the learning method always stops at a local maximum. An ensemble consisting of the classifiers produced by running the learning method with many different start parameters, e.g. different training sets, may provide a better approximation of the optimal classifier than any of the individual classifier.

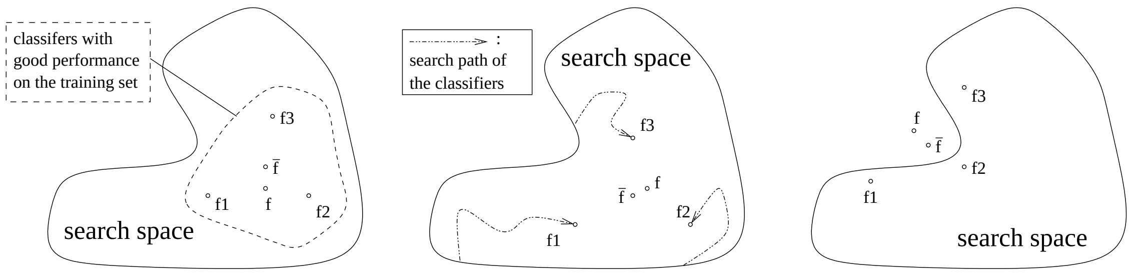


Figure 4.1: Three reasons why an ensemble may work better than a single classifier: f1, f2 and f3 denote the individual classifiers, $\bar{f}$ is the combined classifier and f the optimal classifier. First the "average" of several good performing classifiers on the training set is likely to have a higher generalization property, i.e. is closer to f, than any individual classifier. Secondly the search of the optimal classifier may be get stuck in a local maximum. The third reason is that the optimal classifier f may be outside of the search space.

3. It may be that the optimal classifier can not be obtained by the used classifier architecture and training set, i.e. the optimal classifier is outside of the search space. The combined classifier of an ensemble of classifiers may also be outside the search space of single classifiers, and therefore such a combined classifier may be a better approximation of the optimal classifier.

A well performing ensemble has two key properties: Firstly, the classifiers of the ensemble are diverse and secondly, the individual classifiers have good performance. The goal of any ensemble method is to produce ensembles with both key properties.
A ensemble contains diverse classifiers, when misclassification of patterns has a low correlation across different classifiers (in other words, the recognition rate of a classifier C_i on the patterns misclassified by another classifier C_j should be close to the average recognition rate of C_i). Ideally, independent classifiers are created, but this is almost impossible in real world applications. The diversity of classifiers is crucial, because all of the known combination rules can only increase the performance of single classifiers if they are used with an ensemble of diverse classifiers. For a more detailed discussion of classifier diversity, which is also called classifier ambiguity, see [54].
The second key property of a good ensemble is that the ensemble only contains classifiers whose recognition rates are not much lower than that of the best individual classifier. It is obvious that the recognition rate of an ensemble using a combination rule depends on the performance of its individual members. If the diversity of the classifiers is high than the ensemble usually outperforms any of its individual classifiers. However, if many members of an ensemble have a poor performance they may eventually become dominant over the well-performing classifiers and thus, lower the overall performance of the ensemble.

4.2 Classic ensemble methods

In this section the classic ensemble methods Bagging, AdaBoost, random subspace method and a special version of the architecture variation ensemble method are described. In addition a literature reference and a short description of a fifth classic ensemble method is given.

4.2.1 Bagging

Bagging [8], an acronym for **b**ootstrapping and **agg**regat**ing**, was among the first methods proposed for ensemble creation. Given a training set S of size n, bagging generates m new training sets $S_1, \ldots, S_m$, each of size n, by randomly drawing elements of the original training set, where the same element may be drawn multiple times. If the probability of being drawn is equally distributed over S, as is the case here, then about two thirds[1] of all training elements will be contained in each modified training set S_i, some of them multiple times. Each of the new sets S_i is used to train exactly one classifier. Hence an ensemble of m individual classifiers is obtained from m new training sets.

4.2.2 AdaBoost

Boosting is one of the most popular ensemble methods. In [82] a good overview of this category of ensemble methods is given. The first boosting algorithms were introduced by Schapire in 1989 [80] and by Freund in 1990 [20]. The most prominent algorithm, AdaBoost, was developed in 1995 [21]. Similarly to Bagging, AdaBoost [21] modifies the original training set for the creation of the ensemble. To each pattern of the training set a selection probability is assigned, which is equal for all elements of the training set in the beginning. Then elements for a new training set are randomly drawn from the original training set taking the selection probabilities into account. The size of the new training set is equal to the size of the original one. After the creation of a new training set, a classifier is trained on the new set. Then the new classifier is tested on the original training set. The probability of selecting correctly classified patterns is decreased and the probability of selecting misclassified patterns is increased. During the execution of the AdaBoost procedure the selection probabilities constantly change. Hence, unlike Bagging, where the classifiers are created independently, the classifiers generated by AdaBoost are created dependent on selection probabilities, which in turn depend on the recognition results of previously generated classifiers.
The main aim of AdaBoost is to concentrate the training on “difficult” patterns. Note that the first classifier is trained in the same way as the classifiers in Bagging. The original AdaBoost algorithm only works for two-class problems, but several extensions were proposed and tested in applications. AdaBoost.M1 [21] is a straightforward extension of the two-class AdaBoost that basically regards the multi-class problem as a two-class problem with the two classes “correct class” and “incorrect class”. It is quite obvious that this simplification has its drawbacks. AdaBoost.MH [83] and AdaBoost.M2 [21] work by reducing the multi-class problem to a larger binary problem. Those methods need well adapted base learners to function properly. Another approach is to combine the method of error-correcting output codes [18] with boosting [3, 81]. Here any two-class base classifier may be used. However, a disadvantage of all these approaches is that for problems with a large number of classes, like word recognition, the complexity or the number of the induced two-class problems may be very large. In this thesis the AdaBoost.M1 algorithm was applied, because it works with any base classifier, does not modify the underlying

[1]63.21 %, or 100 % $\cdot (1 - \frac{1}{e})$, for a training set of infinite size

Input: Base classifier C, training set $T = \{e_1, \ldots, e_n\}$, selection probabilities $p(e_i)$ for all elements e_i of T, and the number of classifiers to produce m.
Output: m classifiers and m weights for these classifiers.

```
$p(x)$ is set to $\frac{1}{n}$ for all $x \in T$;
for(i=1;i≤m;i++)
    sample $n$ elements of $T$ using the selection prob. $p(x) \rightarrow T_i$;
    train $C$ on $T_i \rightarrow$ classifier $C_i$;
    test classifier $C_i$ on all elements in $T$;
    error=0;
    for(j=1;j≤n;j++)
        if($C_i$ misclassifies $e_j$)
            error = error + $p(e_j)$;
    if(error>0.5)
        $p(x)$ is set to $\frac{1}{n}$ for all $x \in T$;
        i=i-1;
    else
        $\beta = \frac{1-\text{error}}{\text{error}}$;
        set weight of classifier $C_i$ to $ln(\beta)$;
        for(j=0;j<n;j++)
            if($C$ misclassifies $e_j$)
                $p(e_j) = p(e_j) \cdot \beta$;
        normalize $p(e_j)$, so that $\sum_{j=1}^{n}(p(e_j)) = 1$;
return $C_1, \ldots, C_m$ (with their weights);
```

Table 4.1: AdaBoost.M1 algorithm

classification problem and is easy to implement. The algorithm of AdaBoost.M1 is shown in Table 4.1. AdaBoost.M1 provides weights for each classifier, and the recommended combination of the produced classifiers is weighted voting using those weights. As it will be shown in Chapter 6 AdaBoost.M1 is able to produce good ensembles, yet the combination by weighted voting using the calculated weights may lead to inferior results compared to other combination schemes. In the algorithm shown in Table 4.1, the selection probabilities of all elements are reset to the same value, if the error is higher than 0.5, to avoid the production of poor performing classifiers. Note that the iteration counter is decreased and that the classifier with the high error is replaced. The error calculated for the classifiers is weighted by the selection probabilities of the training patterns.

4.2.3 Random subspace method

In the random subspace method [28] an individual classifier uses only a subset of all features for training and recognition. The size of the subset is fixed and the features are randomly chosen from the set of all features where each feature may only be selected once. So instead of a re-sampling of the training set, as in Bagging, a new reduced feature set is chosen. Note that the

sampling is done without replacement in the random subspace method, but with replacement in Bagging. The individual classifiers are trained on the whole training set.
For the classifiers described in Section 2 the situation is special in the sense that the number of available features is rather low. Therefore, the original random subspace method was slightly modified so that the following property always holds: If the number of classifiers which use feature f_i is denoted by $n(f_i)$, then $\forall(i, j\ |n(f_i) - n(f_j)| \leq 1)$. This means that each individual feature is used in approximately the same number of classifiers. Therefore, all features have approximately the same importance. By means of this condition it is enforced that the information of every feature is exploited as much as possible. By contrast, when choosing completely random feature sets, it is possible that certain features are not used at all. The modification of the algorithm does not have an impact on the probability of a feature set to be selected for one of the classifier, i.e. this probability is the same for all feature sets of the fixed size.
When applying the random subspace method with the classifier introduced in Section 2.2 always 6 of the 9 features were randomly chosen.

4.2.4 Architecture variation

In the architecture variation ensemble method the structure or the parameters of the architecture of the classifier is varied. In a feed-forward neural network, for example, one may change the number of hidden layers or the number of neurons in each layer [73]. For the HMM classifiers described in chapter 2 there are several properties which may be varied:

- Number of training iterations
- Number of Gaussians of the output distributions of the states
- Number of states per HMM
- HMM topology
- Reading direction (left-to-right or right-to-left)

In preliminary experiments it was found out that the variation of the HMM topology and the reading direction have a very positive impact.
Four different HMM topologies, namely linear, Bakis, semi-jumpin and semi-jumpout, were considered. The structure of these topologies are depicted in Figure 4.2. The linear topology is the original topology used in the classifiers described in Chapter 2. In the Bakis topology the transition from one state to the second next state is also possible. This topology allows more flexibility in the decoding process by skipping certain states. The HMM models used in our base classifier do not include any ligature states[2]. But the transition from one character to the next is often context dependent. Therefore, if certain character pairs are not sufficiently well represented in the training set, misalignments at the beginning and at the end of a character model during decoding may be expected. To account for this kind of problem, the semi-jumpin and semi-jumpout topologies shown in Fig. 4.2 were introduced. Here the first or last $\frac{n-2}{2}$ states

[2]Here the term ligature denotes a connection stroke between two consecutive characters

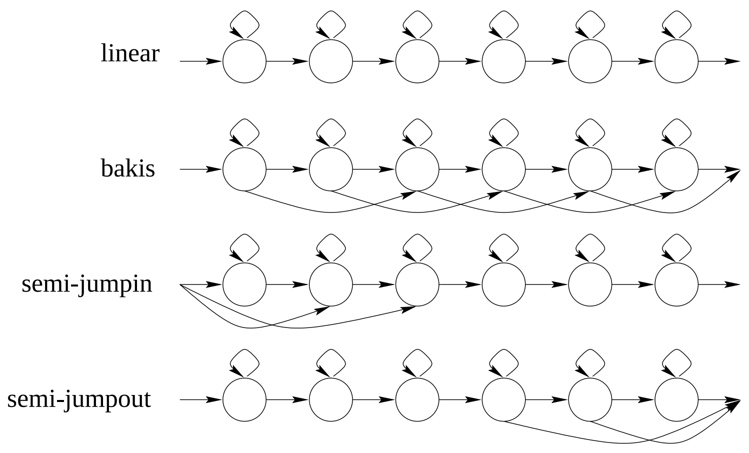


Figure 4.2: Different HMM topologies for a small HMM with 6 states.

of the linear model may be skipped (with n denoting the total number of states of the considered HMM).
Normally the columns of a word image are read form left to right. Another possibility is to read them from right to left. Because the Viterbi search used in the decoding phase of the classifiers described in Chapter 2 is a suboptimal procedure that prunes large portions of the search space, the results of a forward and a backward scan of the word are not necessarily the same. To implement a right-to-left scan of the image, only the concatenation of character HMMs needs to be changed appropriately.
Apparently, left-to-right as well as right-to-left scanning can be combined with any of the topologies shown in Fig. 4.2. Therefore, a total of eight different classifiers can be generated. Each of these classifiers is trained on the full training set.

4.2.5 Half and half Bagging

Half and half Bagging was introduced in [9]. In this ensemble method the training sets of the classifiers consist half of training elements misclassified by the ensemble of already produced classifiers. The other half of the training set contains only elements correctly classified by this ensemble. As half and half Bagging is only used for a few experiments in this thesis, no further description of this ensemble method is given. For details see [9]. The reason that half and half Bagging is not widely used in the experiments is that in the first experiments the results of this ensemble method were not better than the results of the other classic ensemble methods, but the other methods were more often applied in the literature.

4.3 Ensemble methods using a partitioning of training set

In this section ensemble methods using a partitioning of the training set are introduced. These ensemble methods will be denoted as partitions based ensemble methods. The basic algorithm of the partitions based ensemble methods as follows. First, the whole training set is split into several partitions. Then each classifier is trained on one or several of these partitions. Note that the training set of a classifier always includes all or no elements of a partition. The training sets of the classifiers may overlap, but the overlapping always contains all elements of one or several partitions. To design a partitions based ensemble method two key parameters must be set:

1. The partitioning algorithm (including the number of partitions).

2. The assignment of the partitions to the training sets of the individual classifiers.

The most simple partitioning algorithm is random partitioning where the training set is randomly divided into n equally sized parts. In preliminary experiments it was observed that such partitioning is not very effective.
In this thesis an ensemble method is used where the partitioning is based on a clustering of the words according to their writing style. In this method the whole training set is first clustered into clusters of words with similar writing style. Each of these clusters is then regarded as one partition and each partition is used for the training of one classifier. If only a few writers contributed to the training set and if the identity of the writers of the words of the training set is known, each cluster may simply be composed of all words of a writer and no clustering algorithm must be applied. This approach is not feasible when the training set consists of words from a large number of writers and if the sizes of the sets of contributed words vary heavily among those writers. In this case some features of the words must be defined, so that a clustering algorithm may be applied. As we are using the data of the IAM database, we always have complete forms of handwritten text at our disposal (compare Section 2.4). Therefore the features were extracted from the whole form and all words of the form belong to the same cluster. The features used for the clustering are the following:

- **Components per word (CPW)**: The average number of connected components per word is calculated over the whole form. All the components with a bounding box smaller than 500 pixels are ignored, as they are likely to be noise. Single characters, like punctuation, and numbers are also not considered. A value of one for this feature corresponds to a complete cursive handwriting.

- **Words per component (WPC)**: This is simply the inverse of CPW.

- **Character width (CW)**: The average width of the characters is calculated over the whole form. Again single characters are not considered and components with smaller bounding boxes than 500 pixels are ignored. To calculate the width of the characters the bounding boxes of the words are used and so the white spaces between the words are not taken into account.

The first and the second features contain in principle the same information, but may lead to different clustering results. The third feature may be used together with the first or the second feature but to use the first two features together is not promising as they are redundant. So there are five possible feature sets: All features alone (CPW,WPC,CW) and the two sets containing CW and one of the other features (CPW/CW and WPC/CW). In the experiments a k-means clustering [63] with the mentioned features was executed where the features were first linearly normalized so that the mean of the features was 0 and the standard deviation was 1.

```
Input: Base classifier $C$, training set $T$ of size $n$, and number of classifiers to produce $m$.
Output: $m$ classifiers.

$p(x)$ is set to $\frac{1}{n}$ for all $x \in T$;
train $C$ on $T \rightarrow$ classifier $C_1$;
for(i=2;i $\leq m$;i++)
    test classifier $C_{i-1}$ on $T$;
    modify $p(x)$ using information derived from the results of $C_1, \ldots, C_{i-1}$ on $T$; (*)
    sample $n$ elements of $T$ using the selection prob. $p(x) \rightarrow T_i$;
    train $C$ on $T_i \rightarrow$ classifier $C_i$;
return $C_1, \ldots, C_m$;
```

Table 4.2: Basic algorithm of the new boosting ensemble methods

4.4 New boosting algorithms

As mentioned in Subsection 4.2.2 most multi-class AdaBoost extensions do not work well with problems with a very large number of classes, like the word recognition task considered in this thesis. The AdaBoost version used for the experiments of this work, the AdaBoost.M1, is easy to implement, but by regarding the multi-class problem as a two-class problem with the two classes “correct” and “incorrect” a lot of information is lost. In this Section three new boosting algorithms are presented which work well for classification tasks involving a large number of classes and which take all information into account. They are all derived from the original AdaBoost algorithm.

4.4.1 Basis algorithm of the new boosting methods

All new boosting methods work with a distribution p of selection probabilities over the training set, similarly to AdaBoost [21]. So $p(x)$ is the probability than an element x of the original training set T is selected when sampling elements for the training set of a new classifier. The basic algorithm of all new boosting ensemble methods is given in Table 4.2.

This algorithm is very similar to AdaBoost where the selection probabilities are also adapted according to the results of the previously created classifiers on the training set. The sampling is also done with replacement. Note however, that in contrast to the AdaBoost algorithm, the first classifier is trained on the full training set. The difference of the three new boosting methods is the modification of the selection probabilities $p(x)$ (line with star sign in Table 4.2). Contrary to AdaBoost, the new boosting algorithms don’t produce any weights for the classifiers and so no constraints on the combination scheme to be used are imposed.

In the following the details of the novel selection probability modification schemes of the proposed boosting methods and their underlying ideas are presented.

4.4.2 Simple probabilistic boosting (SPB)

The main theory of this boosting algorithm is to set the selection probability $p(x)$ of a training element x at proportion to the probability $e(x)$ that the ensemble of classifiers will misclassify elements similar to element x[3]. Similarly to AdaBoost, the selection probability of "hard" elements, i.e. elements which are likely to be misclassified, are set to a high value. It was decided to make the selection probabilities linearly dependent on the misclassification probability, because this approach is simple and the optimal function $o : e(x) \rightarrow p(x)$ is unknown anyway. The question remains of how to calculate $e(x)$ from the results of the classifiers $C_1, \ldots, C_m$ that were already created. Four different functions are considered in this thesis where the ensembles are always combined by the voting combination scheme:

- If the ensemble of the classifiers $C_1, \ldots, C_m$ classifies x correctly then $e_1(x)$ is set equal to 0, otherwise it is set equal to 1. Although correct for the actual ensemble this is a poor estimation, because it does not reflect how many classifiers classified x correctly even though the ensemble as a whole misclassified x.

- Let $k(x)$ be the number of classifiers of $C_1, \ldots, C_m$ which output the correct class label for x. Then $e_2(x)$ is 0 if the ensemble of the classifiers $C_1, \ldots, C_m$ classifies x correctly. Otherwise it is $\frac{1-k(x)}{m}$. In contrast to $e_1(x)$, we also take here the information of the number of classifiers into account which had a correct output although the ensemble output was incorrect.

- Let $k(x, y)$ be the number of occurrences of label y in the output of the classifiers $C_1, \ldots, C_m$ for training element x. The probability that *one* classifier of the ensemble outputs y is therefore $\frac{k(x,y)}{m}$. The probability of the ensemble to be wrong is then defined as the probability that the correct label c is not output more often than all other labels y, given the output probabilities $\frac{k(x,y)}{m}$ and an ensemble of size m. This is the most exact estimation of the probability of the misclassification of elements similar to element x, but may be very difficult to calculate. Consequently, two approximations will be considered as follows:

  – Only the label i of the incorrect class that occurs most frequently in the output of the classifiers $C_1, \ldots, C_m$ and the correct label c are considered and the probability $e_3(x)$ is calculated that i will occur more frequently than, or as often as, c in the output of an ensemble of size m.

  – The output probabilities $\frac{k(x,y)}{m}$ are used and a number of simulations of the output of an ensemble of size m are run. The fraction of simulations where the correct label c was not the most frequent label is the error probability $e_4(x)$. In the experiments of this thesis the number of simulations was set to 1,000.

Note that under $e_3(x)$ and $e_4(x)$ the results of $C_1, \ldots, C_m$ are only used for the calculation of the probabilities $\frac{k(x,y)}{m}$ that one classifier outputs label y for training element x. This means that $e(x)$ may be less than 1 even when the actual ensemble misclassifies the element.

[3]Note that a well trained classifier should not only classify training elements correctly, but also patterns which are similar to the training elements.

On the other hand $e(x)$ may also be larger than 0 for an element x which is classified correctly by the ensemble. The reason why the actual results are not directly used is that we are not interested in the optimization of the ensemble for the training set, but in the optimization for elements similar to the ones in the training set. So by using the results only indirectly we may avoid overfitting effects.

For a better understanding of the functions $e_1(x), e_2(x), e_3(x)$ and $e_4(x)$ consider the following example: Assume four classifiers were produced for a classification problem with three classes X, Y and Z. The correct class of the test pattern t is X and the output of the classifiers for t are X, Y, Y and Z. The value of function $e_1(t)$ is equal to 1, because the wrong label Y appears more often than the correct label X. The value of function $e_2(t)$ is equal to 0.75, because one of the four labels in the output is correct. For $e_3(t)$ and $e_4(t)$ the value of function k must be calculated first. Here, $k(t, X) = k(t, Z) = \frac{1}{4}$ and $k(t, Y) = \frac{1}{2}$. The value of $e_3(t)$ is the probability that Y appears more often than X using the output probabilities of the labels given by k. Here $e_3(t) = 0.81640$. For $e_4(t)$ the same values for k as for $e_3(t)$ are used for the simulations, and $e_4(t) = 0.846$ was obtained for this example.

4.4.3 Effort based boosting (EBB)

The main theory of this method is that the selection probabilities $p(x)$ of the training elements are regarded as resources and that those resources should be distributed to, or shared among, the elements, for which a high value of $p(x)$ has the largest positive effect on the ensemble performance. Unlike in SPB when using functions $e_3(x)$ or $e_4(x)$, the results of the ensemble of already produced classifiers are directly used to optimize the ensemble performance on the training set.

To understand the EBB algorithm two new terms must be introduced: The effort $\text{ef}(x)$ and the required resources $r(x)$ of a training element x. The effort $\text{ef}(x)$ of a training element x is defined as the average number of instances of x required to be present in the training set so that the trained classifier makes a correct decision for x. We can estimate $\text{ef}(x)$ as follows:

$$\text{ef}(x) = \frac{\text{total number of instances of } x \text{ in all training sets of all classifiers}}{\text{number of correct decisions for } x \text{ made by all classifiers}} \tag{4.1}$$

As mentioned before, the performance of the ensemble should be optimized. So the resources, i.e. the selection probabilities, should be distributed to the elements for which a correct voting result of the ensemble is achieved with minimum effort, i.e. with as few resources as possible. The quantity $r(x)$ is defined as the average total number of required training instances, so that the ensemble consisting of the already produced classifiers and some new classifiers outputs the correct result for x. Moreover, $n(x)$ is the number of classifiers with the correct result for x needed to be added to the actual ensemble to get a correct voting result for x. Because we need to train $n(x)$ classifiers outputting the correct result for x to get a correct voting result of the ensemble and because we need in average $\text{ef}(x)$ training instances in the training set of each of this classifier, we can calculate $r(x)$ by the equation $r(x) = n(x) \cdot \text{ef}(x)$. If $n(x)$ is larger than the number of classifiers which are still to be added to the ensemble, then it is impossible to achieve a correct voting result of the final system for x and $r(x)$ is set to infinity.

First classifier

	# instances in T_1	result of C_1	ef	r
a (class A)	1	A	1	0
b (class B)	1	A	∞	∞

Second classifier

	# instances in T_2	result of C_2	ef	r
a (class A)	0	A	0.5	0
b (class B)	2	B	3	3

Table 4.3: Example of values of the effort ef and the required resources r of two training elements.

Consider the following example illustrated in Table 4.3. The training set consists of the elements a and b which belong to class A and B, respectively. The first classifier is trained with both a and b and tested on the training set. Element a is classified correctly, but b isn't. The second classifier is then trained on two instances of b and when tested on the training set it classifies both a and b correctly. After the creation of the second classifier the effort of a is the number of instances in the training sets of the classifiers which is equal to 1 divided by the number of classifiers with correct result which is equal to 2, so $\text{ef}(a) = \frac{1}{2}$. Similarly, we get $\text{ef}(b) = 3$. Because the voting result for a is correct, $n(a)$ and $r(a)$ are both 0. For b, we need one more classifier which outputs the correct result to achieve a correct voting result, so $n(b) = 1$ and $r(b) = 3$.
In Table 4.4 the method for setting the selection probabilities $p(x)$ of the training elements is shown. All elements with $r(x) > 0$ are considered in ascending order of $r(x)$ and the selection probabilities are set to the value $\text{ef}(x) \cdot \alpha \cdot \frac{1}{n}$ until the sum of the selection probabilities reaches 1. Note that with this equation the average number of instances in the training set of the new classifier is $\text{ef}(x) \cdot \alpha$. If the sum of the selection probabilities gets larger than 1 for a set of elements $\{x_1, \ldots, x_n\}$ with the same value of r then the selection probabilities of all those elements is set equal to the value $c \cdot \text{ef}(x) \cdot \alpha \cdot \frac{1}{n}$ where c it the constant which leads to a sum of the selection probabilities equal to 1. If this sum doesn't reach the value 1 for the elements with $r(x) \neq \infty$, then the rest of the sum of selection probabilities is equally distributed among the elements with $r(x) = \infty$[4]. However, often the selection probabilities of many elements with large, finite values of r are not set by the algorithm, i.e. the selection probabilities of these elements remain 0. Note that in the algorithm of Table 4.4 the elements with $r(x) = 0$, i.e. the elements which are classified correctly by the actual ensemble, also have a selection probability of 0. The algorithm separates the elements of the training set in three groups. The first group contains elements which are already correctly classified by the ensemble. The second group contains elements which are misclassified by the ensemble, but for which a correct voting result of the ensemble may be achieved by including only rather few instances in the training sets of new classifiers. The third group consists of elements which are also misclassified by the ensemble and for which a large number of instances in the training sets of new classifiers would be needed

[4]Alternatively the elements with infinite effort, i.e. elements which were never classified correctly by any classifier, could be excluded, because it's very likely that they are outliers.

Input: Training set T of size n, and required resources $r(x)$ and effort $\mathrm{ef}(x)$ for all elements $x \in T$.
Output: Selection probabilities $p(x)$ for all elements $x \in T$.

```
$p(x)$ is set to 0 for all elements of $x \in T$;
sort all elements $x$ in ascending order according to the value of $r(x) \rightarrow$ list $L$;
remove all elements $x$ with $r(x) = 0$ from $L$;
totalprob=0;
while(totalprob<1 and $\min_{x \in L} r(x) \neq \infty$)
    $S$ holds all elements of $L$ with the smallest value of $r$;
    remove all elements in $S$ from $L$;
    totalneeded = 0;
    for(all elements $x$ in $S$)
        totalneeded= totalneeded + $\mathrm{ef}(x) \cdot \alpha \cdot \frac{1}{n}$;
    fraction = 1;
    if(totalneeded>(1-totalprob))
        fraction = (1-totalprob) $\cdot \frac{1}{\text{totalneeded}}$;
    for(all elements $x$ in $S$)
        $p(x)$=$\mathrm{ef}(x) \cdot \alpha \cdot \frac{1}{n} \cdot$ fraction;
    totalprob = totalprob + totalneeded $\cdot$ fraction;
if(totalprob<1)
    for(all elements $x$ in $L$)
        $p(x)$ = (1-totalprob) $\cdot \frac{1}{|L|}$;
return $p(x)$ for all elements $x \in T$;
```

Table 4.4: Calculation of the selection probabilities in EBB

to obtain a correct voting result. The main idea of the algorithm is to train exclusively on elements of the second groups. This is because elements of the first group are already correctly classified and to achieve a correct result for the elements of the third group too many training instances would be needed. As mentioned previously, unlike SPB, the focus of this method is on the optimization of the actual ensemble, so it is not necessary to include correctly classified patterns in the training set.

Because $\mathrm{ef}(x)$ is the number of training instances of x needed on *average* for the correct classification of x by the new classifier, the training set should contain more elements than $\mathrm{ef}(x)$ to ensure that the classifier correctly classifies x. This is the reason why the parameter α is introduced. The larger the value of α, the more likely the elements with $p(x) > 0$ are classified correctly by the created classifier. On the other hand the smaller α is, the higher the number of elements that have positive selection probabilities. The optimal value of α is application dependent and must be determined by experiment.

4.4.4 Simple probabilistic boosting with effort (SPBE)

The idea of this algorithm is to train on elements for which a correct voting result may be easily achieved and for which, at the same time, the probability of a misclassification of similar elements by the ensemble is high. It uses the concepts of the methods described in the two last subsections, and therefore may be regarded as a sort of combination of them.
SPBE is identical to SPB with one exception. The only modification to SPB is a different procedure for the calculation of the selection probability $p(x)$ which is given by the following equation:

$$p(x) \propto e(x) \cdot (1 + \frac{\alpha}{\mathrm{ef}(x)}) \tag{4.2}$$

The effect of α is as follows. The higher the value of α, the higher the selection probabilities of the elements with low effort values, but the less selection probability is "distributed" to elements with higher effort values. Similarly to EBB, there is a tradeoff between the number of elements to be emphasized and the likelihood of the correct classification of an element when emphasizing it. The optimal value of α is also application dependent and must be experimentally determined. In the experiments of this thesis the special case of SPBE where α is set to a very high value is used. The equation 4.2 can then be simplified to the equation given below:

$$p(x) \propto \frac{e(x)}{\mathrm{ef}(x)} \tag{4.3}$$

An advantage of this special case is that the equation no longer contains the parameter α.

4.5 Feature selection ensemble methods

One of the most successful ensemble methods is the random subspace method introduced in Subsection 4.2.3. In this ensemble method each individual classifier uses its own subset of the given features for both training and recognition. The subsets contain elements randomly selected out of the whole feature set, where the size of the feature subset is usually the same for each classifier. The main goal of the algorithm is not to select good subsets of features, but to create diverse classifiers to get a good performance of the ensemble of the classifiers.
The key idea of the new method proposed in this section is not to select subsets of features randomly, but to apply an algorithm that selects well performing subsets of the features for the individual classifiers of the ensemble. In Subsection 4.5.1 the basic version of the ensemble methods which use underlying feature selection algorithms is described. The following subsection introduces an extension of the basic version which allow the use of several feature selection algorithms. Subsection 4.5.3 contains the description of some ensemble methods which work with the results of an exhaustive feature search. In Subsection 4.5.4 a heuristic version of the methods introduced in the other subsections is presented.

4.5.1 Methods using one underlying feature selection algorithm

The algorithm of the proposed methods is summarized in Table 4.5. By sequentially calling the underlying algorithm some well performing feature sets are expected to be found. In principle

Input: A feature selection algorithm $S(f, L)$, where f is the objective function and L a list of feature sets, which are prohibited as result; n is the desired number of classifiers.
Output: Classifiers $C_1, \ldots, C_n$.

```
for(i:=1;i≤n;i++)
    feature set $\mathrm{fs_i} = \mathrm{S(f, \{fs_1, \ldots, fs_{n-1}\})}$;
    $C_i$ is a new classifier which uses the features set $\mathrm{fs_i}$;
return $C_1, \ldots, C_n$;
```

Table 4.5: Basic algorithm of ensemble methods using feature selection

any known algorithm for feature selection can be used. The only modification needed is that the feature selection algorithm should not return the same set of features multiple times. Hence the particular feature selection algorithm has, as an additional parameter, a list of feature sets that are forbidden to be the result of the algorithm. Each of the feature sets found by the underlying algorithm is used by a classifier. In Table 4.5, f denotes the objective function to be optimized by the feature selection algorithm. That is, the feature selection algorithm tries to find the set of features that results in the best value of f. In this thesis only feature selection algorithms which use a validation set to measure the performance of a classifier are applied.
When using the algorithm of Table 4.5 with any objective function known from the literature, for example, the minimization of the classification error rate on a validation set, the selection of the new feature set is independent of the sets selected before. That is, function f is optimized without taking any of the sets selected before into account. This behavior may be not desirable, because of the following reasons. Firstly, some feature selection algorithms tend to only produce results that are near to a global or local maximum of the objective function, so that very similar feature sets are selected, which also cause the classifiers to be similar. However, this contradicts the requirement that an ensemble should consist of diverse classifiers. Secondly, the available information is not fully exploited. In this thesis we propose to use the objective function f in such a way that the information about the feature sets already selected is actually used. Therefore we use the performance of the ensemble consisting of the already produced classifiers plus the classifier using the actually considered feature set as the objective function f. In the following four possibilities of measuring the performance of the ensemble are discussed.

- **Objective function f_1:** This function is the recognition performance, on the validation set, of the ensemble using the voting combination scheme. Function f_1 is based on the mere percentage of patterns correctly classified by the ensemble. It doesn't take into account any additional available information about the misclassified patterns. In particular, it doesn't consider how many classifiers made a correct decision although the ensemble as a whole gave a wrong vote. This kind of information is included in the next objective function.

- **Objective function f_2:** This function is defined as $f_2 = f_1 + (1 - f_1) \cdot p$, where p is the average percentage of classifiers that correctly classify the patterns which are misclassified by the ensemble. As an example, consider the situation in Table 4.6. Assume one classifier has already been generated, two classes are to be distinguished, and there are three patterns in the validation set. Evaluating a new set of features, i.e. a new classifier, we observe that voting only classifies pattern 1 correctly, so f_1 is $\frac{1}{3}$. (We assume that an input is rejected if a tie between two or more classes occurs). For pattern 2 and 3 one classifier out of two is correct, so $p = 0.5$ and $f_2 = f_1 + \frac{2}{3} \cdot \frac{1}{2} = \frac{2}{3}$.

There are three reasons supporting using this function instead of f_1. Firstly, even if an ensemble outputs a wrong class for a pattern, the output class may become correct, when adding more classifiers to the ensemble. Consider again Table 4.6. Here patterns 2 and 3 are rejected, i.e. they are not correctly recognized, because for both classes there is one vote. When we add a third classifier which produces the same output for the three patterns of the validation set as either the first or the second classifier, the ensemble has a correct voting result for either pattern 2 or 3, i.e. the performance of the ensemble will improve. Function f_2 takes this potential of a wrong voting decision to become correct into account. Secondly, at the beginning of the algorithm more diverse classifiers are produced. When using f_1 then the second classifier is selected so that its results are as similar as possible to the results of the first classifier. This leads to a decrease of classifier diversity, which can be prevented by using f_2. The third reason is that objective function f_2 is less susceptible to overfitting than f_1. In preliminary experiments it was observed that when using f_1 classifiers with poor individual performances were chosen, because the voting performance of the ensemble on the validation set slightly increased when including that classifier. Yet on the test set the ensemble performance decreased. When using f_2 we also take the individual classifier performance into account (in the term p). So good performing classifiers are favored and a classifier with poor performance is only chosen if the voting result would significantly increase by adding it. Consider the example in Table 4.7. The two existing classifiers each classify six out of eight patterns correctly. The voting result is correct for four patterns. By adding the new classifier the correct voting result increases from four to five patterns although the new classifier only classifies one out of eight patterns correctly. It is quite unlikely that the performance of the ensemble will increase on any set other than the given validation set. Calculating f_2 for the first two classifiers, we get $p = \frac{1}{2}$ because one classifier is always correct for the patterns 1,2,7 and 8 which are misclassified by the ensemble. Therefore $f_2 = f_1 \cdot p \cdot (1 - f_1)$=$\frac{1}{2} + \frac{1}{2} \cdot \frac{1}{2} = \frac{3}{4}$. After adding the new classifier f_1 increases to $\frac{5}{8}$, as mentioned before. For the patterns 2,7 and 8, which are still misclassified by the ensemble, only one out of three classifiers is correct, so $p' = \frac{1}{3}$. This leads to $f_2' = f_1' \cdot p' \cdot (1 - f_1')$=$\frac{5}{8} + \frac{1}{3} \cdot \frac{3}{8} = \frac{3}{4}$. So the value of f_2 doesn't increase when we add the third classifier.

- **Objective function** f_3**:** This function is similar to f_2, from the global point of view. It takes the number of correctly classified patterns under the voting scheme, i.e. the value of function f_1, and adds a fraction p of the number of misclassified patterns. For each misclassified pattern the number of votes n_c for the correct class c and the number of votes n_w for the not correct class w with the most votes are determined. (The case $n_c = n_w$ indicates a tie.) For pattern 2 in the example of Table 4.6 the correct class c is B and the class w is A. Because each of the classes A and B have one vote, $n_w = n_c = 1$ holds.

	results of the validation patterns		
	1	2	3
correct class	A	B	A
results of existing classifier	A	A	A
results of new classifier	A	B	B

Table 4.6: Example for the description of the different objective functions

Once n_c and n_w have been determined, the probability is calculated that there are $n_c - n_w + 1$ more votes for the correct class c then for class w in the results of the remaining classifiers (so that the correct class c is the result of the ensemble when using the voting combination scheme). The fraction p is then defined as the average of the above mentioned probability for all misclassified elements. The probability of a classifier outputting class i is estimated by the number of votes in the actual ensemble for this class, divided by the number of classifiers. Similarly to f_2, this function takes into account the potential of a wrong voting decision to become correct, but it uses a different, more complex mathematical procedure (at the cost of an increased computation time). Again consider the situation in Table 4.6. For f_3 the number of classifiers to be generated is important. The probability of a classifier outputting the correct result for patterns 2 and 3 is estimated to be 50 %. If only two classifiers are produced then $f_3 = f_1 + 0 \cdot (1 - f_1) = \frac{1}{3}$, because for the pattern 2 and 3 there is no chance that the correct result receives more than one vote, as no more classifiers will be added. In case of a total of three classifiers the chance is 50% for each pattern 2 and 3, so for this scenario $f_3 = f_1 + \frac{2}{3} \cdot \frac{1}{2} = \frac{2}{3}$.

- **Objective function f_4:** This function is similar to f_3. It adds the average estimated probability that an incorrect voting result may become correct (when adding the remaining classifiers) to the voting performance. In f_3 only two classes are considered per pattern, the correct class and the most frequent not correct class, while under f_4 all classes are considered. Again the probability of a classifier outputting class i is estimated by the number of votes in the actual ensemble for this class, divided by the number of classifiers. The calculation of f_4 proceeds as follows. For the remaining classifiers the results are randomly selected N times using the estimated class output probabilities and the frequency n of the correct class being the final voting result is determined. The average of $\frac{n}{N}$ for all misclassified patterns in then defined as p_4 and finally $f_4 = f_1 + (1 - f_1) \cdot p_4$ is obtained. So the creation of the remaining classifiers is simulated N times and the percentage of the simulations where the correct result became the voting result is counted. The reason for using such simulations is that the calculation of the exact probability is too complex. In the experiments of this thesis, $N = 1000$ was chosen. Similarly to f_3, the value of f_4 also depends on the total number of classifiers.

Note that there is a strong similarity between the four objective functions f_1, f_2, f_3 f_4 described in this section and the four ensemble error functions p_1, p_2, p_3 and p_4 introduced for the simple probabilistic boosting (SPB) ensemble method in Subsection 4.4.2. The ensemble error functions measure the error and the objective function the performance of the ensemble. The first

	patterns of validation set							
	1	2	3	4	5	6	7	8
correct class	A	A	A	A	B	B	B	B
results of classifier 1	B	A	A	A	B	B	A	B
results of classifier 2	A	B	A	A	B	B	B	A
results of new class.	A	B	B	B	A	A	A	A

Table 4.7: Example for possible overfitting when using f_1 objective function

functions, f_1 and p_1, only take the voting result into account. In the second functions, f_2 and p_2, the percentage of correct classifiers for a misclassified element is also considered. In f_3 and p_3 the exact value when only considering the most frequent not correct class and the correct class is calculated. Finally, results of ensembles are simulated in f_4 and p_4. The main difference of the calculation of the objective function f and the error probability p is that for f_3 and f_4, unlike to p_3 and p_4, the results of the actual ensemble are not directly used. The reason for not directly using the results in SPB is to avoid overfitting. Such overfitting effects occur when the recognizer is trained on sets consisting of instances of only few original training samples. As the ensemble methods considered in this section always train the classifiers on the full training set and only varies the feature set, this overfitting avoidance is not needed.
When executing the algorithm of Table 4.5 it may happen that the objective function f is calculated for the same feature set several times. When using a validation set, all patterns of this set must be classified for the calculation of the value of f, which may be very time consuming. It is therefore useful to save all calculated results and to check in each call of the objective function whether the results for the considered feature set were already computed before.
As noted before any feature selection method may be used in conjunction with the proposed ensemble methods. In this work we always use the forward or backward version of the feature search method presented in Subsection 3.3.1.
An ensemble method using feature selection was already introduced in [2]. In this method, a classifier always selects the single feature that leads to the best recognition rate of the whole classifier ensemble. In contrast to the algorithm proposed in this subsection, a feature may only be used by one classifier and the selection of the features is not reversible, i.e. features that have been selected can not be abandoned at a later time.

4.5.2 Extension to several underlying algorithms

As mentioned before, a good ensemble consists of classifiers that are as diverse as possible. One way to create diversity among the classifiers produced by the algorithm of Table 4.5 is to use one of the objective functions introduced in the last section. Another way is to use different feature selection algorithms. By doing this the returned feature sets are likely to be more diverse than in the case of using just one selection algorithm and so it may also be expected that the classifiers using these divers feature sets are also more diverse. The extended ensemble method which employs multiple feature selection algorithms is shown in Table 4.8.
The feature selection algorithms can be randomly chosen or selected according to a special order.

Input: A set of feature selection algorithms $S = \{S_1(f, L), \ldots, S_m(f, L)\}$, where f is the objective function and L a list of subsets, which are prohibited as result; n is the desired number of classifiers.
Output: Classifiers $C_1, \ldots, C_n$.

```
for(i:=1;i≤n;i++)
    select feature selection algorithm CS from set $S$;
    feature set $\mathrm{fs_i} = \mathrm{CS}(\mathrm{f}, \{\mathrm{fs_1}, \ldots, \mathrm{fs_{n-1}}\})$;
    $C_i$ is a new classifier which uses the features set $\mathrm{fs_i}$;
return $C_1, \ldots, C_n$;
```

Table 4.8: Extended version of the ensemble method using feature selection

The best selection strategy obviously depends on the available feature selection algorithms and the application. Note that if we use the objective functions discussed in the last section the order of the selection algorithm may be important even in case the set of selected algorithms is the same.
In this thesis we only use the method given in Table 4.8 with two feature selection methods; the forward and backward versions of the feature search method presented in Subsection 3.3.1. The two versions are alternately applied, starting with the backward feature search.

4.5.3 Methods using exhaustive feature search

If the full set of features is small, as it is the case for the classifier described in Section 2.2, the objective function f may be calculated for all possible feature sets. This means that an exhaustive search of the best feature set is possible. As noted before, the algorithm of Table 4.5 may be executed with any underlying feature selection algorithms, so the exhaustive search may also be used. This method will be denoted as the ensemble method using exhaustive feature search.
The most time consuming part of calculating of the objective functions is the calculation of the recognition results on the validation set of the classifier using the considered feature set. As mentioned before, it is therefore a good idea to save those results, as the calculation of the objective function is very fast when those results are given.
The algorithm of Table 4.5 is a greedy algorithm, i.e. a once selected feature set is never removed. There are two reasons for doing so: Firstly, it is the simplest approach and secondly it is the approach for which the least classifiers have to be tested on the validation set. As the results for all feature sets are calculated in the approach discussed in this subsection anyway, it may be promising to use a more sophisticated search approach. The following approach was used in the experiments: The algorithm starts with an empty list of feature sets. After adding the best feature set, i.e. the feature set which achieved the highest value for one of the objective functions described in Subsection 4.5.1, repeated checks are carried out as to whether the replacement of any existing feature set by any other lead to a better value of the objective function. The replacement which yields the largest improvement over the current solution is adopted. If no replacement improves the solution, the next feature set is added. At the end a

Input: Objective function f and the desired number of classifiers n. $C(\mathrm{fs_i})$ denotes the classifier using feature set fs_i. Method $\mathrm{rep}(L, i, e)$ replaces the i-th element of a list L with element e.
Output: n classifiers.

```
for(i=1;i≤n;i++)
    $\mathrm{fs}_i = \arg(\max_{\mathrm{fs} \notin \{\mathrm{fs}_1,\ldots,\mathrm{fs}_n\}} f(C(\mathrm{fs}_1),\ldots,C(\mathrm{fs})))$;
    replaces =0;
    while($f(C(\mathrm{fs}_1),\ldots,C(\mathrm{fs}_i))$ not increased or replaces=0)
        replace_id = 0;
        bestf = $f(C(\mathrm{fs}_1),\ldots,C(\mathrm{fs}_i))$;
        for(j=1;j<i;j++)
            fs = $\arg(\max_{\mathrm{fs} \notin \{\mathrm{fs}_1,\ldots,\mathrm{fs}_n\}} f(\mathrm{rep}((C(\mathrm{fs}_1),\ldots,C(\mathrm{fs}_i)), j, C(\mathrm{fs}))))$;
            if($f(\mathrm{rep}((C(\mathrm{fs}_1),\ldots,C(\mathrm{fs}_i)), j, C(\mathrm{fs}))) >$ bestf)
                replace_id = $j$;
                bestfs = fs;
                bestf = $f(\mathrm{rep}((C(\mathrm{fs}_1),\ldots,C(\mathrm{fs}_i)), j, C(\mathrm{fs})))$;
        if(replace_id > 0)
            $\mathrm{fs}_{\mathrm{replace\_id}}$ = bestfs;
        replaces= replaces + 1;
return $C(\mathrm{fs}_1),\ldots,C(\mathrm{fs}_n)$;
```

Table 4.9: Ensemble method using exhaustive feature search with replacement

classifier is returned for each feature set in the list. The algorithm of this approach is shown in Table 4.9. Unlike the greedy algorithm of Table 4.5, once selected feature sets may be replaced by another. Note that the number of selected feature sets never decreases. The above described method is denoted as ensemble method using exhaustive feature search *with replacement* in the experiments.

4.5.4 Heuristic versions of the feature selection ensemble methods

In Subsection 3.3.3 a method was introduced which simulates the recognition results of classifiers using different features sets. All the methods described in the previous part of this section repeatedly test classifiers on a validation set where the classifiers are using different feature sets for the recognition. Instead of calculating the real results, only the simulated results may be used to decrease the computational complexity of the ensemble methods. The disadvantage of this approach is that only approximated results are produced which may lead to inferior ensembles. The ensemble methods using feature selection where the results of the validation set are simulated are denoted as heuristic ensemble methods using feature selection. Similarly the method described in Subsection 4.5.3 using simulated results is denoted as the heuristic ensemble method using exhaustive feature search.
It was observed that the method of Section 3.3.3 assigns too high recognition rates to small feature sets and that the inclusion of classifiers using such small feature sets has a negative impact on the performance of the ensemble. Therefore the minimal size of a selected feature set

Input: Base classifiers $C_1, \ldots, C_n$, original ensemble method EM and number of classifiers per ensemble m.
Output: Ensemble of $m \cdot n$ classifiers.

```
CS is an empty set of classifiers;
for(i:=1;i<=n;i++)
    use $EM$ with $C_i$ as base classifier to produce $m$ classifiers;
    put all $m$ classifiers produced using $C_i$ in CS;
return CS;
```

Table 4.10: Extension of ensemble methods using one base classifier to ensemble methods using several base classifiers

used for the training and the recognition of a classifier was set to 3 for the heuristic ensemble method using exhaustive feature search in the experiments. Note that the best value for this minimal size is application dependent.

4.6 Ensemble methods using several base classifiers

One common feature of all the ensemble methods discussed before is that they start from a single classifier, the base classifier, to derive an ensemble. In this section some ensemble methods are proposed which start from a set of classifier prototypes, the base classifiers, to create an ensemble. In Subsection 4.6.1 a simple approach to extend the ensemble methods using one base classifier to ensemble methods using several base classifiers will be described. In Subsections 4.6.2 and 4.6.3 a more advanced algorithm will be introduced which takes the diversity of the base classifiers into account. The basis of the algorithm will be described in Subsection 4.6.2 and the details of the algorithm which is based on the simple probabilistic boosting ensemble method is given in Subsection 4.6.3.

4.6.1 Extension of classic ensemble methods

The idea of the proposed extension of the ensemble methods is quite simple. Rather than starting with a single base classifier, as it is done in the classic ensemble methods, we initially consider the set of base classifiers and use the original ensemble method to generate an ensemble out of each individual base classifier. Then we merge all classifiers of these ensembles to get a single ensemble. This procedure is described in Table 4.10.

The produced classifiers using a base classifier are independent from the classifiers produced using the other base classifiers, because the ensemble method is executed separately for each base classifiers. So the diversity of the classifiers is not explicitly exploited (there is an implicit exploitation by merging the classifiers to a single ensemble). The method described in the two next subsections incorporates a mechanism which try to exploit the diversity of the base classifiers.

```
Input: Base classifiers $C_1, \dots, C_n$, training set $T$ of size $s$, and number of classifiers per ensemble $m$.
Output: $m \cdot n$ classifiers.

$p_i$ is the selection probability distribution for base classifier $i$;
$p_i(x)$ is set to $\frac{1}{s}$ for all $x \in T$ and $i \in \{1, \dots, n\}$;
for(i=1;i $\leq n$;i++)
    train $C_i$ on $T \rightarrow$ classifier $C_i^1$;
for(i=2;i $\leq m$;i++)
    for(j=1;j $\leq n$;j++)
        test classifier $C_j^{i-1}$ on $T$;
    modify $p_i(x)$ using information derived from all tested classifiers; (*)
    for(j=1;j $\leq n$;j++)
        sample $s$ elements of $T$ using the selection prob. $p_j(x) \rightarrow T_j^i$;
    train $C_j$ on $T_j^i \rightarrow$ classifier $C_j^i$;
return classifiers $C_1^1, \dots, C_n^1, C_1^2, \dots, C_n^2, \dots, C_1^m, \dots, C_n^m$;
```

Table 4.11: Basic algorithm of the advanced boosting ensemble method using several base classifiers

4.6.2 Advanced boosting ensemble methods using several base classifiers

The basic algorithm of the advanced boosting ensemble methods using several base classifiers introduced in this thesis is given in Table 4.11. First each base classifier is trained on the whole training set, producing a classifier for each base classifier. Then repeatedly one classifier for each base classifier is produced using a separate selection probability distribution for each base classifier. The most important line of the algorithm is indicated by the star sign. Here the selection probabilities are modified using the information of *all* previously created classifier, i.e. the classifiers produced using a base classifier are also considered when modifying the selection probability distributions of other base classifiers and so they may have an indirect impact on classifiers produced by those other base classifiers.

In Figure 4.3 the difference of a boosting algorithm extended with the method introduced in Subsection 4.6.1 and the boosting algorithm introduced in this subsection is illustrated in the case of two base classifiers. In this illustration the feedback loop of the production of the classifiers is shown. Note that this loop is several times repeated, i.e. several classifiers are produced for each base classifiers, and that the test results denote the results of all previously created classifiers. The only difference of the methods is that in the first case the modification of the selection probabilities of a base classifier is done using only the results of classifiers produced by this base classifier.

Different boosting methods were used for the design of advanced boosting algorithms. So methods based on AdaBoost (compare Subsection 4.2.2), simple probabilistic boosting (compare Subsection 4.4.2), effort based boosting (compare Subsection 4.4.3) and simple probabilistic boosting with effort (compare Subsection 4.4.4) were designed, were only the one based on simple probabilistic boosting produced promising results. This promising method will be introduced

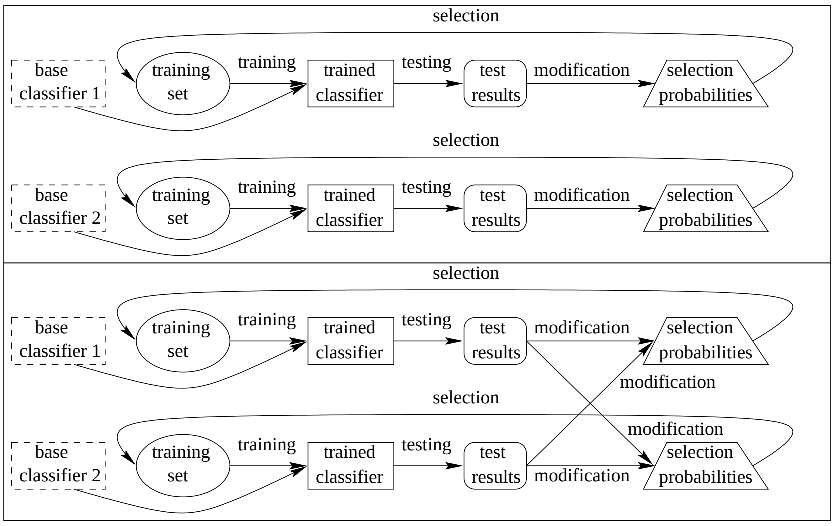


Figure 4.3: The difference of the extended boosting algorithms using several base classifiers (top) and the advanced boosting algorithms using several base classifiers (bottom) illustrated for the case of two base classifiers

in the next subsection.

4.6.3 Multi simple probabilistic boosting (MSPB)

The basic algorithm of the multi simple probabilistic boosting ensemble method is given in Table 4.11. The modification of the selection probability distributions is based on the same idea as in simple probabilistic boosting (compare Subsection 4.4.2), i.e. the selection probability of an element x is set proportional to the probability of a misclassification of this element by the ensemble. The detailed algorithm for setting the selection probabilities is given in Table 4.12. First the probability $p_{global}(x)$ of a misclassification of the training element considering the whole ensemble of already produced classifies is calculated. This value indicates how large the average selection probabilities of the base classifiers should be and the selection probability of element x is set proportional to $p_{global}(x)$ for all base classifiers. The different base classifiers may not be equally suited for the recognition of an element. The ability of a base classifier C_i to recognize a training element x is estimated by calculating the probability $p_{local(i)}(x)$ of a misclassification of that element by the ensemble consisting only of the classifiers produced by this base classifier. The previously mentioned ability is then assumed to be proportional to the value of $1 - p_{local(i)}(x)$. In this algorithm the base classifiers which are most suited for the recognition of an element x receive the highest selection probabilities for the element, because we want each base classifier to "focus" on the elements which they are most suited to recognize[5]. So the selection probability of the i-th base classifier of the element x is set proportional to $1 - p_{local(i)}(x)$. To ensure that the sum of the selection probabilities of all base classifiers is 1 a

[5]The reason for doing this is similar to the reason stated for the effort based boosting (compare Subsection 4.4.3). We want to distribute the resources in a way so that the ensemble performance is optimized.

Input: Training set T and results of all classifiers C_i^j with $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, m\}$, with n the number of base classifiers and m the number of classifiers already produced for each base classifier. The function $p(\{C_1, \ldots, C_k\}, x)$ calculates the misclassification probability of x by the ensemble consisting of the classifiers $C_1, \ldots, C_k$.
Output: The selection probabilities $p_i(x)$ for all elements x of T and $i \in \{1, \ldots, n\}$.

```
for(all elements $x$ of the training set $T$)
    $p_{\text{global}}(x) = p(\{C_i^j | i \in \{1, \ldots, n\}, j \in \{1, \ldots, m\}\}, x)$
    for(i=1;i$\leq n$;i++)
        $p_{\text{local(i)}}(x) = p(\{C_i^j | j \in \{1, \ldots, m\}\}, x)$;
    for(i=1;i$\leq n$;i++)
        $p_i(x) = p_{\text{global}}(x) \cdot \frac{1 - p_{\text{local(i)}}(x)}{\sum_{j=1}^{n}(1 - p_{\text{local(j)}}(x))}$;
for(i=1;i$\leq n$;i++)
    normalization of the selection probabilities $p_i(x)$ so that $\sum_{i \in T}(p_i(x)) = 1$;
```

Table 4.12: Calculation of the selection probabilities $p_i(x)$ for MSPB

normalization is done at the end of the modification of the selection probabilities.
To calculate the misclassification probability of an ensemble the same four functions as for SPB may be used (compare Subsection 4.4.2).

Chapter 5

Combination methods

As noted in the introduction, only parallel combination schemes are examined in this thesis. In Section 5.1 an explanation is provided as to why some well-known combination schemes do not work for the classifiers introduced in Chapter 2, when applied to the word recognition task. The following section briefly describes some classic combination schemes which work for the considered application, and which were applied in the experiments. In Section 5.3 some adaptions to the score combination schemes are introduced which render these schemes applicable for the classifiers used in the experiments. A combination scheme for special HMM classifiers is presented in Section 5.4. Finally, a combination scheme is introduced in the last section which first reduces the set of classifiers and then applies another scheme to combine only the results of the classifiers which are in this set.

5.1 Inapplicable combination schemes

This section explains why some well-know combinations schemes are inapplicable for the classifiers used in the experiments.

5.1.1 Bayesian combination rule and behavior knowledge space

The Bayesian combination rule [49] calculates the probability of a pattern to belonging to class i when the classifier outputs class j for all pair of classes and all classifiers is calculated. We normally deal with a large number of classes in word recognition. So it is not possible to estimate the above mentioned probability accurately. Consider the following example: The training set consists of 20,000 training patterns and there are 2,000 classes. There are $\frac{2000 \cdot 1999}{2} = 1,999,000$ possible pairs of classes, which is far more than the number of training patterns.
In a simpler version of the Bayesian combination rule only the classification performance of the classifiers for each class is considered. In word recognition there are often only very few instances of the same word class in the training set, so even for this simpler version of the Bayesian combination rule the probability would be estimated inaccurately.
As the estimated probabilities are unreliable, both versions of the Bayesian combination rule are not promising and were not applied in the experiments.

In the Behavior knowledge space method [31], which may be regarded as a refinement of the Bayesian combination rule, even more probabilities must be estimated (the number of probabilities is M^{K+1} where M is the number of classes and K the number of classifiers). For the same reason as for the Bayesian combination rule the Behavior knowledge space is inapplicable for word recognition.

5.1.2 Score combination schemes and trained combiner

In the score combination schemes the score for all classes and for all classifiers are considered and the maximum, minimum, average or the median of the scores for each class is calculated. The calculation of the scores of the classes for the classifiers described in Chapter 2 is not easily possible because of the following reasons:

- The Hidden Markov Model Toolkit (HTK) [102] used by the classifiers limits the size of the output list to a maximum of 128, i.e. only the 128 best classes may be output. Note that in word recognition tasks often a very large number of classes, e.g. more than 2000, are considered.
- The memory and time complexity of the recognition increases with the size of the output list.
- The output scores are unnormalized and it is expected that the scores of classes which were not in the first ranks do not hold a lot of information.

Because of the above mentioned reasons, the classic score combination schemes were not applied in the experiments.
In the literature, often the combination of the output scores of every class and every classifier of the ensemble by a new classifier is presented, e.g. see [99]. The new classifier has to be trained to optimize the combination. This approach will not work for the classifiers described in Chapter 2 because it is difficult to calculate the scores, as mentioned above, and because the number of input features of this new classifier would be too large (e.g. if there are 2000 classes and 10 classifiers, the feature vector input to the new classifier has a dimensionality of 20000).

5.2 Classic combination schemes

In this section some classic combination schemes which were applied in the experiments are described. The following notations will be used. The i-th best class output by the classifier C for the pattern x is denoted by $class(C,i)(x)$. Similarly $score(C,i)(x)$ denotes the score of the i-th best class output by the classifier C for the pattern x. Because a higher score value signifies a better result, $score(C,j)(x) > score(C,i)(x)$ holds for all $j < i$. $C_1, \ldots, C_n$ are the classifiers of the ensemble which should be combined and $c_1, \ldots, c_m$ are the classes of the classification problem. The score value of the class c_j output by the classifier C_i for the pattern x is denoted as $score(C_i, c_j)(x)$ and the rank of the class c_j in the k-best list of classifier C_i for x is given by $rank(C_i, c_j)(x)$. If the class c_j is not in the k-best list output by the classifier then $rank(C_i, c_j)$ is defined as $k+1$.

5.2.1 Voting

Voting is the simplest combination method and was used as early as 1974 [87]. Voting considers only the best class output by each classifier and regards the class which appears most often in the output of the classifiers as output of the combined classifier. So the following equation holds:

$$\text{vote}(C_1, \ldots, C_n)(x) = \arg(\max_{c_i \in \{c_1, \ldots, c_m\}} |\{C_j | \text{class}(C_j, 1)(x) = c_i\}|) \tag{5.1}$$

If there are several classes which appear a maximum number of times in the output of the classifiers then a tie occurs. These classes are then denoted as tied classes. To resolve such ties the following approaches may be used:

- **Ties handling by rejection**: All patterns for which a tie occurs are rejected by the combined classifier.
- **Random ties handling**: Randomly one of the tied classes is selected as output of the combined classifier.
- **Ties handling by another combination scheme**: Another combination scheme is applied to the results of the tied classes. Note that often the combination is done only with a part of the whole ensemble, because there may be many classifiers which do not output one of the tied classes.

5.2.2 Weighted voting

The voting combination scheme introduced in the last subsection treats all classifiers equally. This may not be appropriate if the individual classifiers have very different performances. Weighted voting is a more sophisticated approach which assigns a weight w_i to each classifier C_i and regard the class which has the highest sum of weights of classifiers outputting this class as the result of the combined classifier. The following equation defines the weighted voting combination:

$$\text{wvote}(C_1, \ldots, C_n)(x) = \arg(\max_{c_i \in \{c_1, \ldots, c_m\}} (\sum_{\{j | \text{class}(C_j, 1)(x) = c_i\}} w_j)) \tag{5.2}$$

In this thesis two approaches to set the weights of the classifiers were examined, which take the results of the classifiers on the training set into account:

- **Performance weights**: The weight w_i of the classifier C_i is set equal to the recognition rate of C_i on the training set. In this combination, better performing classifier get a higher weight. If the classifiers have similar performances it is rather unlikely that this weighted voting approach will produce different results other than voting in cases where no tie occurs. Yet if there is a tie, this combination scheme is likely to produce a unique result. So this scheme may be regarded as a slight modification of voting.
- **Optimized weights**: The weights $w_1, \ldots, w_n$ of the classifiers $C_1, \ldots, C_n$ are set to values for which an optimal ensemble performance is achieved on the training set. To optimize

the weights a genetic algorithm [23, 89] is used in this thesis. The same approach to set the weights of the classifiers was also used in [56, 58]. In the genetic algorithm the genomes are arrays of real numbers between 0 and 1 where the value at position i of the array corresponds to the weight of the i-th classifier. For details of the used genetic algorithm see [25].

As an alternative to measuring the performance of the classifiers or the ensemble on the training set, these values could also be measured on a validation set to avoid overfitting effects. The disadvantage of using a separate validation set is that more data is needed for the design of the system.

5.2.3 Borda count and other ranked list based combinations

Voting and weighted voting only take the best class of each classifier into account. The classifiers described in Chapter 2 are able to output a k-best list of classes, instead of only the best class. There are some combination schemes working with such k-best lists in the literature of which Borda count [29] is the most popular. It is defined by the following equation:

$$\mathrm{bcount}(C_1, \ldots, C_n)(x) = \arg(\max_{c_i \in \{c_1,\ldots,c_m\}} \sum_{j=1}^{n} (k - \mathrm{rank}(C_j, c_i)(x) + 1) \tag{5.3}$$

To each result in the k-best lists the number (k-rank of the result+1) is assigned and for each class the number is summed over all classifiers. The class with the highest sum is the output of the combined classifier. Borda count only works if a rank is assigned to each class for all classifiers. Like voting, Borda count requires no training and is very simple to compute. There are several disadvantages of Borda count: It does not take the different abilities of the classifiers into account and it lacks a theoretical underpinning. As in voting, in Borda count ties may occur. They are normally quite rare, but as in voting a tie breaking mechanism must be defined.
A more complex combination than Borda count based on ranked lists is given by the following equation:

$$\mathrm{rcombi}(C_1, \ldots, C_n)(x) = \arg(\max_{c_i \in \{c_1,\ldots,c_m\}} \sum_{j=1}^{n} (a_j + w_j(\mathrm{rank}(C_j, c_i)(x)))) \tag{5.4}$$

For each classifier C_j there is a weight function $w_j(\mathrm{rank})$ and in addition a weight a_j is assigned to each classifier C_j. Borda count is a special version of this combination with $a_j = 0$ and $w_j(\mathrm{rank}) = k - \mathrm{rank} + 1$. The parameters of the combination may be set by hand, as is done in Borda count, or may be set to optimize the performance of the combined classifier on the training or a validation set. In this thesis only Borda count and another method where the weights are set by hand are tested.

5.2.4 Maximum score scheme

As mentioned in Subsection 5.1.2 the score combination schemes are inapplicable for the classifiers described in Chapter 2, because the score values for all classes and classifiers must be calculated. There is however one exception: The maximum score combination scheme. This scheme is defined in the following equation:

$$\text{maxscore}(C_1, \dots, C_n)(x) = \arg(\max_{c_i \in \{c_1,\dots,c_m\}} \max_{j=1}^{n}(\text{score}(C_j, c_i)(x))) \tag{5.5}$$

The output of the combined classifier is the class with the highest score of all classes and all classifiers. Because $\text{score}(C, j)(x) > \text{score}(C, i)(x)$ holds for all $j < i$, equation 5.5 can be transformed in:

$$\text{maxscore}(C_1, \dots, C_n)(x) = \text{class}(\arg(\max_{C_i \in \{C_1,\dots,C_n\}}(\text{score}(C_i, 1)(x))), 1) \tag{5.6}$$

So only the score of the best class for each classifier must be known and therefore the maximum score combination scheme is applicable for the considered classifiers.

5.3 Adaptions of the score combination schemes

In this section some adaptions of the classic score combinations schemes are shown. In Subsection 5.3.1, schemes in which the score values of only a few classes are calculated, are presented. The following section contains the description of schemes which only work with the score values of the best results of the classifiers. In Subsection 5.3.3 a way to normalize the score of the classifiers is introduced. In the last subsection score schemes based on a confidence measure are presented.

5.3.1 Class reduced score combination schemes

The main idea of the class reduced score combination scheme is to calculate the score values for only a few candidate classes. This reduced set of candidate classes should contain the classes which are most likely to be the correct. A straightforward approach is to use the classes for which the highest score for one of the classifiers was achieved, because it is reasonable to assume that at least one classifier output the correct class. (In the case where no classifier outputs the correct class as best result, a correct classification of the pattern using any combination scheme is rather unlikely). The algorithm of the new combination schemes is shown in Table 5.1. Here the average score scheme is used. By replacing the sum operator with a minimum, maximum or median operator the other schemes may also be implemented. Firstly, the best classes of all classifiers for pattern x are calculated. Those best classes comprise the set of candidate classes. The vocabulary which contains all the words that can be output by the classifier is set equal to this set. In the next step the classifiers are used again for the recognition of x, but this time we calculate a k-best list where k is the size of the set of candidate classes. It is therefore ensured that each classifier outputs results for all candidate classes. (The number of classes n and the size of the ensemble k may not be the same as some outputs of the classifiers may be the same). Note that any pruning mechanism of the classifiers must be deactivated so that all these results are really produced. Finally the score scheme is applied to the results of the candidate classes. The whole procedure is illustrated in Figure 5.1.

Input: A pattern x and classifiers $C_1, \ldots, C_n$.
Output: Output of combined classier.

```
for(i:= 1;i≤n;i++)
    recognition of $x$ by $C_i$ producing a 1-best list;
set $L = \{class(C_1, 1)(x), \ldots, class(C_n, 1)(x)\}$;
let $L = \{l_1, \ldots, l_k\}$;
set the vocabulary of recognizable patterns to $L$;
for(i:= 1;i≤n;i++)
    recognition of $x$ by $C_i$ producing a $k$-best list;
return arg($\max_{l_i \in \{l_1,\ldots,l_k}(\frac{1}{|n|} \cdot \sum_{j=1}^{n}(\text{score}(C_j, l_i))))$;
```

Table 5.1: Class reduced average score combination scheme

5.3.2 Best class score combination schemes

The method of the last subsection has a rather high time complexity as the classifiers are run twice for the recognition of a pattern. In the first run the best result is output, and in the second run a list of the k-best output list is produced. The best class score scheme introduced in this subsection only uses the score values of the best classes and is therefore quite fast. The equation of the best class average score scheme is the following:

$$\text{av\_score}(C_1, \ldots, C_n)(x) = \arg(\max_{c_i \in \{c_1,\ldots,c_m\}}(\frac{\sum_{\{C_j|\text{class}(C_j,1)(x)=c_i\}} \text{score}(C_j, 1)(x)}{|\{C_j|\text{class}(C_j, 1)(x) = c_i\}|})) \tag{5.7}$$

The sum operator is applied on all best results of the classifiers which belong to the same class. If there are no such results, i.e. if a class is not among the best result of the classifiers, the class is not considered for the maximum operator in equation 5.7. By replacing the sum operator with a minimum, maximum or median operator the other score schemes may also be implemented. As in the class reduced score scheme, only the best classes of the classifiers may be output by the combined classifier. As mentioned before, the advantage of this combination scheme is that the classifiers must only provide the best result. The fact that the compared score values were output by different classifiers, and therefore may not have the same range of values, is a rather strong disadvantage of this scheme.
Note that the best class maximum score scheme, the reduced class maximum score scheme and the classic maximum score scheme are equivalent. They all output the class with the highest score of all classifiers.

5.3.3 Normalization

The score value output by the classifiers described in Chapter 2 is a likelihood value where a higher likelihood corresponds to a higher probability that a pattern is classified correctly. It is not guaranteed that classifiers with different HMM topologies, different feature sets or even different training sets have the same range of likelihood values. For example a likelihood value

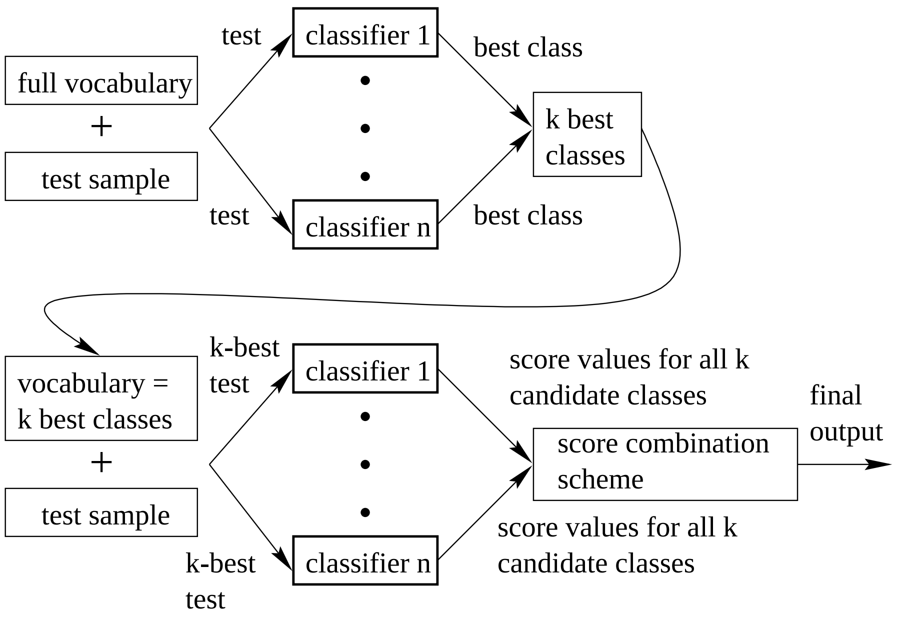


Figure 5.1: Class reduced score combination scheme

which indicate a classification with low confidence for one classifier may indicate a very confident classification for another classifier. The different meaning of the same likelihood value for the classifiers may bias the results of the adapted score combination rules introduced in this section. This is especially the case for the best class score combination schemes. A way to reduce this bias is to normalize the score values of the different classifiers. The normalization equation is the following:

$$\text{score}_{\text{new}} = \frac{\text{score}}{\overline{\mid (score - \overline{score}) \mid}} - \overline{score} \tag{5.8}$$

The averages in equation 5.8 are taken over the whole training set. The score values are linearly transformed so that the average score is 0 and the standard deviation of the scores is 1 on the training set. Note that there is a different normalization equation for each classifier. After the normalization the score values of different classifiers are comparable (e.g. a score of 0 corresponds to the average score).
Instead of using the training set to calculate the normalization parameters, a validation set may also be used. This reduces overfitting effects, however more data is needed in the design phase of the classifiers.
The normalization procedure introduced in this subsection may be applied to any score combination scheme, including the schemes presented in this section.

5.3.4 Confidence combination schemes

As mentioned before, the score value output by the classifiers used in the experiments is a likelihood value which is not a reliable measure of the quality of the recognition. In Section 3.5 a confidence measure for classifiers was introduced which may be more reliable. This measure is defined in equation 3.3. The confidence is the normalized difference of the score of the best and the second best class. (There is already a normalization term included in equation 3.3.) For each classifier we can calculate the confidence of the recognition of the best class using this measure. The average confidence combination scheme is defined by the following equation:

$$\mathrm{conf\_av}(C_1,\ldots,C_n)(x) = \arg(\max_{c_i\in\{c_1,\ldots,c_m\}}(\frac{\sum_{\{C_j|\mathrm{class}(C_j,1)(x)=c_i\}}\mathrm{conf}(C_j)(x)}{|\{C_j|\mathrm{class}(C_j,1)(x)=c_i\}|})) \quad (5.9)$$

Here the confidence of the classifier C_j, when classifying element x, is denoted as $\mathrm{conf}(C_j)(x)$. By replacing the sum operator with a minimum, maximum or median operator the minimum, maximum and median confidence combination schemes may also be implemented. The equation 5.9 is almost the same as the equation of the best class average score scheme (compare equation 5.7). The maximum confidence combination is the same as the dynamic classifier selection when the confidence value of the classifiers is used as the selection criteria (for more information about the topic of dynamic classifier selection see for example [22]).

5.4 Special HMM classifier combination scheme

In this section a new classifier combination method for special HMM classifiers is proposed. The classifiers to be combined must satisfy two constraints. First, the HMMs for the classes must be composed of smaller HMMs corresponding to basic elements of the classes, e.g. in the word recognition character HMMs are often used and the word models are built by concatenating the appropriate character models. The HMMs corresponding to the basic elements will be denoted as basic HMMs in the following. The second constraint is that the feature vectors input to the HMMs must be the same for all classifiers. The classifiers described in Chapter 2 satisfy the first constraint and classifiers produced by ensemble methods which just modify the training sets, like Bagging or AdaBoost, satisfy the second constraint. There are other domains than handwritten text recognition where the proposed method may be applied. One example is speech recognition, where the basic HMMs often are on the phoneme level and the word HMMs are composed of these phoneme HMMs.

The main idea of the combination is to combine the basic HMMs of the different classifiers instead of the class HMMs. In the case of word recognition this means that the character HMMs of the different classifiers are combined. The class models of the combined classifiers are then built from the combined basic HMMs. The algorithm of the combination is shown in Table 5.2. In Table 5.3 the method to combine different HMMs corresponding to the same basic element is shown. Here a “dummy” state denotes a state which does not emit any observation symbols (i.e. it doesn’t consume any input feature vector). The combined HMM contains the states of the HMMs of all classifiers and there is a transition from the start state of the new HMM to the start state of each HMM of the classifiers (with equal probability for all those transitions). The end states of the combined HMM include all end states of the individual HMMs. Here it is assumed that each HMM has only one start state, but only a small modification of the method is needed for the combination of HMMs with many start states. Note that none of the output distributions of the states, and none of the transition probabilities between states originating from the same HMM are changed.

Input: Classifiers $C_1, \ldots, C_n$, the set of the basic elements $e_1, \ldots, e_k$, and the set of the classes $c_1, \ldots, c_m$ where the class c_i is build from the basic elements in sequence E_i. HMM(C_i, e_j) denotes the HMM of the classifier C_i corresponding to the basic element e_j.
Output: Combined classifier C.

```
create new classifier $C$;
for(i:=1;i≤k;i++)
    HS is a set of HMMs;
    for(j:=1;j≤n;j++)
        put HMM($C_j, e_i$) in $HS$;
    $M_k$ is the combination of all HMMs in $HS$;
for(i:=1;i≤m;i++)
    let $E_i = \{e_{i1}, \ldots, e_{io}\}$;
    CM$_\mathrm{i}$ is build from the models $M_{i1}, \ldots, M_{io}$;
    the model of class $c_i$ for classifier $C$ is set to CM$_\mathrm{i}$;
return $C$;
```

Table 5.2: Algorithm for the special combination of HMM classifiers

In Figure 5.2 the combination of two HMM based word recognizers is shown. The character models for “t”, “h” and “e” of the combined classifier are constructed from the corresponding character models of the two classifiers, and the word model for “the” is the concatenation of the new character models.
The motivation of the proposed combination approach is the following: Consider a test pattern which consists of several basic elements. The model best suited to the recognition of the basic elements may not all belong to the same classifier. For example in the situation illustrated in Fig. 5.3, where two classifiers are each trained on one pattern instance of the word “is”, it is obvious that for character “i” of test pattern 1, the HMM of the first classifier is the most suitable, whereas character “s” is better modeled by the HMM of the second classifier. In the combination scheme introduced in this section the optimal path found by the Viterbi recognition algorithm (compare Subsection 2.1.3) may include the character models of both classifiers, e.g. in the example of Figure 5.2 all the paths shown in Figure 5.4 are possible. So if the two classifiers of Fig. 5.2 are combined by the proposed scheme, the optimal path for the test pattern 1 includes the HMM of the character “i” of the first classifier and the HMM of the character “s” of the second classifier, and a high score for the word “is” may be obtained, as it is accurately modeled. If the test pattern is best modeled by basic HMMs that all belong to the same classifier, as it is the case for the test pattern 2 in the example of Figure 5.3, the new combination scheme does not result in any improvement, compared to the maximum score combination scheme (compare Subsection 5.2.4), where the best suited *class* model, i.e. the model which output the highest likelihood, is selected.
The new method proposed in this section has some similarity with the concept of allographs. Allographs of a character are shapes which correspond to different writing styles of the character. When using allographs for a recognition system the shape of a character in a word may be like any allograph of that character. Allographs are often used in on-line handwritten character

Input: The HMMs $M_1, \dots, M_n$ corresponding to the same basic element.
Output: New HMM M for the basic element.

```
for(i:=1;i≤n;i++)
    Include all states of $M_i$ (with all transitions) in $M$;
create dummy start state $s_s$ for $M$;
for(i:=1;i≤n;i++)
    let $s_i$ be the start state of $M_i$;
    set the transition probability from $s_s$ to $s_i$ to $\frac{1}{n}$;
for(i:=1;i≤n;i++)
    all end states of $M_i$ are also marked as end states of $M$;
return $M$;
```

Table 5.3: Algorithm for the combination of the HMMs corresponding to the same basic element

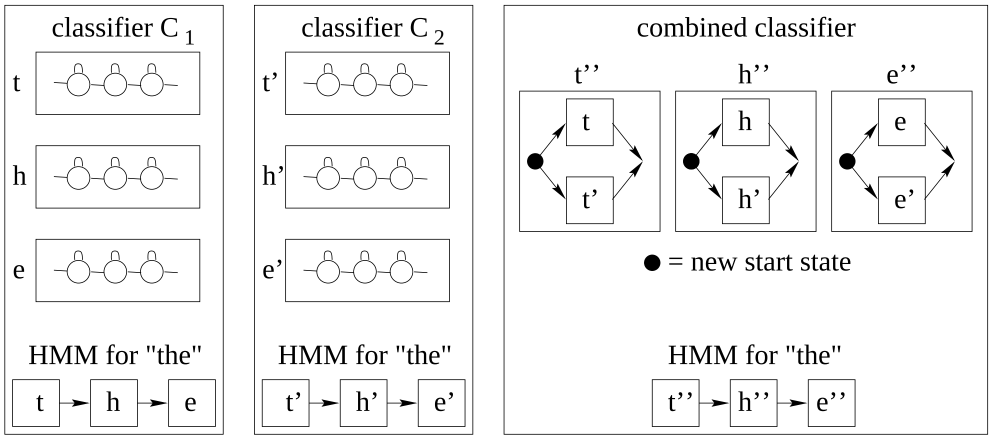


Figure 5.2: Combination of two HMM based word recognizers for the model of the word "the"

recognition [92, 101]. There are however some important difference between using allographs and the proposed method. Firstly, the different allographs of a character must usually be determined. To do this, the data must be manually labeled or some clustering algorithm must be applied (for word recognition a segmentation of the words into characters is also needed). Secondly, the proposed method may be applied to arbitrary HMMs, e.g. HMMs trained on different random subsets of the whole training set. In addition, the proposed method is not limited to the domain of word recognition.

5.5 Confidence based classifiers reduction scheme

The results of all classifiers of an ensemble are usually considered for the combination to produce the output of the ensemble. In this section a combination scheme is introduced which combines only the results of classifiers which recognized the pattern with a high confidence. The idea of the confidence based classifiers reduction combination scheme is that only the classifiers for

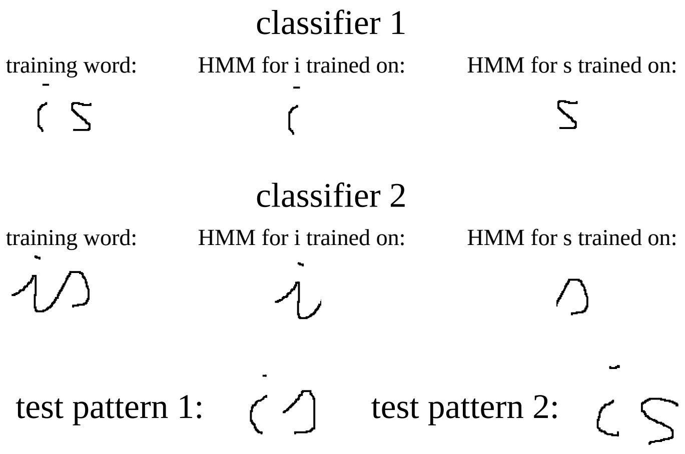


Figure 5.3: Two example of test patterns. For the first pattern the character models best suited for its recognition don't belong to the same classifier, which is not the case for the second test pattern

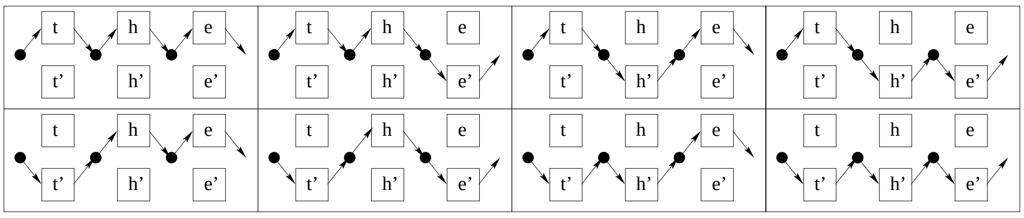


Figure 5.4: The 8 possible combinations of inclusions of different character models in the optimal path for the word "the" when using the combined classifier of Figure 5.2

which it is likely that the pattern is classified correctly, should have an impact on the ensemble result. The likelihood of a correct classification is estimated by a confidence measure. The combination scheme proposed in this section uses an underlying combination scheme which is applied to combine the reduced set of results. The detailed algorithm of the confidence based classifiers reduction combination scheme is given in Table 5.4. Only classifiers whose confidences are not lower than a threshold t are used for the combination. If there is no such classifier then the underlying combination scheme is applied to the ensemble containing only the classifier with the highest confidence (which normally leads to the output of the result of this classifier).
In this thesis the confidence measure introduced in Section 3.5 and defined in equation 3.3 was used for the combination scheme described in this section. This confidence measure already includes a normalization step which maps the average confidence value of all training patterns and classifiers to the value 1. Note that the confidence based classifiers reduction scheme may be used in conjunction with any other combination scheme, e.g. with all schemes introduced in this chapter.

Input: Given is an underlying combination scheme S, a rejection threshold t, a set of classifiers $C_1, \ldots, C_n$ and a test pattern x. cf(C, x) is the confidence of classifier C for pattern x.
Output: Combined result for pattern x.

```
CS is an empty set of classifiers;
for($i$=1;$i \leq$n;$i$++)
    if(cf$(C_i, x) \geq t$)
        put $C_i$ in CS;
if(CS is an empty set)
    $C_{max} = \arg(\max_{C_i \in \{C_1,\ldots,C_n\}}(\mathrm{cf}(\mathrm{C_i}, \mathrm{x})))$;
    put $C_{max}$ in CS;
return the combination of the results of all classifiers in CS for element $x$ using $S$;
```

Table 5.4: Algorithm of the confidence based classifiers reduction combination scheme

Chapter 6

The selection of good parameters and methods

In this section initial experiments are presented. The goal of these experiments was to determine the best methods in optimizing the parameters of the classifiers described in Chapter 2, the best ensemble methods, the best values of the parameters of the ensemble methods and the best combination schemes. As the two classifiers of Chapter 2 are very similar, only the classifier in Section 2.2 was used in the experiments described in this chapter.

In Section 6.1 the different experimental setups used in this thesis are described. The next section introduces abbreviations for the most frequently applied combination schemes. Section 6.3 describes some statistical tests used in the analysis of the experiments. In the following sections the properties and the optimization of the single classifier are addressed. In Section 6.4 the stability of the used classifier is measured. The following section evaluates the rejection criterion. Experiments with feature selection algorithms and PCA are then discussed in Sections 6.6 and 6.7. In Section 6.8 experiments with the optimization of the number of states are described. The optimal order in which to apply two kinds of optimization methods is then determined in Section 6.9 and experiments with the strategies to find the optimal training method are presented in Section 6.10. Time and memory complexities of the classifier are examined in the following section. The question of whether an N-list should be output by the classifier or just the best class is addressed in Section 6.12.

The following sections evaluate different combination schemes. In Section 6.13 the score schemes which were adapted for the application with the considered classifier are evaluated. The score schemes based on the confidence measure are then examined in the next section. The normalization of the adapted score schemes is examined in Section 6.15. Experiments with the combination scheme that first reduces the set of classifiers before the combination are described in Section 6.16. In Section 6.17 the special HMM classifier combination scheme is compared to other schemes.

The rest of the chapter addresses ensemble methods. In Sections 6.18 and 6.19 experiments with the classic ensemble methods are shown. The influence of the vocabulary and training set sizes on the performance of the classic ensemble methods is studied in the following section. Experiments with the partitioning and the new boosting ensemble methods are shown in

Section 6.21 and Section 6.22, respectively. The ensemble methods using feature selection are tested in Section 6.23. The extension of the classic ensemble methods for the use of several base classifiers is evaluated in the next section. In Section 6.25 experiments with the multi simple probabilistic boosting ensemble method are shown. In the final section, methods to find the optimal classifier number of the ensemble methods using feature selection are evaluated.

6.1 Experimental setups

In this section all experimental setups used are described. As mentioned in Section 2.4 the data for the experiments was extracted from the IAM database. The following terms are used to describe an experimental setup:

- **Training set**: The free parameters of the classifiers are set using the data of the training set. For the classifiers presented in Chapter 2 the Baum-Welch algorithm was used with all patterns of the training set.
- **Validation set**: Values of some meta-parameters of the classifiers are determined by measuring the performance of the classifier on the validation set. For the classifiers introduced in Chapter 2, such parameters are, for example, the number of training iterations of the Baum-Welch algorithm (compare Section 3.2), the number of states for each HMM (compare Section 3.1) or the features used by the individual classifiers (compare Section 4.5). Some ensemble methods also use a validation set, e.g. the ensemble methods introduced in Section 4.5.
- **Test set**: The performance of a classifier or an ensemble of classifiers is measured with the test set. To avoid overfitting effects, the test set must not be used to determine optimal values for any parameters of the classifiers, ensemble methods or combination schemes. This means that the recognition of the test patterns is always the last step in any experiment.
- **Cross-validation**: In the first step of cross-validation, the original training set is split into n equally sized partitions. Then $n-1$ partitions are used as the new training set, and the remaining partition as the validation set. So each partition may be used once as a validation set, and n validation processes with different training and validation sets may be executed. A cross-validation with $n=4$ is depicted in Figure 3.3. The reason for using cross-validation is that when using fixed training and validation sets, the estimations obtained for the optimal values of the meta-parameters may be unstable. By repeating the validation process using cross-validation and averaging over all n runs, more stable estimations are obtained.
- **Partial cross-validation**:Similar to cross-validation the original training set is split into n partitions. Instead of using each of the n partitions once as validation set, only k partitions are used. This means that the results are only averaged over k validations processes. The reason for doing so is that the time complexity of a full cross-validation is often too high, but more stable estimations are needed, than obtained when using a fixed training and validation set.

- **Vocabulary**: The classifiers described in Chapter 2 use a list of recognizable words, called the vocabulary. In most experiments in this thesis the vocabulary contains all word classes which occur in the union of the training, validation and test set. Note that by defining the vocabulary in this way the correct classes of the test patterns are always included in the vocabulary. The problem of the correct class of the test words not being in the vocabulary was addressed in [93, 96].
- **Writer independent setup**: In a writer independent setup, the writers of the words of the test set did not contribute words to the training and validation set. On the contrary, in a writer dependent setup there are some writers who contributed both to the test set, and either to the training or validation set.

Six data sets with different setups were used in the experiments:

1. **Preliminary set**: This data set was used for initial small experiments. Unlike the other sets, the words of the text lines are segmented by the algorithm introduced in [65]. This segmentation algorithm filters out most of the punctuation and has a rather high error rate of about 45 %. In addition, the definition of a word is slightly different than when using the segmentation algorithm of [104]. The data set comprises all correctly segmented words from the forms whose names start with *c03*, which is a total of 2,153 words. Those forms were written by 6 writers, and the identity of the writer of each form is known. Therefore the data set is suitable for experiments where the classifiers are trained on words of only one or few writers. The size of the vocabulary of the data set is 383 words. The following four experimental setups were used with this data set:
   (a) A fixed training set of 1,953 words and a fixed test set of 200 words.
   (b) A fixed training set of 1,753 words and a fixed test set of 400 words.
   (c) The test set contains 1,000 randomly sampled words from the whole data set and the training set consists of the remaining 1,153 words. For each experiment with this setup the sampling is repeated, i.e. the training set is never the same for different experiments.
   (d) The whole set is divided randomly into three equal sized subsets. Each subset assumes the position as test set for the other sets. This results in three independent tests which each set being used as the test set once and as a part of the training set twice. Unlike setup (1c) the division in the three subsets is done only once and the same subsets are always used.

   In all cases no validation process is done and therefore no validation set are defined. The writers of the words in the training set and the test set are always the same, so the experimental setups are all writer dependent.
2. **Small set**: This data set contains the words in the same forms as the *preliminary* set, yet the more sophisticated word segmentation of [104] is used to extract the words. A total of 3,850 words were segmented correctly. As there are more words in this data set than in the *preliminary* set, the size of the vocabulary increases to 412. Several experimental setups were used with this data set:

   (a) The test set contains 1,000 randomly sampled words and the training set consists of the remaining 2850 words. For each experiment with this setup the sampling is repeated, i.e. the training set is never the same for different experiments. As the test elements are randomly sampled, the experimental setup is writer dependent.

   (b) The training set consists of the words of 5 writers, and the test set holds the words of the remaining writer. This setup is obviously writer independent. There are 6 different such setups, as there are 6 different writers.

   (c) The training set consists of the words of 4 writers and the test set contains the words of the two remaining writers. This setup is also writer independent. There are 15 different setups.

   (d) The whole data set is separated into 3 partitions by a clustering algorithm. The training set consists of the words of two clusters, while the test set comprise the words of the last cluster. The training and test set are dependent on the clustering algorithm and there are 3 possible setups for each clustering (each cluster may form the test set). If the words of each writer are all in the same cluster, the setup is writer independent. In the experiments where this setup was used, this was not the case. Therefore the experiments were writer dependent.

   (e) The whole set is randomly divided into three equal sized sets. Each subset assumes the position as test set for the other sets. This results in three independent tests which each set being used as the test set once and as a part of the training set twice. The division into three subsets is done only once and always the same subsets are used. As the test elements are randomly sampled, the experimental setup is writer dependent.

   There is no validation set, because no validation process is done with these experimental setups.

3. **Large set**: This set may be regarded as the standard set, because it is used in most of the experiments in this chapter. It contains all words correctly segmented from the forms whose names start with *a*. The whole data set contains 10,927 words, and a total of 86 writers contributed to it. The vocabulary for this data set contains 2,296 word classes. Three experimental setups were used:

   (a) A fixed training set of 9,861 words and a fixed test set of 1,066 words. The training set contains the words of 75 writers and therefore the test set contains the words of the remaining 11 writers, thus the experimental setup is writer independent.

   (b) The same test set of 1,066 words as in the previous experimental setup is used. The original training set was split into a fixed training set of 8,795 words and a fixed validation set of 1,066 words. The setup is also writer independent.

   (c) The original set is three times randomly divided into a fixed training set of 9,861 and a test set of 1,066 words. The three divisions are carried out only once and the same three training and three test sets are always used. As the division is random, this setup is writer dependent.

The first and third setups do not include a validation set.

4. **Reduced large set**: This data set contains all words correctly segmented from the forms whose names start with *a01-0*. It is a subset of the *large* set which is described above, and contains 3495 words. There is only one experimental setup with this data set. The words are split into a training and a test set in the same way as in the experimental setup (3a), i.e. the training set of this setup is a subset of the training set of setup (3a). The 3,290 training words are written by 13 writers, and 5 writers contributed to the 205 words of the test set. There are 651 words in the vocabulary. This setup is writer independent, because the experimental setup (3a) is writer independent.

5. **Combined set**: This data set contains all words from the *small* and *large* set. Therefore the data set consists of a total of 14,777 words. The sets of writers of the *small* and *large* set are disjoint, therefore 92 writers contributed to the combined set. The vocabulary has 2,578 entries. In the only setup of this data set, the training set of setup (3a) is used as the training set and the test set of setup (3a) is used as the validation set. The whole *small* set is then used as the test set. The setup is writer independent.

6. **Final set**: The final data set is used for the evaluation of the optimized classifiers, ensemble methods and combination schemes. Experiments using this set are presented in the next chapter. This final set contains all words correctly segmented from the forms whose names start with *b*, *d* and *g*. With a total of 22,184 words written by 153 writers, the final set is by far the largest data set. The vocabulary has 3,997 entries. Therefore we are dealing with a 3,997 class problem. Three different setups were used:

   (a) All the words originating from forms whose names start with *b* and *g* are used for the training set of the setup. This results in a training set consisting of 18,920 words. The test set is composed of the remaining 3,264 words. The set of 116 writers of the training patterns and the 37 writers of the test patterns are disjoint.

   (b) The same test set as in setup (5a) is used. This setup employs a kind of partial cross-validation. The original training set is partitioned three times into a fixed training set of 17,920 words, and a fixed validation set of 1,000 words. The partitioning is done once, and in such a way that the three validation sets are disjoint. Apart from the previously mentioned constraint, the elements of the validation set are chosen randomly.

   (c) The same test set as in setup (5a) is used and a normal 10-fold cross-validation is applied, i.e. the original training set is split in 10 partitions (each of 1,892 words). 9 partitions are used for training and one partition is used for validation. Therefore there are 10 validation processes.

   All setups are writer independent. The setups (b) and (c) are used to determine the best values for the parameters of the base classifiers and ensemble methods. Therefore the test set is only used to test the classifiers, or the ensemble methods, which have optimized parameters.

nr	data set	total	train	valid	test	voca	writers	wi
1a	prelim.	2,153	fix 1,953	-	fix 200	383	6	no
1b	prelim.	2,153	fix 1,753	-	fix 400	383	6	no
1c	prelim.	2,153	random 1,153	-	random 1,000	383	6	no
1d	prelim.	2,153	3*fix 1,436	-	3*fix 718	383	6	no
2a	small	3,850	random 2,850	-	random 1,000	412	6	no
2b	small	3,850	5 writers	-	1 writer	412	6	yes
2c	small	3,850	4 writers	-	2 writers	412	6	yes
2d	small	3,850	2 clusters	-	1 cluster	412	6	(no)
2e	small	3,850	3*fix 2,566	-	3*fix 1,283	412	6	no
3a	large	10,927	fix 9,861	-	fix 1,066	2,296	86	yes
3b	large	10,927	fix 8,795	fix 1,066	fix 1,066	2,296	86	yes
3c	large	10,927	3*fix 9,861	-	3*fix 1,066	2,296	86	no
4	red. large	3,495	fix 3,290	-	fix 205	651	18	yes
5	combined	14,777	fix 9,861	fix 1,066	fix 3,850	2,578	92	yes
6a	final	22,184	fix 18,920	-	fix 3,264	3,997	153	yes
6b	final	22,184	3* fix 17,920	3*fix 1,000	(fix 3,264)	3,997	153	yes
6c	final	22,184	10* fix 17,028	10*fix 1,892	(fix 3,264)	3,997	153	yes

Table 6.1: Summary of all experimental setups

In all experimental setups where several possible combinations of training, validation and test sets exist, the results are always averaged over all possible combinations.
Table 6.1 summarizes the experimental setups used in this thesis. The column *nr* contains the number of the experimental setup, which will be used to refer to the setups in the following. The abbreviated name of the used data set is given in column *data set*. The number of words in the training, validation and test set is given in the column *train*, *valid*, and *test*, respectively, and the sum of these numbers is shown in column *total*. Under columns *voca* and *writers*, the size of the vocabulary and the number of writers in the data set are given. The last column states whether the experimental setup is writer independent.

6.2 Abbreviations for the most used combination schemes

In this section, the abbreviations for the most frequently used combination schemes are introduced. These abbreviations will be used in the descriptions of the experiments of this, and of the next Chapter.
The most used combination schemes are:

- **score**: The maximum score combination scheme as described in Subsection 5.2.4.
- **perf. weight**: The weighted voting scheme using performance weights as described in Subsection 5.2.2.

- **ga weight**: The weighted voting scheme using optimized weights as described in Subsection 5.2.2.
- The voting combination scheme as described in Subsection 5.2.1. The ties are handled in five different ways:
  - **vote reject**: Ties handled by rejection.
  - **vote random**: Random ties handling.
  - **vote best**: Ties handling by the best class score combination scheme, as described in Subsection 5.3.2. According to the applied operator, the combination scheme is called **vote best ave** (average operator), **vote best max** (maximum operator), **vote best min** (minimum operator) and **vote best med** (median operator).
  - **vote perf.**: Ties handling by *perf. weight* (see above).
  - **vote ga**: Ties handling by *ga weight* (see above).
- **special**: The special HMM classifier combination scheme introduced in Section 5.4.

6.3 Statistical tests

In this section some statistical tests are described, which are used to analyze the results. All of these tests compare the results which are produced by two different methods. In the null hypotheses it is assumed that both methods produce results of the same quality. A significance test is then applied to a test measure to calculate the probability that the null hypotheses is false. If this probability is high, the null hypothesis is rejected. Often a fixed significance level is used. Yet in this thesis always the lowest meaningful significance level is stated for which the null hypotheses is still rejected. This is done so as to provide more information about the comparison of the methods.

6.3.1 Standard comparison test

In many experiments of this thesis, results are produced by applying both a new method and a standard method. To evaluate the quality of the new method (in respect of the standard method) those results are compared. Often only the word recognition rate for both methods is known. In the following, r_1 denotes the recognition rate obtained by the new method, and r_2 the recognition rate obtained by the standard method. To compare the results the t-test is used. The null hypotheses is that the recognition rate of both methods is the same and we therefore assume the independence of the results of both methods. The test measure is given as:

$$Z = \frac{r_1 - r_2}{\sqrt{\frac{1}{n} \cdot (r_1 \cdot (1 - r_1) + r_2 \cdot (1 - r_2))}} \tag{6.1}$$

where n is the number of test patterns. If the null hypotheses holds, then the mean of Z is 0 and the standard deviation is 1. The distribution of the the test measure Z is approximately normal for a large number of test patterns n. Assuming the normal distribution of the test measure of equation 6.1, we can check the probability, or significance level, of the null hypotheses not being rejected. Because we only check the results of experiments in which the new method was better than the standard method, a rejection of the null hypotheses means that the new method is better than the standard one. For the same reason a one-sided, and not a two-sided, significance test is applied.

6.3.2 Advanced comparison test

In the standard comparison test presented in the last subsection, the independence of the results of the two methods is assumed. This is a very conservative approach, as the results of the two methods are likely to be correlated, and a difference in the recognition rates is more significant when the results are correlated. A statistical test which takes this correlation into account is the paired t-test. For this test a new random variable X is introduced. If $m_1(w)$ and $m_2(w)$ denote the results of the two methods corresponding to the same word w then the value of X for w, is defined by the following equation:

$$X(w) = \begin{cases} 1 & \text{if } m_1(w) \text{ is correct and } m_1(w) \text{ is incorrect} \\ 0 & \text{if } m_1(w) \text{ and } m_1(w) \text{ are both correct or incorrect} \\ -1 & \text{if } m_1(w) \text{ is incorrect and } m_1(w) \text{ is correct} \end{cases} \tag{6.2}$$

Using the same null hypotheses as in the last subsection, we assume that X has a normal distribution with mean 0. The test measure for this test is given as:

$$Z = \frac{\mu_x}{\frac{1}{\sqrt{n}} \cdot \sigma_x} \tag{6.3}$$

where μ_x is the mean and σ_x the variance of the random variable X estimated using the results. Again n is the number of test patterns. If the null hypotheses holds then the mean of Z is 0, the standard deviation is 1 and Z is normally distributed for large values of n. So we can once again apply again a one-sided significance test.
The advanced comparison test was applied to some of the results in Chapter 7.

6.3.3 Sign test

In many experiments the new method and the standard method were applied with a range of different configurations. To compare the two methods using the obtained results, the sign test [32] may be used. The sign test takes only pairs of corresponding recognition rates into account. How often the result of the new method is better than the result of the standard method is counted. The null hypothesis of the test is that both methods are of the same quality, i.e. that the likelihood of the first method being better than the second is 50 %. So a binomial distribution of the counter variable mentioned above is expected (with $p = 0.5$) and we can again apply a one-sided significance test.

6.4 Test of the stability of the classifier

A classifier type is called stable when the classifier trained on a set T is similar to the classifier trained on a modified version of T. This means that the behavior of a stable classifier changes

only slightly when the training set is modified. For instable classifiers only a small modification of the training set may have a large impact on the behavior of the classifier. Decision trees and neural nets are unstable classifiers and nearest neighbors and linear discriminant analysis classifiers are stable [8].
Bagging (see Subsection 4.2.1) and other ensemble methods which modify the training set are known to work best with instable classifiers, because the instability leads to diverse classifiers being created by these ensemble methods. To estimate the benefit of Bagging for the classifier described in Section 2.2, an experiment to measure the stability of this classifier was done and the results are shown below.
The setup (1b) in Table 6.1 was used for this experiment. Classifiers trained on subsets of the whole training set were tested, where two kinds of subsets were used:

1. A fixed number of elements was drawn randomly without replacement from the original training set to form the new training set. This kind of subset is denoted as the sampled subset.

2. A fixed number of randomly chosen elements were removed from the whole training set to build the new training set subset. This kind of subset is denoted as the reduced subset.

All tests were repeated ten times and the results of the single runs were compared to each other by measuring the correlation of the recognition results. Here the correlation is defined as:

$$\frac{E(X \cdot Y) - E(X) \cdot E(Y)}{\sqrt{Var(X) \cdot Var(Y)}} \tag{6.4}$$

where X and Y are the random variables for the correct classification using the first and the second classifier, respectively. $E(Z)$ is the expectation and $Var(Z)$ the variance of random variable Z. The value of the random variables X and Y is 1 when the classification is correct, and in all other cases it is 0. All values of the correlation are in the interval [-1,1], where a value close to 0 indicates the independence of the results of the two classifiers, and the value 1 signifies that the results are completely correlated[1] The correlation can be estimated by the equation:

$$\frac{c_{12} - c_1 * c_2}{\sqrt{(c_1 - c_1 * c_1)(c_2 - c_2 * c_2)}} \tag{6.5}$$

where c_1 and c_2 denote the fraction of correctly classified test patterns of the first classifier and second classifier, respectively, and c_{12} is defined as the fraction of the test patterns, for which both classifiers output the correct result.
The results of the experiments are shown in Table 6.2, where *mean* signifies the mean of the recognition results, *dev* denotes the standard deviation of the recognition results, and *num* is the number of sampled, respectively removed, elements. The average correlations of the classifiers are shown in column *corr*.
The standard deviation of the recognition rate first increases with the number of elements, which can be expected due to the increase of the recognition rate. But with larger sets the classifiers become more correlated and so the deviation decreases again.

[1]Complete correlation only occurs if the results of the two classifiers are always both correct or both incorrect.

sampled subset				reduced subset			
num	mean	dev	corr	num	mean	dev	corr
10	5.45 %	2.08 %	0.130	5	82.8 %	0.44 %	0.963
20	15.1%	2.84 %	0.257	10	82.975 %	0.57 %	0.955
40	33.23 %	4.22 %	0.302	20	82.65 %	0.21 %	0.934
80	57.88 %	3.15 %	0.479	40	83.175 %	0.53 %	0.913
160	68.48 %	1.82 %	0.647	80	83.125 %	0.68 %	0.909
320	73.58 %	1.48 %	0.679	160	82.175 %	0.72 %	0.863
640	78.98 %	1.46 %	0.706	320	81.65 %	1.28 %	0.805
1280	82.53 %	0.89%	0.810				

Table 6.2: Results of the stability test

The correlation between classifiers is large even with rather small training sets. For example when using training sets of 160 the average correlation was 0.647. Two such classifiers with the same recognition rate of 58 % are considered in the following. The probability that a pattern correctly recognized by the first classifier is misclassified by the second classifier is 0.35 times lower than the overall error rate of the second classifier, i.e. the probability that the second classifier will also classify the pattern correctly is 83 %. Note that when using training sets of 160, two classifiers have on average only 15 shared training elements.
The fact that classifiers trained on small training sets are moderately strongly correlated, and that the classifiers trained on reduced training subsets are strongly correlated, lead to the conclusion that the considered classifier is very stable.
An interesting observation is that the classifiers trained on reduced subsets with a small number of removed elements, e.g. 5 or 10, are not completely correlated. Consider for example the case of five removed training patterns. The training sets of the classifiers differed only by 10 from a total of 1753 training words. Yet on average more than 2 test patterns from 400 were misclassified by one classifier but not by the other. Therefore it seems that the classifier is not completely stable for very small modifications of the training set.
In summary it may be concluded that the classifier is in general very stable, but is slightly unstable when the modification of the training set is very small. From this it may be concluded that the potential of Bagging and similar methods to improve the performance of the classifier is not great.

6.5 Error-rejection curve

In Section 3.5 a rejection mechanism of the classifier described in Section 2.2 was introduced. To evaluate the quality of this rejection mechanism the experimental setup (3a) in Table 6.1 was used. In Figure 6.1 the error-reject curve is shown. The x-axis of the plot shows the fraction of test patterns which are rejected by the classifier, and the value in the y-axis is the corresponding error rate of the non-rejected patterns, i.e. one minus the accuracy of the classifier. The rejection threshold t was continuously increased to create the plot. The error rate of the classifier which does not reject any patterns is 33.8%.

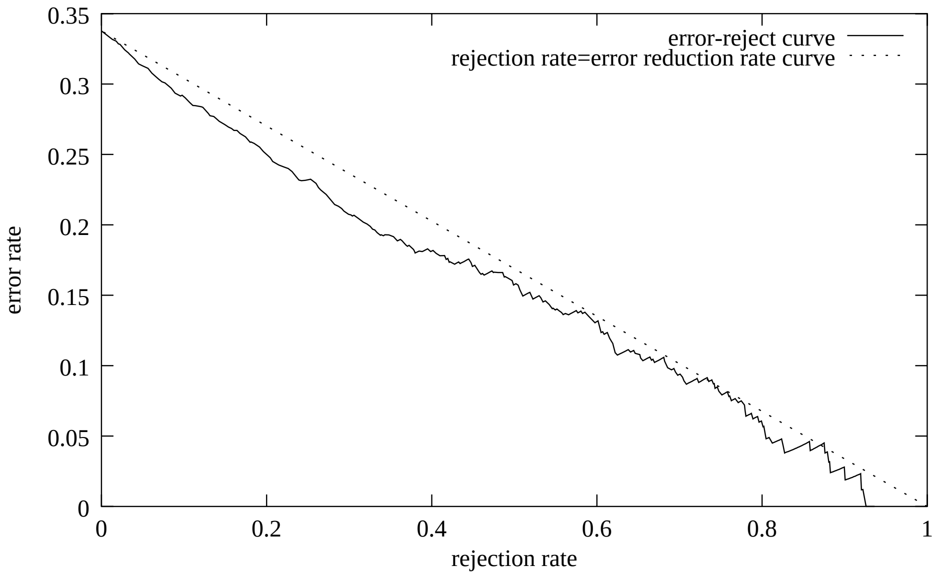


Figure 6.1: Error-rejection curve using the rejection criteria in Section 3.5

The quality of the rejection criterion is moderately good. It is certainly far better than random rejection (which would be a horizontal line with error rate=33.8 %). The error-rejection curve is for the most part below the curve for which the rejection rate is equal to the reduction rate of the error, i.e. if we reject a fraction x of the patterns, we have an error rate lower than $(1-x)\cdot 33.8\%$. Note that in some pattern recognition problems far better error-rejection curves were obtained, e.g. curves where the error rate drops much faster and has an asymptotic behavior to the x-axis.

6.6 Feature selection for one classifier

In this section the methods introduced in Section 3.3 are applied using the experimental setup (3b) in Table 6.1. The following methods were tested:

- **validation**: Feature selection using a validation set (compare Subsection 3.3.1). Three different search algorithms were tested.
- **parameters**: Feature selection using HMM parameters (compare Subsection 3.3.2). Three different measures were tested.
- **simulation**: Feature selection using simulated results (compare Subsection 3.3.3). Two methods to determine the final feature set were tested:
  - **best**: The best found feature set is used.
  - **validation**: The best found feature set of each size is validated on the validation set and the feature set which obtained the best recognition rate is used.

For each method the parameter N was varied from 1 to 5.

method	parameter	valid	test	time
validation	backward take 1- remove 1	71.48 %	68.76 %	45
validation	forward take 1- remove 1	73.08 %	67.73 %	25
validation	exhaustive	73.08 %	67.73 %	511
parameters	fisher	71.39 %	68.01 %	9
parameters	local	71.39 %	68.01 %	9
parameters	root local	71.39 %	68.01 %	9
simulation	best, $N = 1$	62.57 %	57.97 %	1.6
simulation	best, $N = 2, 4$	71.53%	67.83 %	$1.65 + 0.05 \cdot N$
simulation	best, $N = 3, 5$	70.83 %	68.11 %	$1.65 + 0.05 \cdot N$
simulation	validation, $N = 1$	72.23 %	67.54 %	10.6
simulation	validation, $N = 2, 3, 4, 5$	73.08 %	67.73 %	$10.65 + 0.05 \cdot N$
original	all 9 features	70.45 %	66.23 %	-

Table 6.3: Results of the feature selection methods

Table 6.3 shows the results of the experiment. The type of method tested is given in column *method* and the details of the tested method are given in column *parameter*. The performance of the classifier using the best found feature set on the validation and test set is given in *valid* and *test*, respectively. The approximate time complexities of the methods are shown in the last column, and were measured in multiples of the time needed for the validation of one feature set (more information on the topic of time complexities is presented in Section 6.11). The classifier using all 9 feature achieved a recognition rate of 70.45 % on the validation set and 66.23 % on the test set. For the *simulation* methods some parameter values produced the same results and are therefore written in the same row. The results of the simulation method with the parameter *best,* $N = 2, 4$ were averaged over two feature sets which both obtained the maximal simulated recognition rate.

All methods, with the exception of the *simulation* methods with $N = 1$, produced good results. The performance of the base classifier was increased by 1.5 % to a maximum of 2.53 %. The *validation* methods produced the best results, yet their time complexity is very high. Note that the exhaustive search did not produce the best results. The *parameters* methods all had the same performance of 68.01 % (they all found the same feature set). For the *simulation* methods the first version was superior to the second as better recognition rates were achieved and the time complexity was smaller. Yet the second method obtained much better results on the validation set. It seems that the correlation between the performances on the validation and the test set is not very strong. There are two reasons for this. Firstly, only a small validation set of 1066 words was used. Secondly, the writers of the validation set and the training set are the same, which is not the case for the training and the test set. By using a larger validation set, or by applying a cross-validation scheme, the correlation between the performances on the validation set and the test set may be increased. So the second version of the *simulation* methods may still be useful when using more representative validation sets, as very good performances on the validation set were achieved.

size of transformed feature vector								
1	2	3	4	5	6	7	8	9
26.83 %	49.91 %	61.91 %	62.85 %	63.13 %	**65.29 %**	64.82 %	65.20 %	65.10 %

Table 6.4: Recognition rates of the classifiers trained and tested on feature vectors transformed by PCA. The performances of the classifier that use the original features is 66.23 %.

In summary, all methods but the *simulation* methods with $N = 1$ are promising. If the training set and validation set are large, the *parameters* and *simulation* methods should be preferred, as their time complexity is low. As the *simulation* methods did not produce significantly different results for $N = 2, 3, 4, 5$, it is not expected that the performance will increase when higher numbers for N are used. Therefore only the values $N = 4$ and $N = 5$ will be tested in the experiments in Chapter 7.

6.7 Experiments with PCA

In this section the results of the application of the PCA method described in Section 3.4 with the experimental setup (3a) in Table 6.1 are shown. The PCA method transforms the feature vectors and reduces their size. Table 6.4 shows the results of the classifiers trained and tested on PCA transformed feature vectors, where the size of the transformed feature vectors is varied from 1 to 9. The recognition rates were measured on the test set, where the classifier using the original feature vectors achieved a performance of 66.23 %.
All results obtained when using PCA to transform the feature vectors were worse than the results of the classifier using the original feature vectors. Please note that the best feature vector size is a priori not known, and must be determined in a validation procedure. Even with knowledge of the best feature vector size, the system's performance could not be increased when using PCA. It seems that the PCA method is not promising for the classifier described in Section 2.2 and it will therefore not be applied in the experiments in Chapter 7 for this classifier.

6.8 Experiments to determine the optimal state numbers

In this section the methods described in Section 3.1 are applied using the experimental setup (5) in Table 6.1. The validation set was used to estimate the optimal values of the parameters for the two methods. The optimal values obtained in [103] ($f = 0.4$ for Bakis and $q = 0.04$ for quantile method) were regarded as references and only values close to them were examined.
In Figure 6.2 the recognition rates measured on the validation set are displayed, where only values near to the optimum are shown. The recognition rate of the const-state system, the system using a fixed number of 14 states per HMM, was 66.23 %. For both $q = 0.028$ and $q = 0.032$ the quantile method achieved a optimal recognition rate of 70.73 %. Here it is reasonable to choose $q = 0.028$, because performances of the classifiers using q values near 0.028 are much better than those near 0.032. This indicates that the measured performance for $q = 0.028$ is more reliable. The optimal performance when using the Bakis method was 70.92 %.

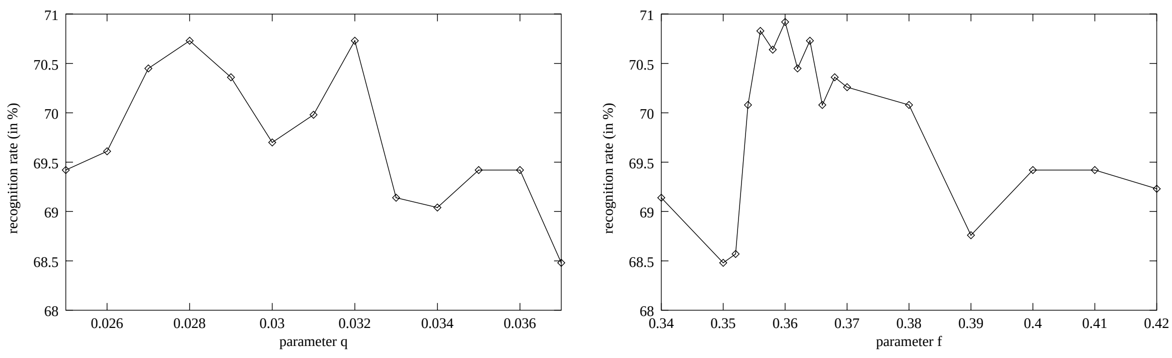


Figure 6.2: Recognition rates of the classifiers optimized by the quantile (left) and the Bakis (right) method with different values for parameters q and f on the validation set. The recognition rate of the original classifier was 66.23 %.

As the Bakis method produced better results, this method was used to choose the number of states of the HMMs for the recognition of the words in the test set. The optimal value of 0.36 was of course used for the parameter f. The recognition rate of the optimized system was 72.46 %, whereas the const-state system only recognized 70.91 % of the test words correctly. The difference in the recognition rates between the optimized and the base system dropped from 4.69 % in the validation set to 1.55 % in the test set. It seems that the Bakis method overfit on the validation set and it is also likely that the quantile method suffered from overfitting effects. Please note that only a small validation set of 1066 words was used. By using larger validation sets, or by employing a cross-validation scheme, the overfitting effects may be reduced. The obtained performance increase on the test set is still quite large and statistically significant using a significance level of 7 %. Therefore the methods for the optimization of the state numbers will be applied in the experiments in Chapter 7. As the Bakis method is not significantly better than the quantile method, both methods will be tested.

6.9 Optimization order

In Section 3.1, methods to optimize the state numbers of HMMs were given, and in Section 3.2 strategies to find the optimal training method for HMM classifiers were described. For the final system we want to optimize both the state numbers and the training method. Because it is not very feasible to optimize both at the same time, first the state numbers and then the training method is optimized in this thesis. In this section, results of an experiment are given which show that the optimization of the training method does not strongly interfere with the optimality of the state numbers of the HMMs. The methods described in Section 3.2 were applied to the following two systems:

- **14-states**: The base system described in Section 2.2 where all HMMs have 14 states
- **12-states**: The base system described in Section 2.2 with a lowered number of 12 states per HMM

system	strategy	validation	test
12-states	1, $C = 3$	76.64 %	78.96 %
14-states	1, $C = 2$	78.33 %	80.30 %
12-states	1, $C = 4$	76.83 %	78.12 %
14-states	1, $C = 3$	78.71 %	79.04 %
12-states	1, $C = 5$	76.74 %	77.86 %
14-states	1, $C = 4$	78.14 %	79.07 %
12-states	3	77.02 %	77.07 %
14-states	3	78.14 %	80.43 %

Table 6.5: Recognition rates of the two classifiers with different state numbers per HMM. The classifiers use optimized training methods found by the different strategies.

The experimental setup (5) in Table 6.1 was used. The performance of the classifier using one Gaussian on the validation set is 65.1 % for the *12-states* system, and 66.23 % for the *14-states* system. The same systems achieved recognition rates of 69.06 % and 70.91% on the test set. In the experiments in this section the classifiers are trained on the training set only, i.e. the validation set is only used for the validation procedure. Only the first and the third strategy were tested[2]. For the first strategy the values for C for which the best results on the validation set were achieved are shown as well as the next smaller and next larger numbers.
There are values for several parameters that belong to the methods in Section 3.2 which must be set. The parameter GAUSSIANS was set to 5, 6, 7, 8 and 9 for the first and to 6 for the third strategy. For the first strategy the two constants MINITER and MAXITER were set to 20 and 60, respectively, as the same constants were set to 5 and 20, respectively, for the third strategy. The values of the constants and the parameter ranges are based on results of preliminary experiments using smaller sets of words.
It seems reasonable to assume that the optimal number of states per HMM will drop when using more Gaussians per state, as the states with Gaussian mixtures are better in modeling more complex distributions. So we may expect that the *12-states* system improves considerably in respect of the *14-states* system.
In Table 6.5 the results of the experiment are shown. In column *system*, the classification system used for the test is given. The entries in column *strategy* denote the strategy for finding the optimal training method. The recognition rates of the systems on the validation and on the test set are given in the columns *validation* and *test*, respectively.
The results of the *14-states* system are always better than the corresponding results of the 12-system. Using the sign test, we discover that this finding is significant using a significance level of 7%. So the *14-states* system is still better than the *12-states* system even when Gaussian mixtures are used. The optimization of the training method did not change the optimality of the 14-state system. This indicates that in general the optimization of the training method does not have a strong impact on the optimality of the state numbers of the HMMs. So the approach used in this thesis to first optimize the state numbers and then to optimize the training method

[2]The second strategy produced inferior results in the experiments in Section 6.10. Please note that the chronological order of the experiments was not the same as their order in this Chapter.

state	strategy	training iterations	gauss	valid	test 1	test 2
const.	1, $C = 2$	4,2,2,2,2,2,13	7	78.33 %	80.30 %	81.22 %
var.	1, $C = 2$	4,2,2,2,2,2,11	7	79.74 %	81.02 %	82.05 %
const.	1, $C = 3$	4,3,3,3,3,32	6	78.71 %	79.04 %	79.68 %
var.	1, $C = 3$	4,3,3,3,19	5	80.58 %	79.18 %	80.53 %
const.	1, $C = 4$	4,4,4,4,4,57	6	78.14 %	79.07 %	79.96 %
var.	1, $C = 4$	4,4,4,4,4,10	6	80.21 %	82.13 %	81.07 %
const.	2	4,0,0,0,0,45	6	76.17 %	77.35 %	78.66 %
var.	2	4,0,0,0,0,34	6	79.27 %	80.47 %	81.02 %
const.	3	4,12,14,15,15,8	6	78.14 %	80.43 %	80.73 %
var.	3	4,11,12,17,11,16,12	7	80.86 %	80.93 %	81.27 %

Table 6.6: The optimized training methods and the performance of the trained classifiers

does not seem to be flawed.

6.10 Methods for selection of optimal training strategy

In this section results of the methods described in Section 3.2 using the experimental setup (5) in Table 6.1 are shown. Unlike the experiments in the last section, where the impact of the optimization of the training methods on the optimality of the state numbers was discussed, the main goal of this section is to compare the different optimization strategies. The same parameter ranges and constants as in the last section were used where the second strategy used the same values as the first strategy.
The three strategies to optimize the training method were applied to following two systems:

- **const-state**: The base system described in Section 2.2, where all HMMs have 14 states.
- **var-state**: The system found to be optimal in the experiments in Section 6.8. The Bakis method with parameter $f = 0.36$ was applied to the base system to optimize the state numbers.

Table 6.6 shows a summary of the experimental results. The entry in column *state* denotes the used system. The strategy used is shown under *strategy*. In the column *training iterations* the optimal training method found is described, where the number of training iterations between each increase of the number of Gaussians is given (separated by commas). The number of Gaussians in the trained system is shown in column *gauss*. The performance of the trained classifier on the validation set is shown in column *valid*. There are two possibilities for using the discovered training method to train the final system: Firstly, the classifier may be trained on the training set only. In the second case classifiers are trained on the union of the training and validation set. Results of classifiers trained using the first possibility are shown in column *test1* and results of classifiers trained using the second possibility are presented in column *test2*.
For all strategies the *var-state* system did better than the *const-state* system, i.e. in 5 out of 5 times the *var-state* system was superior. Using the sign test we discover that this finding

is significant using a significance level of 4%. Please note that the superiority of the *var-state* system holds for all three recognition results *valid, test1* and *test2*. So it could be argued that not only in 5 out of 5, but in 15 out of 15 times the *var-state* system was better than the *const-state* system. Yet all pairs of corresponding results are assumed to be independent in the sign test which may not be the case for the three recognition rates.
In 9 out of 10 tests the performance of the system trained on both the training and the validation set was better than the system trained only on the training set. When we again use the sign test we conclude that this finding is significant using a low significance level of 2 %.
The best results overall (82.13 % for *test1* and 82.05 % for *test2*) were achieved using the first strategy. Unfortunately there is little correlation among the optimal value of C for the two test versions, i.e. *test1* and *test2*, and for the validation set. When using the optimal value of the validation set ($C = 3$), quite low recognition rates on the test set were obtained with both test versions. Therefore the problem with the first strategy is that the best value for the constant C is a priori unknown. If the validation set is very similar to the test set, e.g. if the words of both sets originate from the same writers, we may assume that the optimal value for parameter C is the same for the validation and test set. However in the experiments shown, the set of the writers of the validation and test set were disjoint. This explains the need for different values of parameter C for the test and validation sets.
For the second and third strategy no constants must be determined. The results produced by the third strategy were always better than those of the second strategy. By assuming the independence of the three different recognition results of the same strategy (which, as mentioned before, may not be the case), we conclude that this finding is significant using a significance level of 2%.
In summary it may be stated that the first strategy should be used when the validation set is similar to the test set; otherwise the third strategy is recommended.
In the experiments in Chapter 7, the best strategies found in this section were used: the *var-state* approach in finding the optimal number of states per HMM, the training on the union of the validation and training set, and the third strategy to find the best training method. The third strategy in finding the training method was used because, due to two disjoint populations of writers in the experiments in Chapter 7, the validation and test sets are not expected to be similar.

6.11 Time and memory complexities and performance

In this section the time and memory complexities and the performances of three versions of the classifier described in Section 2.2 are examined. Only the time and memory complexity of the recognition process was addressed and the size of the N-best list, the list of classes output by the classifiers, is varied. The experiments were done with the setup (5) in Table 6.1.
The three tested classifiers are:

1. The base classifier as described in Section 2.2.

2. The base classifier trained with the training method found by the third strategy introduced in Section 3.2. This classifier corresponds to the classifier whose results are shown in the

second last row in Table 6.6.

3. The base classifier whose state numbers are optimized using the Bakis method with a value of 0.36 for parameter f. is trained with the training method found by the third strategy introduced in Section 3.2. This classifier corresponds to the classifier whose results are shown in the last row in Table 6.6.

For each system, all patterns of the validation set were recognized and an N-best list was produced for each word. The size N of the output list was set to the following values: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 30, 40, 50, 75, 100 and 125. No values larger than 125 were tested for N, because in HTK [102] the maximal value for N is 128.
The Figures A.1, A.2 and A.3 show the results of the experiments. For each tested size N the time complexity of the recognition of all validation words in seconds, the average memory complexity during the recognition in kilo Bytes and the recognition rate is measured. A test pattern is regarded as correct if the correct word class is in the output list. The memory complexity is estimated by averaging about 10 check samples and is therefore not very reliable. Please note that the Figures have a logarithmic scale for the x-axis.
The behavior when varying N of the time and memory complexities and the performance is very similar for the three systems. The time complexity increases strongly from $N = 1$ to $N = 2$, increases again significantly from $N = 2$ to $N = 3$, and then remains roughly constant. This means that a system outputting only the best class is much faster than a system outputting an N-best list with $N > 1$, yet the size of the output list when doing an N-best recognition with $N > 2$ does not have a strong impact on the time complexity. The memory complexity increases rapidly until $N = 30$ and then remains constant. It is noteworthy that there are large variations due to the unreliable measurement process. The recognition rate also increases strongly at first, but the performance gain slows down for high N. For example from $N = 75$ to $N = 125$ there is no improvement for all three systems.
The performance of system 3 is always better than that of system 2 which in turn has a better performance than system 1. This ordering also applies to the memory complexity for almost all values of N. The time complexities of the systems are quite similar. System 1 was slightly faster for all values of N than the other two systems which have comparable time complexities.
The three systems have similar time complexities, but different memory complexities. This is due to the fact the actual case the memory complexity is not that important, as state-of-the-art computers have far more memory than needed by the recognition process. Therefore it is straightforward to always use the system with the best performance, which is system 3. A system which output only the best class is much faster than a system which output a list containing several classes. Therefore a system designer must consider the trade-off between the benefit of having an N-best list output and the speed of the system.

6.12 Use of N-best list instead of best result

There are several classifier combination schemes which work with classifier outputting N-best lists (compare Subsection 5.2.3). In the experiments in the last section it was observed that a system which outputs only the best class is much faster than a system which output an N-best

combination scheme	1st rank	2nd rank	3rd rank
vote	1	0	0
borda2	2	1	0
borda3	3	2	1
rank5-3-1	5	3	1

Table 6.7: The numbers assigned to the ranks when using different combination schemes

list. In this section, results are presented which were used to decide whether the benefit of producing N-best list outputs, and applying one of the above referenced combination schemes, justifies the higher time complexity of the classifiers.
The three ensemble methods Bagging, AdaBoost and Half and half Bagging which are all described in Section 4.2 were used in the experiments. The experimental setup (1c) in Table 6.1 was used and the results were averaged over three runs of the ensemble methods. All classifiers produced a 3-best list output.
The classifiers produced by the ensemble methods were combined by four combination schemes:

- **vote**: Same as *vote reject* described in Section 6.2.
- **borda2**: The Borda count combination described in Subsection 5.2.1 was applied to the 2-best list. Please note that the class on the third rank of the output list is ignored.
- **borda3**: The Borda count combination described in Subsection 5.2.1 was applied to the full 3-best list.
- **rank5-3-1**: The combination defined by the equation 5.4 was applied where $w_i(\text{rank}) = 5 - 2 \cdot \text{rank}$ and $a_j = 0$. As in Borda count, a number is assigned to each rank. Here the number is 5 for the first rank, 3 for the second rank and 1 for the third rank.

In the case of a tie, i.e. if several classes have the same maximal sum of assigned numbers, the pattern is rejected. In Table 6.7 the numbers assigned to each rank for all combination schemes are shown. For the AdaBoost ensemble method the number assigned to the rank is multiplied by the weight of the classifier calculated by the AdaBoost algorithm. Therefore weighted versions of the above described combination schemes are used for AdaBoost.
In Table 6.8 the results of the experiments are given. The values in brackets are the percentages of rejected patterns, i.e. patterns for which a tie occurred[3]. The values in column *cn* denote the number of classifiers produced by the ensemble methods.
The *vote* and *borda2* combination schemes obtained comparable recognition rates, but when using the *vote* scheme a higher number of ties occurred. A higher number of ties will normally lead to a higher number of correct results when a ties breaking mechanism other than ties rejection is applied. Therefore the *vote* combination produced superior results to the *borda2* scheme. The *borda3* and *rank5-3-1* combination schemes produced inferior results to the other schemes. In summary the *vote* combination scheme produced better results than the three combination

[3]There are no ties in the results of the AdaBoost algorithm, because the weights of the classifiers have continuous values.

algorithm	cn	vote	borda2	borda3	rank5-3-1
Bagging	10	80.4 (2.6) %	80.63 (1.13) %	80.27 (0.77) %	80.53 (0.4) %
AdaBoost	10	82.03 %	81.9 %	81.73 %	81.93 %
h. & h. Bagging	20	81.4 (1.87) %	81.27 (0.83) %	80.73 (1) %	81.23 (0.2) %

Table 6.8: Recognition rates for the different combination schemes

schemes using N-best list outputs. It could be that more sophisticated combination schemes which use N-best list outputs than the ones tested in this section would achieve better recognition rates, but the results in Table 6.8 indicate that the performance gain over the *vote* combination scheme would not be large. As the classifier which output an N-best list is much slower than the classifier which output just the best class, only combination schemes working with the best class output by the classifiers will be used in the following.

6.13 Adapted score schemes

In this section the adaption of the classic score schemes described in Section 5.3 are evaluated. The experimental setup (1d) in Table 6.1 was used. The recognition rate of the base classifier described in Section 2.2, and trained on the whole training set is 81.24 % for this setup. The following ensemble methods were tested:

- **Bagging**: As described in Subsection 4.2.1. 10 classifiers were produced for the ensemble.
- **rsm**: Random Subspace Method as described in Subsection 4.2.3.
- **AdaBoost**: As described in Subsection 4.2.2. 10 classifiers were produced for the ensemble.
- **H. & h. Bag**: Half and half Bagging as described in Subsection 4.2.5. Here 20 classifiers were produced.
- **archi**: Architecture variation as described in Subsection 4.2.4.

The ensemble methods were combined by the combination schemes *score*, *vote reject*, *vote random* and *vote best*. In addition the following two combination schemes were tested:

- **class red.**: The class reduced score combination scheme as described in Subsection 5.3.1.
- **vote c. red.**: The voting combination as described in Subsection 5.2.1, with ties handling by the class reduced score combination scheme.

Similar to the *vote best* schemes, there exists a version of the *class red.* and *vote c. red.* combination schemes for each of the four operators average, maximum, minimum and median. In Table 6.9 the results of the experiments are summarized. The recognition rates are given as relative change to the recognition rate of the base classifier which was 81.24 %.
It is noteworthy that the recognition rates of the *score*, *vote best max* and *class red max* schemes are different. In theory all three combination methods output the class for which the highest

combination	Bagging	rsm	AdaBoost	h. & h. Bag	archi
score	-0.63 %	-0.17 %	1.43 %	0.91 %	-30.82 %
class red. ave	-1.54 %	-9.78 %	-2.4 %	-3.66 %	-15.72 %
class red. max	0.46 %	-4.06 %	1.43 %	1.14 %	-31.45 %
class red. min	-4.29 %	-10.29 %	-9.09 %	-11.15 %	-29.16 %
class red. med	-0.06 %	-10.29 %	-1.09 %	0.17 %	-10.58 %
vote reject	0.11 %	0.97 %	-0.12 %	0.57 %	-1.26 %
vote random	0.63 %	1.94 %	1.26 %	1.37 %	0.0 %
vote best ave	1.43 %	2.4 %	1.83 %	1.31 %	-0.69 %
vote best max	1.37 %	2.11 %	2.0 %	1.54 %	-0.57 %
vote best min	1.26 %	2.23 %	1.83 %	1.26 %	-0.86 %
vote best med	1.54 %	2.23 %	1.77 %	1.26 %	-0.92 %
vote c. red. ave	0.69 %	1.71 %	1.71 %	1.26 %	-0.17 %
vote c. red. max	1.43 %	1.83 %	1.94 %	1.54 %	-0.74 %
vote c. red. min	0.57 %	1.49 %	1.31 %	0.86 %	-0.11 %
vote c. red. med	0.69 %	1.54 %	1.6 %	1.14 %	-0.11 %

Table 6.9: Recognition rates of the adapted score combination schemes relative to the base classifier

score among all classifiers was achieved. This difference is caused by the pruning mechanism of the HTK.

It is obvious that the *class red.* combination scheme produced very poor results. The results of the *vote best* and *vote c. red* combination schemes were always better than the corresponding results of the *class red.* scheme. In addition the results of the *class red.* scheme were only 1 in 20 times better than the *vote random* scheme. The maximum score scheme also produced quite inferior results. Only for AdaBoost was a slightly better performance achieved than for the *vote random* scheme, but for all other ensemble methods it was worse, sometimes significantly. It seems that the voting schemes are superior to schemes which only take the score values of the results into account. From these observations it is concluded that the *class red.* combination scheme is not suitable for the actual application and this scheme is therefore not considered in the following experiments.

For comparison purposes the results of the *vote best* and *vote c. red* combination schemes are plotted in Figure 6.3. Results obtained from an ensemble method are shown at the same x-coordinate. The order of the ensemble methods is the same as in Table 6.9.

For all ensemble methods except the architecture variation, the results of the *class best* scheme were better than the corresponding results of the *vote c. red.* scheme. Please note that all eight results of both schemes are inferior to the results of the *vote random* scheme for the *archi* ensemble method. The reason for this is that the classifiers produced by the *archi* ensemble method have very different ranges of score values, so combination schemes relaying on the score values tend to perform very poorly. This is also shown by the extremely poor results for the *class red.* scheme. In summary, the *class best* combination method is always better than the *vote c. red.* scheme with the exception of one ensemble method where both combination methods

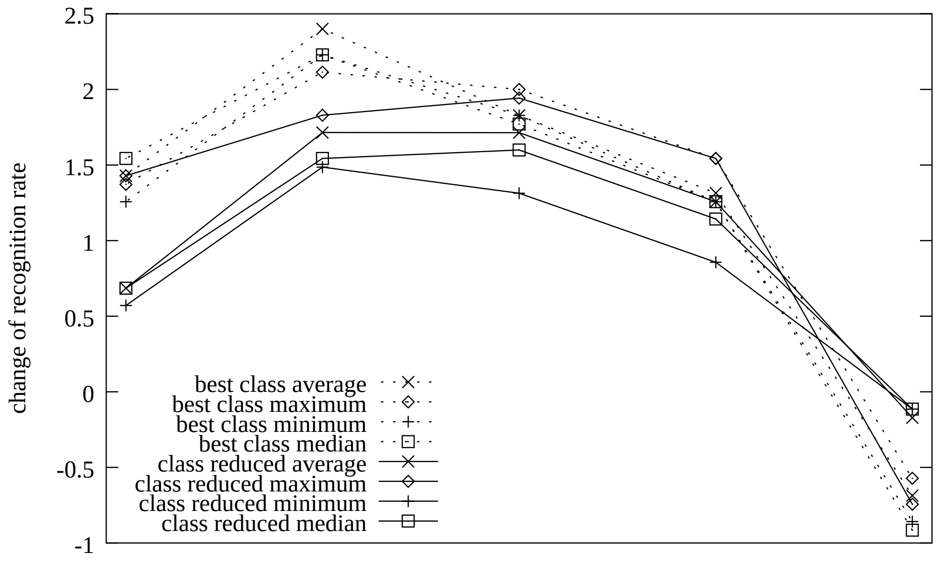


Figure 6.3: Comparison of the voting schemes with ties handling by the best class score combination scheme and the class reduced score combination scheme

are poor and other combination schemes should be used. Therefore it is obvious that the *vote c. red.* combination scheme is inferior to the *vote best* scheme for the actual application, and it was not considered in the following experiments.

The *vote best* combination scheme produced better results than the *vote random* scheme for all ensemble methods but *archi*. This indicates that this combination scheme successfully extracts additional information from the score to improve the recognition rate. The minimum and median versions produced only good results for the *archi* ensemble method, as average and maximum versions were superior than those two versions for all other ensemble methods. As the *vote best* combination scheme is not suited for the architecture variation ensemble method, the use of the minimum and median version of the *vote best* scheme is not promising.

The results of the *class red* and *vote c. red.* combination schemes are disappointing. A possible explanation for their poor performance is the following: the score value of the best class recognized by the classifier contains some additional information, as it can be seen by the superiority of the *vote best* combination scheme over the *vote random* scheme. Yet the score values of classes which are not the best class output by the classifier do not contain a lot of additional information. The two combination schemes *class red* and *vote c. red.* both make their decision partially on such score values, therefore the decision is usually not based on additional information, but on random values.

Please note that the best class score combination scheme was not tested (the *vote best* scheme is the voting combination scheme with ties handling by this scheme). On the basis of the results of the *class red*, *vote c. red.* and *score* schemes it was decided to skip this test, because all the combination schemes which were based only on the score values of the results did not produce good results.

combination	Bagging	AdaBoost	rsm	archi
score	64.92 %	64.26 %	62.1 %	46.06 %
conf max	67.35 %	66.6 %	66.04 %	62.76 %
vote best ave	67.45 %	68.2 %	67.54 %	67.35 %
vote best max	67.54 %	67.82 %	67.64 %	66.98 %
vote best min	67.45 %	68.39 %	67.64 %	67.26 %
vote best med	67.54 %	68.11 %	67.45 %	67.26 %
vote conf ave	67.73 %	67.92 %	67.54 %	67.82 %
vote conf max	67.64 %	67.73 %	67.45 %	67.82 %
vote conf min	67.73 %	68.2 %	67.35 %	67.82 %
vote conf med	67.73 %	68.11 %	67.35 %	67.82 %

Table 6.10: Comparison of the confidence combination schemes to some score combination schemes. The recognition rate of the base classifier is 66.23 %.

For AdaBoost the weighted voting combination scheme using weights produced by the algorithm was also tested (compare Subsection 4.2.2), and an improvement of the recognition rate of 0.97 % was obtained. As this is much lower than the increases obtained for all voting schemes except the *vote reject* scheme, the weighted voting combination scheme using weights produced by the algorithm will not be used in following experiments.

6.14 Confidence combination schemes

In this section the confidence combination schemes described in Subsection 5.3.4 are compared to some score combination schemes. For the experiments in this section the experimental setup (3a) in Table 6.1 was used. The applied ensemble methods are Bagging, AdaBoost, random subspace method and architecture variation (compare Section 4.2). Random subspace method and architecture variation will be abbreviated as *rsm* and *archi* in the following. For the *archi* ensemble method 8 classifiers were produced, whereas for all other ensemble methods 10 classifiers per ensemble were used.
The ensemble methods were combined by the combination schemes *score* and *vote best*. In addition the following two combination schemes were tested:

- **conf max**: The maximum confidence combination scheme as described in Subsection 5.3.4.
- **vote conf** : The voting combination scheme with ties handling by the confidence combination scheme. Similar to the *vote best* scheme, there exists a version of this scheme for each of the four operators.

The results of the experiments are given in Table 6.10.
It is obvious that the *conf max* combination scheme did much better than the *score* combination scheme. This shows that the used confidence value is a more reliable measure of the quality of the recognition than the score value, when only the score and confidence values and not the occurrences of the classes are considered. The *vote conf* combination schemes are better than the

corresponding score schemes for Bagging and architecture variation, but not for AdaBoost and the random subspace method. This means that the confidence measure does not always provide more information than the score value for the ties breaking mechanism. Although superior to the *score* combination, the performance of the *conf max* scheme is always lower than the performances of all *vote conf* and *vote best* combination schemes, therefore it is not suitable for the considered application.
Please note that the minimum, median and average confidence score combination schemes were not tested. The reason for this is that the poor performance of the *conf max* combination scheme indicates that they too would not produce good results.
To calculate the confidence of a result the classifier must output a 2-best list. There are two reasons why the *vote conf* combination scheme does not seem to be suitable for the considered application. Firstly, the classifiers which output a 2-best list are much slower than the classifiers which output the best class only, as it was observed in Section 6.11. In addition the *vote conf* combination schemes are not significantly better than the *vote best* combination schemes. As the *vote best* combination scheme does not seem to be suitable for the considered application it was not used in the following experiments.

6.15 Evaluation of normalization of adapted score schemes

In this section the normalization procedure introduced in Subsection 5.3.3 is evaluated. The base classifier as described in Section 2.2 was used and the ensemble methods Bagging, AdaBoost and random subspace method were applied in the experiments (compare Section 4.2). The classifiers were combined by two versions of the *vote best* combination scheme:

1. **unnormalized vote best**: The original *vote best* combination scheme as defined in Section 6.2.

2. **normalized vote best**: The *vote best* combination scheme normalized by the procedure introduced in Subsection 5.3.3, and defined by equation 5.8. Please note that the normalization procedure only affects the test patterns for which a tie has occurred.

The setup (3a) in Table 6.1 was used for the experiments in this section and a total of 29 classifiers were produced for each ensemble. After the production of each classifier by the ensemble method, the ensemble of already created classifiers was combined by the above mentioned schemes, i.e. for each ensemble method the combination schemes were applied 29 times. To obtain stable results all ensemble methods were run three times and the results averaged over the three runs. The results of the experiments are shown in the Figures. A.4-A.15. There are no large differences in the performance of the different versions of the combination schemes. The maximum version seems to be slightly superior to the other versions for AdaBoost.
For Bagging and AdaBoost there is no significant difference between the normalized and unnormalized *vote best* combination schemes. For random subspace method the unnormalized schemes clearly outperform the normalized ones when using more than 9 classifiers. Therefore the normalization procedure did not improve the performance of the combination schemes and therefore the unnormalized versions of the *vote best* combination schemes will be used in the following.

A possible reason why the normalization did not work is the following. Well performing classifiers tend to produce results with higher scores than poorly performing classifiers. Therefore the results of well performing classifiers are privileged. The scores are normalized in such a way that for all classifiers a result with average score value has a score of 0. Therefore a result of a good performing classifier with an average score is treated in the same way as the result of a poor performing classifier with an average score, and we in turn loose the preference of the well performing classifiers.
The normalization procedure was not tested for the "pure" adapted score combinations, i.e. the schemes which just consider the score value and not the occurrences of the classes in the output of the classifiers, because of two reasons: Firstly, as it was shown in the results in this section, the normalization procedure does not increase the performance of the *vote best* combination scheme. In addition those combination schemes produced very poor results (compare Section 6.13).

6.16 Confidence based classifiers reduction scheme

In this section the confidence based classifiers reduction (CBCR) combination scheme introduced in Section 5.5 is evaluated. The results in this section originate from the same experiments as the results in Section 6.14. In Figs. A.16, A.17, A.18 and A.19 the results of the *CBCR* combination scheme are shown. Only the combination schemes which have been found to perform well in the last sections are used as underlying combination schemes. In the figures, the x-axis shows the rejection threshold t. The graphics in the figures only show the relevant parts of all evaluated threshold values, i.e. the parts where the highest performance was achieved.
From the results the following can be observed:

- For the *ga weight* and *vote ga* combination schemes only small improvements were achieved. The reason for this is that the weights were optimized for the combination of all classifiers and they are suboptimal for subsets of the whole ensemble.

- The plots of the different versions of the *vote best* combination scheme are very similar. At least one global maximum is shared by all four versions.

- The global maximum of the recognition rate depends on the ensemble method.

- The quality of the combination schemes depends on the ensemble method. For Bagging, the *vote perf.*, *perf. weight* and *best vote max* schemes did well. As for AdaBoost the *vote perf.* and *perf. weight* schemes also produced good results, but the best version of the *vote best* scheme was the median version. For random subspace method good results were obtained for the maximum version of *best vote*, which were comparable to the results of the four combination schemes using weights. The *perf. weight* combination scheme produced the best results for the architecture variation ensemble method, while the results for all versions of the *best vote* scheme were quite poor. The *perf. weight* combination scheme was among the best schemes for all ensemble methods.

- The best recognition rates were always obtained using rejection thresholds lower than 0.375. Please note that a threshold of 0.375 means that classifiers for which the confidence is lower than 0.375 of the average confidence are not used for the combination. There are two reasons why the optimal thresholds are so low:

- The improvement of the recognition rate in respect of the base classifier is achieved by correctly classifying test patterns for which the classifiers output different classes. In such cases the classifiers normally have low confidence values because those patterns are difficult to classify.
- There is a tradeoff between the quality of the classifiers and the quantity of classifiers in the ensemble. If the results of too many classifiers are rejected then any combination scheme will perform poorly, even if the individual classifiers perform well. The optimal rejection threshold seems to be very low, i.e. the best ensemble performance is achieved when results with very low confidence values are rejected.

• If the recognition rate of the best combination schemes using all classifiers are compared to the recognition rate of the CBCR combination scheme, using the best underlying combination scheme and the best threshold, the following increases of the recognition rate were observed: 0.84 % for Bagging, 0.19 % for AdaBoost, 0.75 % for random subspace method and 0.2 % for architecture variation. Therefore the increases in the recognition rate are rather small.

The confidence based classifiers reduction combination scheme is able to increase the performance of the underlying combination scheme. Yet the increases are quite small. Furthermore the optimal threshold t is a priori unknown and has to be determined, e.g. by using a validation set. A large disadvantage of the combination scheme tested in this section is that for each result the confidence must be calculated. To calculate the confidence a 2-best list must be computed and classifiers outputting 2-best lists are much slower than classifiers outputting only the best class (compare Section 6.11). As the CBCR combination scheme increased the recognition rate of the underlying combination schemes only slightly, but the time complexity of the recognition process was much larger for it, this scheme will not be used in following experiments.

6.17 Experiments with special HMM combination scheme

In this section the special HMM classifier combination scheme introduced in Section 5.4 is compared to other combination schemes. The experimental setup (2) in Table 6.1 is used. The following experiments were conducted:

- **Bagging**: The experimental setup (2e) was used and the Bagging ensemble method was applied to produce 10 classifiers.
- **AdaBoost**: The experimental setup (2e) was used and the AdaBoost ensemble method was applied to produce 10 classifiers.
- **3 cluster**: The experimental setup (2a) was used. A clustering algorithm was applied and the words of each cluster, which were not in the test set, were used to train one classifier. The applied clustering algorithm used the features of the number of components per word

and the ratio of the heights of the two baselines to create three clusters of words with similar writing styles. Details of the clustering algorithm are reported in [70].

- **writers 3*2**: The experimental setup (2a) was used. The writers are grouped in pairs and to each of the three classifiers a pair was assigned. The classifiers are then trained on all words of the training set belonging to one of the writers of the pair assigned to it.
- **writer 6*1**: The experimental setup (2a) was used. Each classifier is trained on all words of the training set belonging to one writer.
- **2 cluster**: The experimental setup (2d) was used where the same clustering algorithm as in *3 cluster* is applied. Two classifiers were each trained on the words of one cluster.
- **writer 2*2**: The experimental setup (2c) was used. The four writers whose words are in the training set are grouped in pairs and to each of the two classifiers a pair was assigned. The classifiers are then trained on all words belonging to one of the writers of the pair assigned to it.
- **writer 5*1**: The experimental setup (2b) was used. Each classifier is trained on all words belonging to one writer.

The experiments *3 cluster* and *writers 6*1* were repeated three times and the results are averaged over the three runs. For experiment *writers 3*2* the results are averaged over all possibilities of grouping the writers in pairs. Please note that there is also an implicit averaging given by the used experimental setup for *Bagging*, *AdaBoost*, *3 cluster*, *writer 2*2* and *writer 5*1*. The approaches applied in this section to train the individual classifiers on subsets of the training set, are very similar to the application of the ensemble method using a partitioning of the training set, which is described in Section 4.3. Yet, there is one significant difference: in the experiments in this section the whole data set, i.e. the union of the training set and test set, instead of just the training set, is partitioned. The produced classifiers all satisfy the two conditions for the application of the special HMM classifier combination scheme. The word HMMs are composed of character HMMs and the feature vectors input to the HMMs are the same for all classifiers. The classifiers are combined by the *score*, *vote best max* and *special* combination schemes. The schemes *score* and *vote best max* were chosen as representations for the other combination schemes because of two reasons. Firstly, those schemes do not need the testing of the classifiers on the training set, which is similar to the *special* combination scheme but unlike the *ga vote*, *perf. weight* and *ga weight* schemes. In addition those schemes produced the best results most of the time.
Table 6.11 shows the results of the experiments. The column *base* contains the recognition rate of the classifier which is trained on the whole training set. This classifier will be denoted as base classifier in the following. The entries in column *wi* indicate whether the experiment was writer independent. The number in column *cn* is the number of classifiers combined in the experiment. For each experiment where the *special* combination scheme obtained the best result, it was tested whether the superiority of the *special* combination scheme over the base classifier and the other combination schemes is statistically significant. In column *sig* the corresponding significance levels are shown.

experiment	wi	cn	base	score	vote best max	special	sig
Bagging	no	10	85.27 %	85.45 %	86.25 %	85.07 %	-
Adaboost	no	10	85.27 %	86.97 %	87.25 %	86.48 %	-
3 cluster	no	3	85.13 %	89.36 %	86.7 %	90.33 %	11 %
writer 3*2	no	3	85.39 %	87.77 %	86.53 %	88.47 %	< 0.1 %
writer 6*1	no	6	85.87 %	88.1 %	81.67 %	90.1 %	1 %
2 cluster	yes	2	69.46 %	69.63 %	69.63 %	72.64 %	0.2 %
writer 2*2	yes	2	75.67 %	74.46 %	74.46 %	77.17 %	< 0.1 %
writer 5*1	yes	5	78.1 %	73.35 %	68.63 %	79.17 %	13 %

Table 6.11: Recognition rates of the special HMM classifier combination scheme compared to other schemes

The results for the *score* and *voting best max* schemes are identical in the sixth and seventh row. This is no coincidence: it occurs because there are only two classifiers. If the two classifiers have the same result, both schemes choose this class, and if there are different results, the one with the best score is selected in both cases.

For *Bagging* and *Adaboost* the classifier combination with the *vote best max* scheme was the best one, and the *special* combination scheme was outperformed by both *score* and *vote best max* schemes. It seems that the special HMM classifier combination scheme is not suited tor the combination of classifiers trained on arbitrary subsets of the training set. However for all other experiments, the *special* combination scheme was better than the other two combination schemes. Similarly the *score* combination scheme is only inferior to the *vote best max* scheme for *Bagging* and *Adaboost*, while for the other experiments the *score* scheme achieved the same or better recognition rates than the *vote best max* scheme.

The most remarkable gain in performance when using the *special* combination scheme in respect of other schemes was achieved for the writer independent experiments. For those experiments both *vote best max* and *score* schemes perform worse or only marginally better than the base classifier, but the *special* combination scheme leads to a clear performance improvement over the base classifier.

In summary it may be stated that the *special* combination scheme is not suited for combination of classifiers trained on arbitrary sets, yet it obtains good results when combining classifiers trained on sets consisting of words with a similar writing style. This is especially the case for writer independent tests. Please note that in the experiments in this section a data set consisting of words produced by only 6 writers was considered. In Section 6.21 the question of whether the above mentioned observations are also true for data sets produced by a large number of writers is studied.

The *special* combination scheme has the disadvantage that the time complexity of the recognition process of the combined classifier is higher than the sum of the time complexities of the recognition process of the individual classifiers. This means that the recognition process when using the *special* combination scheme is slower than when using other combination schemes. In the experiments in this section, and in Section 6.21, it was observed that the recognition using the *special* combination scheme had a 1.2 to 2.5 higher time complexity than when using other combination schemes.

6.18 Evaluation of classic ensemble methods

In this section the best parameter values of the classic ensemble methods are determined and the results of the ensemble methods using these best parameter values are compared. The first parameter of an ensemble method is the number of classifiers. The second parameter is the used combination scheme. In the previous sections the quality of some combination methods were evaluated. It was observed that many combination schemes introduced in Chapter 5 are not suited for the considered application and therefore they will not be applied to the experiments in this section. The results of the same experiments as in Section 6.15 are evaluated. The three ensemble methods Bagging, AdaBoost and random subspace method combined by the seven combination schemes *vote reject*, *vote random*, *vote best*, *vote ga*, *perf. weight* and *ga weight* are compared. Because the results of the different versions of the *vote best* combination scheme were very similar, as noted in Section 6.15, only the maximum version was compared to the other schemes. The results of the *vote perf.* combination scheme are not shown, because they were always the same as the results of the *perf. weight* combination scheme. The results of the *vote reject* scheme are given to illustrate the benefit of the ties breaking mechanism when using the other voting schemes. The architecture variation ensemble method is addressed separately in Section 6.19, because this ensemble method has a fixed number of classifiers.
First the three schemes using weights are compared. The results are shown in Figures A.20, A.21 and A.22. For Bagging and AdaBoost the *perf. weight* scheme seems to be the best combination scheme. The *ga weight* combination scheme often produced very good results, especially for Bagging, but very poor ones were also obtained. Therefore, the more stable *perf. weight* combination scheme should be preferred. For random subspace method the *ga weight* combination scheme was clearly superior.
In Figures A.23, A.24 and A.25 the best found combination schemes using weights are compared to the other combination schemes. From the results the following can be observed:

- For Bagging and AdaBoost the *perf. weight* scheme seems to be the best combination scheme.

- The *vote best max* combination scheme did worse than the *vote random* scheme for Bagging and the two schemes had similar performance for AdaBoost. It seems that the score information is not very useful in order for these ensemble methods to break ties.

- For random subspace method the results of the *ga weight* combination scheme were almost always superior to the results of the other schemes, so it should be the preferred combination scheme for this ensemble method, despite its instability. It seems that here the score value of the results holds some useful information for the ties breaking mechanism because the *vote best max* combination scheme was clearly better than the *vote random* scheme.

- The number of 21 classifiers seems to be a good choice for Bagging. There are no significant improvements when adding more classifiers and the best result for the *perf. weight* combination scheme was achieved with this number of classifiers.

ensemble method	combination scheme	classifier num.	performance	sig
Bagging	*perf. weight*	21	68.23 %	5 %
AdaBoost	*perf. weight*	14	69.11 %	1 %
random subspace m.	*ga weight*	25	**69.35**%	0.5 %

Table 6.12: Best parameters of the classic ensemble methods and the performance of the classifiers using these parameters. The performance of the base classifier was 66.23 %.

- The number of 14 classifiers seems to be a good choice for AdaBoost. There is very little improvement when adding more classifiers and one of the best results for the *perf. weight* combination scheme was achieved with this number of classifiers.

- For random subspace method the best performance when combining the results by the *ga weight* scheme was achieved with 25 classifiers, so this number of classifiers should be used in future tests. Please note that the choice of 25 classifiers is not as obvious as the choice for Bagging and AdaBoost, because it is close to the maximal tested number.

The summary of the best found parameters is given in Table 6.12. Those parameters will be used in the experiments in Chapter 7. The entries in column *sig* in Table 6.12 show the significance level of the statistical test of the superiority of the ensemble methods over the base classifier. Please note that the performance of the ensemble methods may be lower when testing on other sets, as the parameter values were optimized on the results of the test set. The random subspace method was slightly better than AdaBoost, while Bagging was clearly inferior to the other ensemble methods.

6.19 Evaluation of the architecture variation

In Section 6.18 the ensemble methods Bagging, AdaBoost and random subspace method were evaluated. In this section we examine the results of the architecture variation ensemble method. The results of the same experiments as in Section 6.14 were used. Please note that the architecture variation ensemble method as described in Section 4.2.4 has a fixed number of 8 classifiers, so the number of classifiers is not a free parameter. The results of the different combination schemes are given in Figure 6.13. The combination scheme which obtained the best results is the *weight ga* and all schemes using weights did far better than the other schemes. A reason for this is the large difference in the performances of the individual classifiers. (The individual classifier performances are 66.23 %, 41.46 %, 57.88 %, 49.36 %, 66.23 %, 44.56 %, 46.53 % and 47.69 %). The superiority of the best recognition rate, 68.67 %, over the performance of the base classifier is statistically significant using a significance level of 12%.
Despite the good results obtained for the architecture variation, this ensemble method will not be used in the experiments in Chapter 7 because for following reasons:

- The performance of the method is inferior to two of the other three classic ensemble methods (compare Table 6.12).

vote reject	vote random	best vote ave	best vote max	best vote min
64.82 %	67.45 %	67.35 %	66.98 %	67.26 %

best vote med	vote ga	vote perf.	weight perf.	weight ga
67.26 %	68.57 %	68.57 %	68.39 %	**68.67** %

Table 6.13: Results of the architecture variation ensemble method

- The number of classifiers is fixed, i.e. it is not possible to produce larger ensembles or to optimize the number of classifiers.
- The Bakis, semi-jumpin and semi-jumpout topologies (compare Subsection 4.2.4) are clearly very inferior to the linear topology. This resulted in the production of very poor performing individual classifiers. It is more promising to combine classifiers with different architectures which all perform quite well. The results of such combinations will be presented in the subsections 6.24 and 6.25.

The experiments with the architecture variation ensemble method were however not futile, because we have learnt the following facts from the results:

- The combination schemes using weights, i.e. the *vote ga*, *vote perf.*, *weigh perf.* and *weight ga* schemes, are good for combining classifiers with very different individual performances. This is especially the case for the *vote ga* combination scheme.
- The combination of classifiers with different architectures may increase the performance of the best individual classifier even when there are very poor performing classifiers in the ensemble. By using the ensemble of classifiers described in Subsection 4.2.4 the recognition rate of the best individual classifier, which is the base classifier, could be increased from 66.23 % to 68.76 %, although the average performance of the individual classifiers was 52.49 %.

6.20 Varying training set and vocabulary sizes

In this section the dependence of the classic ensemble methods Bagging, AdaBoost and random subspace method on the training set and vocabulary size is studied.

6.20.1 Dependence of classic ensemble methods on vocabulary size

In this subsection the dependence of the performance of three classic ensemble methods on the vocabulary size is evaluated. The tested ensemble methods are Bagging, AdaBoost and the random subspace method and the experimental setup (4) in Table 6.1 was used. Four different vocabularies were tested which contained 651, 998, 1,518 and 2,296 words. The first vocabulary is the original vocabulary of experimental setup (4) and the fourth vocabulary is the vocabulary of experimental setup (3). The second and third vocabularies were produced by adding randomly words from the fourth vocabulary to the first. To obtain stable results the ensemble methods

were run three times for all vocabularies and the results were averaged over the three runs. 10 classifiers were always produced by each ensemble method.
The results of the experiments are shown in Figures A.26, A.27 and A.28. The results of the combination of the ensembles by the *vote best*, *perf. weight* and *ga weight* combination schemes are given. As the results of the different versions of the *vote best* scheme were very similar, only those of the maximum version are shown. The *vote perf.* combination scheme again produced the same results as the *perf. weight* scheme, therefore the results of this scheme are not included in the figures. The *vote ga* combination scheme obtained recognition rates between those of the *perf. weight* and *ga weight* schemes. To simplify the figures the results of the *vote ga* combination scheme were also not included.
From the results the following can be observed:

- For Bagging and AdaBoost the performance of the three combination schemes is very similar. For AdaBoost the differences in the combination schemes decreases with the vocabulary size.

- For Bagging and AdaBoost the improvement of the ensemble methods in respect of the base classifier increases with the size of the vocabulary. This increase is much larger for AdaBoost, i.e. the superiority of AdaBoost over Bagging increases with the vocabulary size.

- For random subspace method the behavior of the performance improvement over the base classifier when increasing the vocabulary size depends on the combination scheme. The improvement slightly decreases for the *vote best max* combination scheme, but remains roughly constant for the *perf. weight* scheme. For the *ga weight* combination scheme the improvement increases, similar slightly which is similar to Bagging.

- For random subspace method the *ga weight* combination scheme was superior to the other combination schemes.

In summary it can be stated that with increasing vocabulary size the improvement over the base classifier increases. For Bagging and AdaBoost the results of all combination schemes show this behavior, whereas for random subspace method only the *ga weight* scheme does. The increase in improvement is largest for AdaBoost, indicating that this ensemble method is the most able to deal with large vocabularies.

6.20.2 Dependence of classic ensemble methods on training set size

In this subsection the dependence of the performance of three classic ensemble methods on the training set size is evaluated. The tested ensemble methods are Bagging, AdaBoost and the random subspace method and the experimental setup (3a) in Table 6.1 was used. For the training of the classifiers of the ensembles, new training sets are randomly sampled from the original training set. The sizes of the new sets were 300, 1,000 and 3,000. In addition to those sampled sets the original training set was also used in one experiment. To obtain stable results the ensemble methods were run three times for all training set sizes and the results are averaged

over the three runs. Please note that for each run the training set was re-sampled. 10 classifiers were always produced by each ensemble method.
The results of the experiments are shown in Figures A.29, A.30 and A.31. For the same reasons as mentioned in the last subsection only the results of the combination of the ensembles by the *vote best max*, *perf. weight* and *ga weight* schemes are shown.
From the results the following was observed:

- For the smallest training set Bagging and AdaBoost obtained worse results than the base classifier. When increasing the training set size both algorithms generally improve relative to the base classifier.
- The improvement of the performance of random subspace method in respect of the base classifier slightly decreases when larger training sets are used.
- Bagging is better than AdaBoost for the smallest training set size, but the performance of AdaBoost increases faster than that of Bagging when the training set is enlarged. This means that AdaBoost takes more advantage of larger training sets than Bagging.
- Random subspace method is very good for small training sets, unlike Bagging and AdaBoost. An obvious reason for this is that the individual classifiers are trained on the whole training set, whereas Bagging and AdaBoost train the classifiers on sets sampled from the already small training set.
- The *perf. weight* and *ga weight* combination schemes are inferior to the *vote best max* scheme for Bagging and AdaBoost when small training sets are used. A reason for this is that those methods estimate the performance of the classifiers on the training set. When using small training sets, the estimated performances may be very different from the real performances because of overfitting effects. It is noteworthy that yet for random subspace method those combination schemes perform well even when using small training sets.
- For all but the largest training set size the *ga weight* combination scheme was the best scheme to combine classifiers produced by random subspace method.

In summary it can be stated that when increasing the training set size the improvements of the ensemble methods in respect of the base classifiers slightly decrease for random subspace method and increase for Bagging and AdaBoost. The increase for AdaBoost is larger than the increase for Bagging.

6.21 Ensemble method using partitioning of training set

In this section the results of the ensemble method using partitioning of the training set as described in Section 4.3 are studied. The focus of this study is the comparison of the special HMM classifier combination scheme introduced in Section 5.4 to other combination schemes. The setup (3a) in Table 6.1 was used for the experiments. All five different feature sets were used for the clustering of the words and the number of clusters was varied from 2 to 5. As each classifier is trained on a separate cluster, the number of clusters is the same as the number of

			feature sets used for the clustering					
cn	combination	random	{CPW}	{WPC}	{WC}	{CPW,WC}	{WPC,WC}
2	score	66.26 %	68.76 %	68.48 %	66.67 %	66.7 %	66.32 %
2	special	66.67 %	**69.70** %	$\underline{69.32}$ %	67.64 %	67.64 %	67.35 %
3	score	65.23 %	68.67 %	**69.7** %	67.26 %	66.89 %	68.67 %
3	special	64.54 %	$\underline{69.14}$ %	$\underline{69.32}$ %	67.64 %	67.26 %	$\underline{69.04}$ %
3	vote best	67.07 %	67.82 %	68.67 %	68.11 %	66.6 %	68.76 %
4	score	65.79 %	**69.51** %	$\underline{69.32}$ %	66.14 %	66.98 %	68.86 %
4	special	65.82 %	68.01 %	$\underline{69.23}$ %	66.04 %	66.89 %	68.86 %
4	vote best ave	67.17 %	68.57 %	67.26 %	67.82 %	67.54 %	**69.51** %
4	vote best max	67.04 %	68.39 %	67.26 %	67.64 %	67.45 %	$\underline{69.42}$ %
4	vote best min	67.07 %	68.48 %	66.42 %	67.73 %	67.82 %	**69.7** %
4	vote best med	67.07 %	68.48 %	66.42 %	67.73 %	67.82 %	**69.7** %
5	score	64.13 %	68.01 %	**69.61** %	66.89 %	66.04 %	**69.51** %
5	special	63.57 %	67.26 %	$\underline{69.32}$ %	66.42 %	66.14 %	$\underline{\textbf{70.83}}$ %
5	vote best ave	67.51 %	67.63 %	68.57 %	$\underline{69.14}$ %	67.82 %	$\underline{\textbf{70.08}}$ %
5	vote best max	67.48 %	67.45 %	68.48 %	68.86 %	67.45 %	**69.79** %
5	vote best min	67.51 %	67.73 %	68.01 %	68.76 %	67.82 %	**69.98** %
5	vote best med	67.51 %	67.73 %	68.01 %	68.76 %	67.82 %	**69.98** %

Table 6.14: Results of the ensembles whose classifiers are trained on the clusters produced by the k-means algorithm with the indicated features. The performance of the base classifier is 66.23 %.

classifiers. Please note that the produced classifiers all satisfy the conditions for the application of the *special* combination scheme: The word HMMs are composed of character HMMs and the features input to the HMMs are the same for all classifiers.

The classifiers produced by the ensemble method are combined by the *score*, *vote best* and *special* combination schemes. Only those schemes were tested, as the best results were obtained with them and they do not need the time consuming testing of the classifiers on the training set. In Table 6.14 the results of the experiments are shown. For two classifiers the *score* combination scheme is the same as all versions of the *vote best* combination scheme, therefore only the results of the first scheme is given. Similarly all versions of the *vote best* combination scheme are the same when three classifiers are combined, therefore only one result for the *vote best* scheme is given in that case. The entries in the column *cn* are the number of classifiers. For comparison purposes the ensemble method using partitioning of the training set was also applied on randomly sampled clusters and the results of these experiments are shown in column *random*. The random clusters used were all the same size and the experiments were repeated three times to get more stable results.

The best results in Table 6.14 are marked in the following way: Recognition rates between 69 % and 69.5 % are underlined, recognition rates between 69.5 % and 70 % are written in bold type and recognition rates over 70 % are both written in bold type and underlined.

From the Table 6.14 the following may be observed:

- Even for random clusters the performance of the base classifier could be increased when using the *vote best* combination scheme. The performance decreased most of the time when the ensemble was combined by other schemes.
- 80 out of 85 results obtained when using clustering were better than the corresponding results using random clusters, which shows that the training of classifiers on clusters of words with similar writing styles is promising.
- The performance of the *special* combination scheme relative to *score* scheme varies with the number of of clusters. For two classifiers the *special* scheme always achieved better results, for three classifiers the *special* scheme was better in 4 of 5 cases. For 4 and 5 classifiers the *score* scheme outperforms the *special* scheme in most cases. It is noteworthy that yet the overall best result is achieved using 5 classifiers and the *special* combination scheme.
- From the 24 results with recognition rate over 69 % 8 were achieved using the *special* combination scheme. As only 20 out of 85 results[4] were produced by the *special* combination scheme, the expected value would have been 5.65. This indicates that the *special* combination scheme is on average better than the other schemes. Please note that the superiority of the *special* combination scheme is not so clear as in the experiments in Section 6.17. It seems that the *special* combination scheme performs better if the clusters contain words of only a few writers.
- The clustering with the feature sets {CPW,WC} and {WC} seems to be inferior to the clustering with the other feature sets: for the first feature set no result with a higher performance than 69 % was achieved and for the second only one was produced. All other feature sets achieved at least three such results.

The two feature sets {CPW,WC} and {WC} will not be used in the following experiments. For small numbers of classifiers the *special* combination scheme should be used and for higher number of classifiers the *score* combination scheme seems to be more suitable. As there is no real significant difference between the performances of the different combination schemes, all of them will be used in the experiments in Chapter 7.

6.22 New boosting algorithms

In this section the new boosting algorithms introduced in Section 4.4 are evaluated. The setup (3a) in Table 6.1 was used for the experiments. The applied ensemble methods are the simple probabilistic boosting (SPB), the effort based boosting (EBB) and the simple probabilistic boosting with effort (SPBE). For SPB and SPBE all four error functions e_1, e_2, e_3 and e_4 were tested. For EBB two values for parameter α for which good results were achieved in preliminary tests were used. To compare the new methods to classic ensemble methods, also Bagging and

[4]The results obtained when using random clusters are not considered.

algorithm	error/α	vote best max	perf. weight	ga weight
Bagging	-	67.51 %	67.73 %	67.89 %
AdaBoost	-	68.17 %	68.67 %	68.35 %
SPB	e_1	68.45 %	68.7 %	68.67 %
SPB	e_2	68.29 %	68.23 %	68.64 %
SPB	e_3	68.04 %	68.45 %	68.2 %
SPB	e_4	68.17 %	68.17 %	68.26 %
EBB	1	69.11 %	69.11 %	69.29 %
EBB	1.5	69.42 %	69.45 %	69.82 %
SPBE	e_1	69.2 %	69.01 %	69.04 %
SPBE	e_2	67.85 %	67.7 %	68.32 %
SPBE	e_3	68.95 %	68.76 %	68.42 %
SPBE	e_4	69.04 %	69.36 %	69.04 %

Table 6.15: Results of the experiments with the new boosting algorithms. The recognition rate of the base classifier is 66.23 %.

AdaBoost were tested. Each ensemble method was applied three times and the results were averaged over the three runs. The number of classifiers produced by the ensemble methods was set to 10.

To simplify the comparison of the methods only the results of the combination of the classifiers by the *vote best max*, *perf. weight* and *ga weight* combination schemes are given. The other versions of the *vote best* scheme produced similar results to the *vote best max* scheme and the performance of the *vote ga* scheme was in between the performances of the *perf. weight* and *ga weight* combinations schemes. In Table 6.15 the results of the experiments are shown. The entries in column *error/α* denote the function used for the error calculation, or the value of the parameter α.

No combination scheme produced significantly better results than the other schemes. For almost all boosting algorithms and all combination schemes better results than Bagging were achieved. Only the performance of SPBE with error function e_2 when combined by the *perf. weight* scheme was marginally below that of Bagging. This shows that the boosting algorithms really produce superior results to Bagging. In the following the new boosting algorithms are only compared to AdaBoost. The SPB algorithm produced comparable or inferior results for error functions e_2, e_3 and e_4 than AdaBoost. Only with error function e_1 is SPB slightly better. EBB produced very good results, especially for $\alpha = 1.5$. It is obvious that SPB is not significantly better than AdaBoost, so we focus on the EBB and SPBE in the following. The two error functions e_1 and e_4 seem to be suited for use with the SPBE algorithm and lead to very good results. With error function e_3 better results than AdaBoost were also achieved, however when using e_2 SPBE cannot compete with AdaBoost.

To compare the EBB and SPBE ensemble methods more systematically with AdaBoost the sign test (see Section 6.3.3) was used. Results of an ensemble method are regarded as better than the results of another ensemble method if the recognition rate of the first ensemble method is

better than the recognition rate of the second for all three tested combination schemes. EBB and SPBE produced results that were in five cases better, and in one case worse, than the results of AdaBoost. The finding that the EBB and SPBE ensemble methods are better than AdaBoost is significant using a significance level of 11%. If we compare the results of the three combination schemes separately we observe that in 15 out of 18 results EBB and SPBE are better than AdaBoost. This finding is significant using a significance level of 0.5%. However the prerequisite of the sign test, the independence of the corresponding pairs of events, is not satisfied for this test version as the results of the same ensemble combined by different combination schemes are strongly correlated.
Only EBB (with $\alpha = 1.5$) and SPBE (without error function e_2) will be used in the experiments in Chapter 7 as the results of SPB indicate that this ensemble method is not promising for the considered application. The number of classifiers of the ensemble methods will be set to 14 for the experiments in Chapter 7, because this number was found to be optimal for AdaBoost in Section 6.18. So the same number of classifiers is used for all boosting methods.

6.23 Feature selection ensemble methods

In this section experiments of the ensemble methods using feature selection algorithms introduced in Section 4.5 are presented. In Subsection 6.23.1 results of the ensemble methods using underlying feature search algorithms are studied. In Subsection 6.23.2 ensemble methods which use the results of all feature sets are tested. Finally ensemble methods which simulate the results for the patterns of the validation set are studied in Subsection 6.23.3.

6.23.1 Feature selection ensemble methods using feature search

In this Subsection the methods described in Subsections 4.5.1 and 4.5.2 are evaluated. The setup (3b) in Table 6.1 was used for the experiments. The following algorithms were applied:

- **b. single**: The backward feature search introduced in Subsection 3.3.1 for the selection of one feature subset
- **f. single**: The forward feature search introduced in Subsection 3.3.1 for the selection of one feature subset
- **b. ensemble**: The ensemble method introduced in Subsection 4.5.1 with the backward feature search as underlying feature selection algorithm
- **b. ensemble**: The ensemble method introduced in Subsection 4.5.1 with the forward feature search as underlying feature selection algorithm
- **b./f. ensemble**: The ensemble method introduced in Subsection 4.5.2 where the backward and forward feature search are alternatively applied starting with the backward version
- **rsm**: The random subspace method as described in Subsection 4.2.3

algorithm	f	ave	max	min	med	valid	sets
b. single	-	classifier performance: 67.73 %				73.08 %	25
f. single	-	classifier performance: 68.76 %				71.48 %	45
b. ensemble	f_1	69.04 %	68.76 %	69.14 %	69.14 %	74.39 %	115
f. ensemble	f_1	69.89 %	70.17 %	69.31 %	69.42 %	71.95 %	111
b./f. ensemble	f_1	69.89 %	69.89 %	69.79 %	69.79 %	74.3 %	142
b. ensemble	f_2	69.89 %	69.98 %	70.08 %	70.17 %	74.3 %	128
f. ensemble	f_2	70.64 %	70.17 %	70.73 %	70.73 %	74.2 %	165
b./f. ensemble	f_2	71.76 %	71.58 %	71.76 %	72.04 %	75.33 %	185
b. ensemble	f_3	68.86 %	68.76 %	68.95 %	68.95 %	74.86 %	127
f. ensemble	f_3	70.83 %	70.83 %	71.2 %	70.83 %	74.39 %	171
b./f. ensemble	f_3	70.92 %	70.92 %	71.1 %	71.2 %	75.61 %	173
b. ensemble	f_4	69.41 %	69.51 %	68.86 %	69.23 %	74.86 %	130
f. ensemble	f_4	71.48 %	71.39 %	71.58 %	71.86 %	74.58 %	144
b./f. ensemble	f_4	70.36 %	70.36 %	70.36 %	70.36 %	74.58 %	175
rsm	-	ga weight: 68.76 %				-	0
original classifier: 66.23 % on test set, 70.45 % on validation set							

Table 6.16: Results of the feature search ensemble methods

The methods *b. single*, *f. single* and *rsm* are examples of classic feature selection algorithms and ensemble method. 10 classifiers were always produced for each ensemble. The ensemble methods *f. ensemble*, *b. ensemble* and *b./f. ensemble* will be denoted as feature search ensemble methods in the following.

The classifiers were combined by the four versions of *vote best* combination scheme. The combination schemes using weights, i.e. the *vote ga*, *perf. weight* and *ga weight* schemes, were outperformed by the *vote best* scheme and their results are therefore not shown. A reason why those schemes did not produce good results is that the ensemble methods introduced in Subsections 4.5.1 and 4.5.2 optimize the performance of the voting combination scheme.

The results of the experiments are shown in Table 6.16. The recognition rate of the base classifier which uses the full set of features was 66.23 % on the test set and 70.45 % on the validation set. The entries in column f indicate the objective function used for the feature search ensemble methods. The values in column *ave*, *max*, *min* and *med* denote the recognition rates of the ensembles combined by the *vote best ave*, *vote best max*, *vote best min* and *vote best med* schemes, respectively. For the *b. single* and *f. single* methods the performance of the produced classifier is given in these columns, as for the random subspace method they contain the name of the best combination and the performance achieved using this scheme. The entries in column *valid* are the recognition rates on the validation set of the single classifiers or the ensembles combined by the *vote reject* scheme. Please note that this is the value which is optimized by the methods. Finally *sets* denotes the number of feature sets for which the recognition results of all elements of the validation set were calculated. For the random subspace method the performance on the validation set is not given because this method does not use a validation set.

All tested methods increased the performance of the base classifier. The results of the feature search ensemble methods are always better than or equal to those of the other methods. The median version of the *vote best* combination scheme is clearly the best version, as in 9 out of 12 experiments the best result was achieved with it and in 5 out of the 12 experiments this best result was not obtained by any other combination scheme. If we compare the median version of *vote best* scheme to the other three versions, in 22 cases it was better and in 5 cases worse than the other versions. With the sign test we find that the discovery that the median version is better than the other versions is significant with a significance level of 0.1 %. Therefore the classifiers produced by the feature search ensemble methods will be combined by the *vote best median* combination scheme in the experiments in Chapter 7.

The *f. ensemble* method has clearly shown a better performance then its backward counterpart *b. ensemble*. A further improvement is achieved for the f_2 and f_4 objective functions when using *b./f. ensemble* method where the two search feature search methods are combined[5]. The best overall recognition rate of 72.04 % was also achieved by the *b./f. ensemble* method. It may be expected that the use of several feature selection algorithms as is done in the *b./f. ensemble* method will achieve even better results when using algorithms with similar performances. Here, as mentioned before, the methods based on the forward feature search produced much better results than the methods based on the backward feature search. A drawback of the *b./f. ensemble* method is that a very large part of the feature space is explored which leads to a high time complexity.

The objective function f_1 produced inferior results. Of the 12 results of this objective function 9 were worse than the corresponding results of all other objective functions. The reason for this poor performance is that with objective function f_1 very similar classifiers are generated (compare Section 4.5.1). The objective function f_2 seems to be slightly better than functions f_3 and f_4 as for both the *b. ensemble* and *b./f. ensemble* methods the best results were achieved with it. However there is no significant difference between the results of the three objective functions f_2, f_3 and f_4. For the overall best result an increase in the recognition rate of the base classifier of 5.81 % was achieved, which is much higher than for other ensemble methods (compare for example Tables 6.12 and 6.14). The superiority of this best result over the base classifier is statistically significant using a significance level of 0.2 %.

An important observation is that the performance on the validation set is not strongly correlated to the performance on the test set. For example it seems that the methods using the forward feature search are always underestimated. The reason for this lack of correlation may be that the validation set is too small. In the experiments in Chapter 7 a partial cross-validation scheme will be applied which may solve this problem.

In the experiments in Chapter 7 all three ensemble methods *b. ensemble*, *f. ensemble* and *b./f. ensemble* will be applied using the objective functions f_2, f_3 and f_4. The classifiers will be combined by the *vote best med* combination scheme. The reason why the *b. ensemble* method is also retested, despite its inferior performance, is that the superiority of the forward feature search over the backward feature search may be an artifact of the small validation set.

[5]For the *vote best median* combination scheme the *f. ensemble* method is slightly better than *b./f. ensemble*, but for the other three combination schemes the statement is true.

algorithm	f	ave	max	min	med	valid	time
std. exhaustive	f_1	70.08 %	70.08 %	70.08 %	70.08 %	74.39 %	< 0.2
replace exhaustive	f_1	69.42 %	69.51 %	68.95 %	68.95 %	75.52 %	1.3
std. exhaustive	f_2	68.95 %	69.51 %	68.67 %	68.67 %	75.05 %	< 0.2
replace exhaustive	f_2	69.7 %	69.7 %	69.7 %	69.7 %	75.52 %	1.5
std. exhaustive	f_3	70.54%	70.83 %	69.88 %	70.08 %	75.52 %	< 0.2
replace exhaustive	f_3	69.23 %	69.41 %	69.04 %	69.23 %	75.61 %	1.3
std. exhaustive	f_4	69.42 %	69.04 %	69.14 %	69.23 %	75.52 %	0.4
replace exhaustive	f_4	70.08 %	70.08 %	69.98 %	70.08 %	75.89 %	4
original classifier: 66.23 % on test set, 70.45 % on validation set							

Table 6.17: Results of the ensemble methods using exhaustive feature search

6.23.2 Feature selection ensemble methods using exhaustive search

In this subsection the ensemble methods introduced in Subsection 4.5.3 are evaluated. The setup (3b) in Table 6.1 was used for the experiments. There are two versions of the ensemble method using exhaustive feature search: the standard approach and the approach with replacement of feature sets. The first will be denotes as *std. exhaustive* and the other as *replace exhaustive* in the following. Each ensemble method was used to create 10 classifiers.

The classifiers were combined with the four versions of the *vote best* combination scheme. The combination schemes using weights, i.e. the *vote ga*, *perf. weight* and *ga weight* schemes, were outperformed by the *vote best* scheme and their results are therefore not shown. The reason for their inferior performance was already stated in Subsection 6.23.1.

The results of the ensemble methods are shown in Table 6.17. The meaning of the entries in the table is the same as in Table 6.16. In an additional column *time* the time complexity of the algorithms are given. This time complexity was measured in multiples of the time needed for the validation of one feature set (more information on the topic of time complexities is presented in Section 6.11). Please note that for any of the algorithm all 511 feature sets must be validated, i.e. the preliminary step which calculates the data needed by the algorithms already has the time complexity of 511[6].

All tested methods increased the performance of the base classifier. The maximum version of the *vote best* scheme seems to be the best version. If we compare it to the other three versions, in 13 cases it was better and in 3 cases worse than the other versions. With the sign test we find that the discovery that the maximum version is better than the other versions is significant with a significance level of 4 %. The classifiers produced by the two ensemble methods considered in this subsection will be combined by the *vote best max* combination scheme in the experiments in Chapter 7.

The performances of the created ensembles on the validation set increase with the complexity of the objective function. Also the *replace exhaustive* method achieved better or equal performance on the validation set than the standard version. Unfortunately the recognition rates on the validation set correlates poorly to the recognition rates on the test set, similar to the results of

[6]Of course this data must only be calculated once for all algorithms.

the last subsection. The *replace exhaustive* ensemble method produced on average slightly worse results than the standard version and there is no significant difference between the results of the different objective functions. So it may be concluded that the *replace exhaustive* method and the more complex objective functions are able to find better solutions on the validation set, but do not obtain higher recognition rates on the test set because they overfit on the validation set. If we compare the results in Table 6.16 to the results in Table 6.17 we see that feature search ensemble methods had in general lower recognition rates on the validation set, but on average much higher recognition rates on the test set than the methods evaluated in this subsection. This is quite surprising as in the *std. exhaustive* and *replace exhaustive* ensemble methods the results of all 511 feature sets are considered. The poor performance of the methods evaluated in this subsection may once again be due to overfitting.
In the experiments in Chapter 7 a partial cross-validation scheme will be applied which may reduce the overfitting effect and increase the performance of the *std. exhaustive* and *replace exhaustive* ensemble methods on the test set. As no objective function and no version has shown to be significantly superior to the others, both *std. exhaustive* and *replace exhaustive* ensemble methods will be retested with all objective functions in the experiments in Chapter 7.

6.23.3 Feature selection ensemble methods using simulations

In this subsection heuristic versions of the ensemble methods tested in the previous two subsections are evaluated. The used heuristic is described in Subsection 4.5.4. It works by simulating the results of the patterns of the validation set. The experimental setup (3b) in Table 6.1 was again used and the ensemble methods are abbreviated by the same terms as in the two last subsections. For the feature search ensemble methods the objective function f_1 was not tested, as inferior results were achieved with this function in the experiments of Subsection 6.23.1. In addition the results of the *f. ensemble* method are not shown as they were clearly inferior to the results of the other two methods. The reason for the poor performance of the *f. ensemble* method is that the simulated results lead to an overestimation of small feature sets (compare Subsection 4.5.4) and such small feature sets are encountered at the beginning of the forward feature search. For the *std. exhaustive* and *replace exhaustive* methods the objective function f_4 was not used, because it lead to high time complexities of the methods and no significant difference between the results of the different objective functions was found in the experiments of Subsection 6.23.2. 10 classifiers were always produced by each ensemble method.
Only the results of the combination of the classifiers by the *vote best max* and *vote best median* schemes are shown. The *vote best ave* and *vote best min* combination schemes usually produced similar or inferior results to the two versions mentioned above and are therefore not included in the results. The combination schemes using weights, i.e. the *vote ga*, *perf. weight* and *ga weight* schemes, were outperformed by the *vote best* combination scheme in the experiments in the last two subsections and were therefore not applied.
The results of the feature search ensemble methods are shown in Table 6.17. The entries in column f indicate the objective function used for the ensemble methods. The values in the rows starting with *max* and *med* denote the recognition rates of the ensembles combined by the *vote best ave* and *vote best med* schemes. The number N is a parameter of the method which simulates the results of the validation set (compare Subsection 3.3.3).

		b. ensemble					*b./f. ensemble*				
combi.	f	N=1	N=2	N=3	N=4	N=5	N=1	N=2	N=3	N=4	N=5
max	f_2	62.66	65.2	69.79	70.45	71.2	68.29	68.01	68.86	70.07	69.14
med	f_2	62.2	64.82	69.51	70.45	71.39	68.2	67.26	68.57	69.89	69.32
max	f_3	69.04	69.89	70.83	69.32	69.7	70.64	69.51	68.76	70.54	69.51
med	f_4	69.23	69.89	70.45	69.23	69.43	69.7	69.7	68.86	70.45	70.08
max	f_4	69.04	69.7	69.89	71.2	70.08	70.64	68.67	68.95	70.36	68.2
med	f_4	69.23	69.51	69.61	71.29	70.26	68.67	68.67	69.04	70.17	68.86
original classifier: 66.23 % on test set, 70.45 % on validation set											

Table 6.18: Results of the heuristic version of the feature search ensemble methods (in percentages)

		std. exhaustive					replace exhaustive				
combi.	f	N=1	N=2	N=3	N=4	N=5	N=1	N=2	N=3	N=4	N=5
max	f_1	68.85	70.64	70.92	69.79	70.26	68.01	69.61	69.7	69.88	70.45
med	f_1	69.14	70.45	70.92	69.98	70.73	68.29	69.98	69.89	69.79	70.08
max	f_2	67.82	69.32	69.89	70.26	70.17	68.76	67.64	68.39	67.64	68.76
med	f_2	67.64	69.42	69.98	69.89	70.64	68.29	68.01	68.58	68.2	69.42
max	f_3	68.01	71.58	71.39	69.32	69.79	69.32	69.51	68.86	70.26	71.39
med	f_3	67.26	71.48	71.58	69.42	70.54	68.86	69.7	69.04	70.73	71.39
original classifier: 66.23 % on test set, 70.45 % on validation set											

Table 6.19: Results of the heuristic version of the ensemble methods using exhaustive feature search using simulated results (in percentages)

The time complexity of the *b. ensemble* and *b./f. ensemble* methods is the same as the complexity of the *simulation best* method which was tested in Section 6.6 and whose results are shown in Table 6.3. It is measured in multiples of the time needed to test the classifier on the validation set and is 1.6 for $N = 1$ and $1.65 + 0.05 \cdot N$ for all other values of N.

For $N = 4$ and $N = 5$ and objective functions f_2 and f_4 good results were achieved when using the *b. ensemble* method. For $N = 4$ the *b/f. ensemble* method produced good results when used with any objective function. In the experiments in Chapter 7 the ensemble methods will only be retested using these parameters. The obtained recognition rates for these parameters range from 70.08% to 71.29% and from 69.89% to 70.58% for the *b. ensemble* method and the *b./f. ensemble* method, respectively. The results obtained are not much worse than when using the real results (compare Table 6.16), and the time complexity is much lower. The results of the *b. ensemble* method were in general better than those of the *b./f. ensemble* method. It is noteworthy that the *b./f. ensemble* method was also quite good for $N = 1$. This good result is likely to be an outliers, as the performance of all methods using the heuristic applied here was always low for $N = 1$.

The results of the ensemble methods using exhaustive feature search are shown in Table 6.19 where the same notation as in Table 6.18 was used.

The time complexity of these methods is the sum of the time complexity for the simulation of the results and the time used for the exhaustive feature search, which is quantified in Table 6.17. The time complexity for the simulation of the results is the same as the time complexity of the heuristic versions of the *b. ensemble* and *b./f. ensemble* methods.
The results of the *std. exhaustive* ensemble method are in general better than the results of the *replace exhaustive* ensemble method, although a much simpler algorithm is used. The reason for this is that the second algorithm tends to overfit. This overfitting effect is stronger for low values of N, because in this case the ensemble is optimized on the basis of a very inaccurate estimation of the recognition results. In the experiments in Chapter 7 a partial cross-validation scheme will be applied which may reduce the overfitting effect.
For both *std. exhaustive* and *replace exhaustive* method the objective function f_2 leads to results with low recognition rates and therefore the two methods will only be used with the other two objective functions in the following. For the *std. exhaustive* method good results were achieved with $N = 2, 3$ and 5 and the *replace exhaustive* method was best with $N = 5$. In the experiments in Chapter 7 the ensemble methods will only be retested using these parameter values. The recognition rates obtained for these parameters range from 69.79% to 71.58% and from 70.08% to 71.39% for *std. exhaustive* and *replace exhaustive*, respectively. The obtained results are even better than when using the real results of the validation set (compare Table 6.17) and the time complexity is in addition much lower.
The heuristic versions of both tested kinds of ensemble methods produced comparable results to the methods which are based on real results of the validation set, whereas their time complexity is much lower. So the heuristic which simulates the results of the validation set is promising.

6.24 Ensemble methods using several base classifiers

In this Section the extension of classic ensemble methods to methods using several base classifiers introduced in Section 4.6.1 is evaluated. The classic ensemble methods which were extended are Bagging, AdaBoost and random subspace method which will be abbreviated as rsm. In Subsection 6.24.1 an optimized and an unoptimized classifier are used as base classifiers. In Subsection 6.24.2 two optimized base classifiers are used. Conclusions of the experiments in the first two subsections are drawn in the last subsection.

6.24.1 Optimized and unoptimized base classifiers

In this subsection the ensemble methods extended by the method introduced in Subsection 4.6.1 are used with the following two base classifiers:

- $\mathbf{C_1}$: The base classifier as described in Section 2.2.
- $\mathbf{C_2}$: A version of the classifier described in Section 2.2 where the state numbers were optimized by the Bakis method introduced in Section 3.1. As the optimal parameter f of a preliminary experiment is used, the parameter value was not optimal for the test set used in the experiments, and the performance of C_2 is slightly lower than the performance of the best classifier found in the experiments of Subsection 6.8.

algorithm	C	vote best max	perf. weight	ga. weight
Bagging	C_1	67.35 %	67.64 %	67.92 %
Bagging	C_2	70.64 %	70.45 %	70.63 %
AdaBoost	C_1	68.29 %	68.86 %	68.29 %
AdaBoost	C_2	70.64 %	70.17 %	70.45 %
rsm	C_1	67.54 %	68.11 %	68.67 %
rsm	C_2	70.17 %	70.54 %	70.92 %
Bagging	C_1, C_2	71.86 %	71.95 %	71.76 %
AdaBoost	C_1, C_2	70.92 %	71.67 %	71.39 %
rsm	C_1, C_2	71.29 %	71.67 %	71.67 %

Table 6.20: Results of the experimental setup (3a). The recognition rate of the base classifiers C_1 and C_2 is 66.23 % and 70.17 %, respectively.

algorithm	C	vote best max	perf. weight	ga. weight
Bagging	C_1	69.82 %	69.95 %	69.82 %
Bagging	C_2	71.51 %	71.42 %	71.64 %
AdaBoost	C_1	70.95 %	70.7 %	70.89 %
AdaBoost	C_2	72.14 %	72.36 %	72.14 %
rsm	C_1	70.61 %	71.33 %	71.54 %
rsm	C_2	72.33 %	72.83 %	73.08 %
Bagging	C_1, C_2	72.23 %	72.58 %	72.2 %
AdaBoost	C_1, C_2	73.08 %	73.20 %	72.95 %
rsm	C_1, C_2	72.92 %	73.17 %	73.01 %

Table 6.21: Results of the experimental setup (3c). The recognition rate of the base classifiers C_1 and C_2 is 69.17 % and 71.2 %, respectively.

The classifiers were combined by the *vote best max*, *perf. weight* and *ga weight* combination schemes. The results of the *vote ga* scheme are not included as they were once more the same as the ones of the *ga weight* scheme. The results of the other versions of the *vote best* combination scheme are not given as they are very similar to the results of the maximum version.
The two setups (3a) and (3c) in Table 6.1 were used for the experiments. The recognition rate of C_1 is 66.23 % for the first and 69.17 % for the second experimental setup as C_2 obtained recognition rates of 70.17 % and 71.2 %, respectively. The recognition rates of both base classifiers are higher for the second experimental setup than for the first, because in setup (3c) the writers of the words in the test set are the same as the writers of the words in the training set. In each experiment 10 classifiers were produced per ensemble.
The results of the experiments are shown in Tables 6.20 and 6.21. The entries in column C denote the base classifiers used for the experiment. If there is only one classifier then the normal classic ensemble method was used. If the entry contains both classifiers then the algorithm introduced in Section 4.6.1 was applied on the ensemble method indicated in column *algorithm*. In this case 5 classifiers were generated from each base classifier.

From rows 1-6 in Tables 6.20 and 6.21 it can be concluded that the classic versions of Bagging, AdaBoost and random subspace ensemble method lead to an improved performance for all combination schemes when compared to the corresponding base classifiers. The ensemble methods using the base classifier C_2 were always better than the corresponding methods using base classifier C_1. This is not very surprising as C_2 is an optimized version of C_1. The ensemble methods using both base classifiers were in all but one case better than the corresponding methods using base classifier C_2. To check the superiority of the ensemble methods using both base classifiers we apply the sign test. The result of the random subspace method of experimental setup (3c) is not considered here as the ensemble method using both base classifiers is better for the *vote best max* and *perf. weight* combination schemes, but not for the *ga weight* scheme. The finding that the ensemble methods using both base classifiers are better than the ensemble methods which have C_2 as base classifier is significant using a significance level of 4%.
Please note that the base classifier C_1 has a much lower performance than the base classifier C_2. However the replacement of 5 classifiers produced from base classifier C_2 by 5 classifiers produced from base classifier C_1 increase the performance. Therefore the diversity of the 5 classifiers from base classifier C_1 had a larger impact on the performance than their inferior individual performances.
The *ga weight* and *perf. weight* combination schemes seem to be better suited for the combination of classifiers produced by random subspace method than the *vote best max* scheme. Apart from this observation there are no significant differences between the combinations schemes.

6.24.2 Two optimized base classifiers

The setup (3a) in Table 6.1 was used for the experiments. The ensemble methods extended by the method introduced in Subsection 4.6.1 are used with the following two base classifiers:

- **C_3**: A version of the classifier described in Section 2.2 where the state numbers were optimized by the Bakis method introduced in Section 3.1. The recognition rate of C_3 on the test set is 70.92 %.

- **C_4**: A version of the classifier described in Section 2.2 where the state numbers were optimized by the Quantile method introduced in Section 3.1. The recognition rate of C_4 on the test set is 70.71 %.

Unlike classifier C_2 in the last section, the optimal parameter values of the Bakis and Quantile methods were optimized directly on the test set of experimental setup (3a). Both base classifiers have optimized state numbers and they differ only in the applied state number optimization method. To illustrate the difference of the base classifiers C_3 and C_4 the used state numbers are given in Table A.1.
The classifiers were combined by the *vote best min*, *perf. weight* and *ga weight* combination schemes. The results of the *vote ga* scheme are not included as they were once more the same as the ones of the *ga weight* scheme. Only the results of the minimum version of the *vote best* combination scheme are given, because it often achieved the best results of all versions and the results of the different versions were very similar. In each experiment 10 classifiers were produced per ensemble.

algorithm	C	vote best min	perf. weight	ga. weight
Bagging	C_3	71.11 %	71.2 %	70.82 %
Bagging	C_4	70.83 %	71.01 %	70.92 %
AdaBoost	C_3	72.23 %	72.33 %	72.23 %
AdaBoost	C_4	71.39 %	71.76 %	71.67 %
random subspace	C_3	71.29 %	71.01 %	71.29 %
random subspace	C_4	70.26 %	70.45 %	70.08 %
Bagging	C_3, C_4	71.29 %	71.67 %	71.67 %
AdaBoost	C_3, C_4	72.8 %	72.51 %	72.7 %
random subspace	C_3, C_4	71.86 %	71.95 %	71.39 %

Table 6.22: Results of the ensemble methods using two optimized base classifiers. The recognition rate of the base classifiers C_3 and C_4 is 70.92 % and 70.71 %, respectively.

The results of the experiments are shown in Table 6.22. The same notation as in Tables 6.20 and 6.21 is used.
The results of the classic ensemble methods are better when using base classifiers C_3 than C_4. The average difference in the performance of the ensemble methods for C_3 and C_4, about 0.5 %, is larger than the performance difference of the two base classifiers which is 0.21 %. Base classifier C_3 seems to have a larger potential of improvement by the ensemble methods than C_4. The performance of Bagging and random subspace method when using only one base classifier is not very good; in some cases it was worse than the performance of the corresponding base classifier. The poor performance may be due to the fact that the base classifiers were strongly optimized on the test set. AdaBoost did quite well when using only one base classifier and produced the best results when both base classifiers were used.
All ensemble methods using both base classifiers outperform the corresponding algorithms using only one of the base classifiers for any combination rule. When applying the sign test the finding that the ensemble methods using both base classifiers are better than the methods using only one base classifier is significant using a significance level of 2%. (Please note that here the comparison is done with the results of both base classifiers, unlike in the experiments of the last subsection. The reason for doing this is that here both base classifiers were optimized). This shows that the ensemble methods using both base classifiers take advantage of the diversity of the base classifiers.
The *perf. weight* combination scheme was slightly better than the other combination schemes, but the performance difference between the results of the combination schemes is rather small. The finding of the last section that the *ga weight* and *perf. weight* combination schemes are better for random subspace method than the *vote best* scheme could not be confirmed.

6.24.3 Conclusions of the two series of experiments

In almost all cases the ensemble methods using two different base classifiers were better than the ensemble methods using only one base classifier, even when the performance of one base classifier is much lower than the performance of the other. For a better comparison of all results

algorithm	C	vote best max/min	perf. weight	ga. weight
Bagging	C_1, C_2	1.22 / 0.72 %	1.5 / 1.16 %	1.16 / 0.56 %
AdaBoost	C_1, C_2	0.28 / 0.94 %	1.5 / 0.84 %	0.94 / 0.81 %
random subspace	C_1, C_2	1.12 / 0.59 %	1.12 / 0.34 %	0.75 / -0.07 %
Bagging	C_3, C_4	0.18 %	0.47 %	0.75 %
AdaBoost	C_3, C_4	0.57 %	0.18 %	0.47 %
random subspace	C_3, C_4	0.57 %	0.94 %	0.1 %

Table 6.23: Performance increases of the ensemble methods using two base classifiers in respect of the best result of the two corresponding ensemble methods which use only one base classifier

of the ensemble methods using two base classifiers presented in this section see Table 6.23. In this table the performance increases of the methods in respect of the the best result of the two corresponding ensemble methods which use only one base classifier are given. The performance increases are in general larger for the base classifiers C_1 and C_2 than for the base classifiers C_3 and C_4. This is quite surprising as C_3 and C_4 are two optimized classifiers and C_2 has a much lower performance than C_1. The explanation for this observation is that the classifiers C_3 and C_4 are not very diverse. Therefore the diversity of the base classifiers is very important for a good performance of the ensemble methods extended by the algorithm introduced in Subsection 4.6.1. It may be concluded that an ensemble method using two base classifiers which are diverse and which have different performances can produce better results than an ensemble method using two similar base classifiers which have the same performance as the best base classifier of the first ensemble method. It is noteworthy that the best performance of the experimental setup (3a) of 72.8 % was however achieved using the base classifiers C_3 and C_4. (The best result for base classifiers C_1 and C_2 was 71.95 %). In the experiments in Chapter 7 two very diverse base classifiers, namely optimized versions of the classifiers described in Sections 2.2 and 2.3, will be used. The number of classifiers for the extended classic ensemble methods will be set to the same numbers as used for the corresponding ensemble methods using one base classifier (compare Subsection 6.18) to allow an unbiased comparison.

6.25 Multi simple probabilistic boosting

In this section the ensemble method multi simple probabilistic boosting (MSPB) introduced in Subsection 4.6.3 is tested with the experimental setup (3a) in Table 6.1. The four base classifiers C_1, C_2, C_3 and C_4 described in the last section were used. The error function e_1 was not tested as it lead to inferior results is preliminary experiments.
The classifiers were combined by the *vote best ave*, *vote best max*, *perf. weight* and *ga weight* combination schemes. The results of the *vote ga* scheme are not included as they were again the same as the results of the *ga weight* scheme. The average and the maximum versions of the *vote best* combination scheme were usually better than the other versions of this scheme and the performance of the different versions was very similar, therefore only the results of these two versions are given.
The results of the experiments are shown in Tables 6.24 and 6.25. In the first table the results

algorithm	vote best ave	vote best max	perf. weight	ga. weight
Bagging	71.67 %	71.86 %	71.95 %	71.76 %
AdaBoost	70.64 %	70.92 %	71.67 %	71.39 %
MSPB e_2	71.95 %	71.58 %	71.58 %	72.33 %
MSPB e_3	71.48 %	71.29 %	71.29 %	71.01 %
MSPB e_4	72.05 %	72.42 %	72.23 %	72.42 %

Table 6.24: Results of the ensemble methods using base classifiers C_1 and C_2. The recognition rate of the base classifiers C_1 and C_2 is 66.23 % and 70.17 %, respectively.

algorithm	vote best ave	vote best max	perf. weight	ga. weight
Bagging	71.29 %	71.2 %	71.67 %	71.67 %
AdaBoost	72.89 %	72.61 %	72.51 %	72.7 %
MSPB e_2	73.55 %	73.36 %	73.08 %	73.17 %
MSPB e_3	72.33 %	72.61 %	72.23 %	72.42 %
MSPB e_4	72.98 %	72.89 %	72.7 %	72.7 %

Table 6.25: Results of the ensemble methods using base classifiers C_3 and C_4. The recognition rate of the base classifiers C_3 and C_4 is 70.92 % and 70.71 %, respectively.

when using the base classifiers C_1 and C_2 are given, whereas in the second table the results of the experiments are shown where the base classifiers C_3 and C_4 are used. For comparison purposes the results of two classic ensemble methods extended by the algorithm introduced in Section 4.6.1 are also given. The results of these ensemble methods originate from the experiments in the last section. Please note that the main comparison should be done with the extended AdaBoost method, as AdaBoost is the classic boosting algorithm. In each experiment 10 classifiers were produced; 5 for each base classifier.
When using the base classifiers C_1 and C_2 the results of MSPB were almost as good as those of Bagging for error function e_2 and better results than Bagging were achieved for error function e_4. For all error functions MSPB did better than or produced comparable results to AdaBoost. When using the base classifiers C_3 and C_4 the results of MSPB are better than AdaBoost for error functions e_2 and e_4, but worse for e_3.
If we compare the results of MSPB of all combination schemes and error functions to the corresponding results of AdaBoost we observed that MSPB is better in 16 cases and worse in 6 cases. Using the sign test we conclude that the finding that MSPB is better than AdaBoost is significant using a significance level of 3%. Please note that the prerequisite of the sign test, the independence of the different pairs of corresponding results, is not satisfied for this test version as the results of the different combination schemes are strongly correlated, therefore the above mentioned finding is not that reliable. However it is indication that MSPB is a promising ensemble method and that it may outperform AdaBoost.
The recognition rates of MSPB using the error function e_3 were always lower than the corresponding rates of the other two error functions for all combination schemes. This indicates that the error function e_3 is not promising and therefore it will not be used in the experiments in Chapter 7. The number of classifiers produced by MSPB will be set to the same number as used

for the AdaBoost algorithm to allow an unbiased comparison of all boosting algorithms.

6.26 Classifier number for ensemble methods

In previous sections the classifier numbers used in the experiments in Chapter 7 were determined for all tested ensemble methods except the feature selection ensemble methods introduced in Section 4.5. In this section methods to determine the optimal number of classifiers for these ensemble methods are studied. In Subsection 6.26.1 the ensemble methods using feature search algorithms are considered, while in Subsection 6.26.2 the ensemble methods using exhaustive search are addressed. The heuristic ensemble methods introduced in Subsection 4.5.4 will not be considered. For them the optimal methods found for the corresponding non-heuristic ensemble methods will be used in the experiments in Chapter 7. For all experiments in this section the setup (3b) in Table 6.1 was used.

6.26.1 Feature selection ensemble methods using feature search

In this subsection methods to determine the optimal number of classifiers for the feature selection ensemble methods using feature search introduced in Subsections 4.5.1 and 4.5.2 are evaluated. All methods tested in Subsection 6.23.1 were examined using the objective functions f_2, f_3 and f_4. The same notation as in that subsection will be used. Objective function f_1 was not tested as inferior results were produced using this function (compare Subsection 6.23.1). The ensemble methods were applied to produce ensembles of size 29 and after the addition of each classifier the ensemble was tested. For objective functions f_3 and f_4 the number of classifiers in the final ensemble must be given. This number was always set to the maximal number of classifiers. Three key values of the ensembles were measured:

- **vote reject val**: The performance of the ensemble on the validation set using the *vote reject* combination scheme. This value can be measured during the design phase of the classifier ensemble. In addition this is the value which is optimized by the ensemble methods.
- **vote reject test**: The performance of the ensemble on the test set using the *vote reject* combination scheme. This value serves as a comparison to the value of *vote reject val* .
- **vote random**: The average performance of the ensemble on the test set using the *vote random* combination scheme. This value is averaged over 10 applications of the scheme. All the methods which are tested in this section try to optimize this value. There are two reasons why this and not another combination scheme is used. Firstly, the results of this scheme, when averaged over 10 combinations, are far more stable then other schemes. In addition the performance of the *vote random* combination scheme correlates strongly with the performance of all versions of the *vote best* combination scheme. Please note that the *vote best* scheme produced the best results for the considered ensemble methods.

In Figures A.32 to A.40 the results of the experiments are given. Based on these results two kinds of methods to set the number of classifiers were designed:

method	fix 9	fix 10	fix 11	fix 12	fix 13	dyn 5	dyn 10
inv. ranksum	35	32	38	42	35	42	46
vote random	70.38 %	70.32 %	70.4 %	70.41 %	70.35 %	70.51 %	70.53 %
class. num	9	10	11	12	13	11	14.2
rank	5/5	7/7	4/4	2/3	5/6	2/2	1/1

Table 6.26: Validation of the methods which set the number of classifiers for the feature search ensemble methods.

- Fixed number of classifiers (**fix** n): The ensemble always contains the same fixed number of n classifiers. This method was used in the experiments in Section 6.23.1 with $n = 10$.
- Dynamic number of classifiers (**dyn** n): The number of classifiers is determined using the value of *vote reject val*. The value of *vote reject val* for the classifier number i will be denoted as $f(i)$ in the following. For each number of classifiers i the value of $f(i)$ is compared to the values obtained for classifier numbers in its "neighborhood". The smallest number $x > n$ is chosen as the final number of classifiers for which one of the following conditions holds:
  - x is even, $f(x+1) \leq f(x)$ and $f(x-1) \leq f(x)$
  - x is odd, $f(x+2) \leq f(x)$, $f(x-2) \leq f_1(x)$ and $f(x+1) \leq f(x)$[7]

  The reason why the parity is considered is that odd numbers of classifiers lead to a lower number of ties than even numbers and therefore the value of f tends to be higher for them. For this method a maximal number of classifiers should also be given.

For the results of Figures A.32 to A.40 the methods *fix* n with $9 \leq n \leq 13$, *dyn 5* and *dyn 10* seem to be promising. The maximal classifier number for the *dyn* method was set to 29. To validate the quality of the methods their results were ranked for each ensemble method and each objective function. The number $m - i + 1$ is assigned to the rank i where m is the total number of methods which are validated[8]. For each method the sum of those numbers is calculated. This sum will be denoted as the inverted rank sum. The higher the inverted rank sum, the lower the ranks of the results of the method and the better the method performed. The inverted rank sums of all seven considered methods are shown in Table 6.27. The row *vote random* in this table contains the performance of the ensemble combined by *vote random* on the test set, averaged over all ensemble methods and objective functions. The entries in row *class. num.* show the average number of classifiers used. Finally the row *rank* show the overall ranking of the methods where the first value considers the inverted ranksum and the second value the average performance.

The method *dyn 10* was best for both the inverted rank sum and the average performance. This method will therefore be used in the experiments in Chapter 7. The maximal number

[7]If these four conditions hold true then also $f(x-1) \leq f(x)$ is true, because if $f(x) < f(x-1)$ then $x-1$ would already have been chosen as final number.

[8]This is similar to the assignment of numbers to the ranks used in Borda count (compare Subsection 5.2.3).

of classifiers will be set to 23 to avoid ensembles which are too large. (The largest number of classifiers obtained by method *dyn 10* was 23).

6.26.2 Feature selection ensemble methods using exhaustive search

In this subsection methods to determine the optimal number of classifiers for the ensemble methods using exhaustive feature search introduced in Subsections 4.5.3 are evaluated.
All but one of the methods tested in Subsection 6.23.2 were examined and the same notation as in this previous subsection will be used. The *replace exhaustive* method using objective function f_4 was not tested because the time complexity of this method was very high. The methods were applied to produce ensembles of size 29 in the case of *std. exhaustive* and of size 14 in the case of *replace exhaustive*. The maximal number of tested classifiers for *replace exhaustive* was set to such a low value because this method has a high time complexity, which increases more than quadratically with the number of classifiers. After the addition of each classifier the ensemble was tested and the same key values of the ensembles as in the last subsection were measured.
In Figures A.41 to A.47 the results of the experiments are given. One first important observation from the results is that the recognition rate on the validation set usually increases, especially for the *replace exhaustive* method. The reason for this is the overfitting of the methods on the validation set. Based on the results three kinds of methods to set the number of classifiers were designed:

- Fixed number of classifiers (**fix** n): The same method as in the last section.
- Dynamic number of classifiers (**dyn'** n): Similar to the method *dyn* in the last section, yet the condition for x to be the final number of classifiers is different: The smallest number $x > n$ is chosen as the final number of classifiers for which the following condition holds:
  - $f(n+1) \leq f(n)$ and $f(n-1) \leq f(n)$

  In contrast to the *dyn* method of the last section the parity of n is not taken into account. The reason for this is that because of the tendency of the two methods *std. exhaustive* and *replace exhaustive* to overfit, f is almost always increasing on the validation set and the *dyn* method of the last subsection would in many cases return the maximum number of classifiers. Similar to the method *dyn*, for *dyn'* a maximal number of classifiers should also be given. For the experiments in this subsection the maximal number was the same as the maximal tested classifier number.
- Even number with high increase (**even** n): It was observed that a strong increase in the performance for an even number of classifiers on the validation set often corresponds to an optimum of the performance on the test set. Based on this observation the smallest number $x > n$ is chosen as the final number of classifiers for which the following condition holds:
  - x is even, $(f(x) - f(x-1)) \geq 2 \cdot (f(x+1) - f(x))$ and $(f(x) - f(x-1)) > 0$.

method	fix 10	fix 11	fix 12	fix 13	fix 14	fix 15	fix 16	fix 17
inv. ranksum	18	23	33	28	34	29	27	41
vote random	69.42	69.64	69.74	69.72	69.77	69.9	70.06	70.34
class. num	10	11	12	13	14	15	16	17
rank	14/14	12/12	6/10	10/11	5/8	9/7	11/6	1/1

method	fix 18	fix 19	fix 20	dyn > 9	dyn > 14	even
inv. ranksum	32	40	39	23	39	33
vote random	70.23	70.32	70.17	69.61	70.31	69.76
class. num	18	19	20	13.25	17.25	12.5
rank	8/4	2/2	3/5	12/13	3/3	6/9

Table 6.27: Validation of the methods which set the number of classifiers for *std. exhaustive*. The values in the row *vote random* are given in percentages.

The number x is used as the final number of classifiers if the increase for x is more or equal than the double of the increase for $x + 1$. The condition $(f(x) - f(x - 1)) > 0$ guarantees that there is an increase for x, and not a decrease.

It is reasonable to assume that for the two ensemble methods *std. exhaustive* and *replace exhaustive* different methods will be good at determining the optimal number of classifiers, so both methods were addressed separately.
For *std. exhaustive* the methods *fix n* with $10 \leq n \leq 20$, *dyn' 9*, *dyn' 14* and *even 9* seem to be promising. Those methods are compared in Table 6.27 where the same notation as in Table 6.26 is used.
The method *fix 17* was the best. The *dyn' 14* method achieved the third rank for both the inverted rank sum and the average performance. Furthermore the difference of the *dyn' 14* method to the best method was only 2 and 0.03 %, respectively.
In the experiments in Chapter 7 the *dyn' 14* method will be used. The reason why this method is used instead of *fix 17* is the following: In the *dyn'* method the information of the test on the validation set is used. In the experiments in this section the validation set is rather small and the values measured on it are therefore quite unreliable. In the experiments in Chapter 7 a partial cross-validation scheme will be applied which should lead to more robust and reliable results on the validation set. The maximal number of classifiers for *dyn' 14* will be set to 29 to avoid ensembles which are too large.
For *std. exhaustive* the methods *fix n* with $5 \leq n \leq 14$, *dyn' 4* and *dyn' 5* seem to be promising. Those methods are compared in Table 6.28, where the same notation as in Table 6.26 is used.
The method *dyn' 5* was best for both the inverted rank sum and the average performance. This method will therefore be used in the experiments in Chapter 7. The maximal number of classifiers will be set to 14 to avoid ensembles which are too large.
It should be noted that the number of tested methods, four for *std. exhaustive* and three for *replace exhaustive*, is rather small and therefore the superiority of the discovered methods which set the number of classifiers is uncertain.

method	fix 5	fix 6	fix 7	fix 8	fix 9	fix 10	fix 11
inv. ranksum	21	24	23	22	22	20	18
vote random	68.6	68.74	68.73	68.74	68.71	68.52	68.42
class. num	5	6	7	8	9	10	11
rank	6/6	2/2	3/4	4/3	4/5	8/8	9/10

method	fix 12	fix 13	fix 14	dyn > 4	dyn > 5
inv. ranksum	13	14	15	21	30
vote random	68.39	68.46	68.32	68.6	68.91
class. num.	12	13	14	5	10
rank	12/11	11/9	10/12	6/6	1/1

Table 6.28: Validation of the methods which set the number of classifiers for *replace exhaustive*. The values in the row *vote random* are given in percentages.

Chapter 7

Experiments using optimized classifiers

In this chapter the results of the experiments using setup (6) in Table 6.1 are presented. In Sections 7.1 and 7.2 the optimization of the two base classifiers are described. In Section 7.3 experiments with the feature selection algorithms are presented. The next section contains results of the classic ensemble methods. In Section 7.5 the results of the ensemble methods using partitioning of the training set are given. The new boosting algorithms and the feature selection ensemble methods are evaluated in Section 7.6 and 7.7, respectively. In the following section the results of some ensemble methods using several base classifiers are presented. Finally, the best results of this and the last chapter are summarized in Section 7.9.

7.1 Optimization of geometric classifier

In this section the optimization of the geometric features based classifier introduced in Section 2.2 is presented. This classifier will be denoted as *geometric* classifier in the following. In the first subsection the optimization of the number of states is shown, whereas in the second subsection the optimization of the training method is addressed.

7.1.1 State number optimization

In this subsection the results of the application of the methods described in Section 3.1 are shown. The experimental setup (6c) in Table 6.1 was used, where a full 10-fold cross validation is applied and for which very stable results may be expected. In Figure 7.1 the recognition rates measured by the validation procedure are displayed, where only values near to the optimum are shown. The recognition rate of the base system, which is the system using a fixed number of 14 states per HMM, was 64.87 %.
The quantile method achieved a recognition rate of 68.82% with parameter $q = 0.023$, while the recognition rate of the classifier, which was produced by the Bakis method with the best value of 0.36 for parameter f, was 68.62%. The recognition rate of the optimized system on the test set was 71.6 % for the quantile method and 70.89 % for the Bakis method. The base system

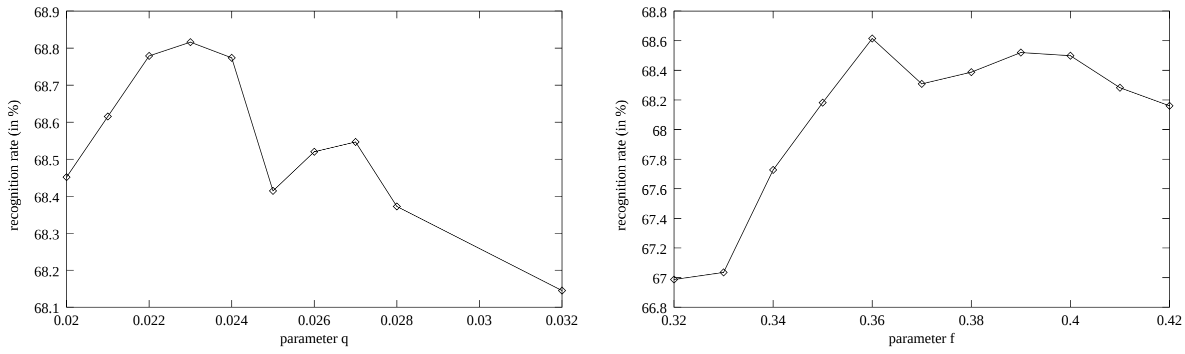


Figure 7.1: Recognition rates (in percentages) of the classifiers optimized by the quantile (left) and the Bakis (right) method with different values for parameters q and f. The recognition rate of the original classifier was 64.87 %.

had a performance of 67.8 % on the test set. The performance increase from the base classifier to the optimized system was similar for the validation procedure (3.95 / 3.75 %) and the test on the test set (3.8 / 3.09 %). This shows that unlike the experiments in Section 6.8 there is no significant overfitting effect.
As the two methods did not produce significantly different results, the optimization of the training method is done for both in the next subsection.

7.1.2 Optimization of the training method

In this section the optimization of the training method using the third strategy described in Section 3.2 is presented. The experimental setup (6b) in Table 6.1 was used. The two classifiers whose state numbers are optimized by the quantile and Bakis method were used as base classifiers. The first system will be denoted as *quantile* system, the second as *Bakis* system.
In Table 7.1 the recognition rates of the two systems with increasing number of Gaussians, and the number of training iterations between the increases in the number of Gaussians, are shown. The recognition rates of the systems are also depicted in Figure 7.2.
The quantile system is better than the Bakis system when using more than 6 Gaussians. In addition the behavior of the quantile system is sounder than that of the Bakis system, e.g. the increase in the recognition rate decreases monotonic for the quantile system, but not for the Bakis system. For these two reasons the system optimized by the quantile method will be used in the following for the *geometric* base classifier.
In the next experiments the system will be trained according to the found optimal training strategy, but only up to and including the 8th Gaussian. The reasons for not using the optimal system with 12 Gaussians are given as follows. Firstly, the performance of this 8-Gaussian system is only 0.5 % worse than that of the 12-Gaussian system. In addition it has a much lower training and recognition complexity, because only 111 instead of 177 training iterations must be done, and the likelihood of only 8 instead of 12 Gaussians must be calculated. The most important reason for using the 8-Gaussian system is that the validation data was written by the same writers as the training data, which could lead to overfitting. (The test data was

	number of Gaussians												
system	1	2	3	4	5	6	7	8	9	10	11	12	13
quantile	4	21	18	17	18	21	6	6	16	15	18	17	5
Bakis	4	24	22	12	6	15	5	8	10	6	6	-	-

	number of Gaussians						
system	1	2	3	4	5	6	7
quantile	68.9 %	72.93 %	75.13 %	77.37 %	78.47 %	79.23 %	79.77 %
Bakis	68.53 %	74.27 %	76.97 %	78.27 %	79.23 %	79.33	79.5 %
	number of Gaussians						
system	8	9	10	11	12	13	14
quantile	**80.13** %	80.3 %	80.43 %	80.57 %	80.63 %	80.63 %	-
Bakis	79.7 %	80.27 %	80.3 %	79.8 %	-	-	-

Table 7.1: Recognition rates (below) and the corresponding optimal training iterations (above) for different number of Gaussians of the systems optimized by the quantile and Bakis method

produced by different writers to the training data). Such overfitting normally happens when there are a lot of free parameters, i.e. in the actual case a high number of Gaussians. Therefore it is reasonable to choose a rather low number of Gaussians.
The performance of the final system on the test set is 80.36 %. The training of the system was done in two ways: In the first approach one computer and in the second several computers were used. In the second approach the training set is split into several parts, and the results of these parts are fused together at the end of each training iteration. Due to the different training processes, slightly different systems were produced. The results stated in this subsection were produced by the second training approach. The system created by the first training approach has a recognition rate of 80.03% for the validation procedure and 80.48% on the test set. In the following it will not be stated which training approach was used, as this will already be indicated by the performance of the base classifier.

7.2 Optimization of sub-sampling classifier

In this section the optimization of the sub-sampling features based classifier introduced in Section 2.3 is presented. This classifier will be denoted as *sub-sampling* classifier in the following.
In [93] the features of the sup-sampling classifier were transformed by the PCA (compare Section 3.4) and so also in the experiments of this thesis the PCA was used. There are three meta-parameters of the classifier which should be optimized: The state numbers, the training method, and the size of the transformed feature vector. From the experiments in Section 6.9 we conclude that the optimization of the state numbers is not strongly dependent on the training method. Furthermore we assume that the optimization of the state numbers is also not strongly dependent on the size of the transformed feature vector. So the optimization of the state numbers was done first and its results are presented in Subsection 7.2.1. Unfortunately, it is not

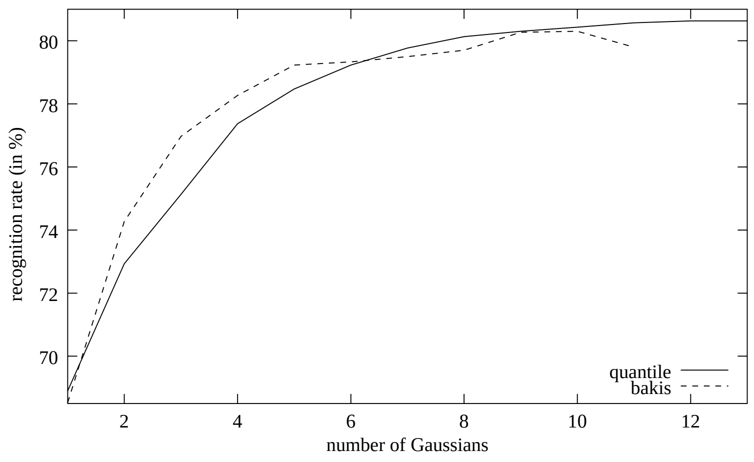


Figure 7.2: Recognition rates of the systems optimized by the quantile and Bakis method and trained by the optimal training method for different number of Gaussians

reasonable to expect that the best feature vector size is not strongly dependent on the number of Gaussians. Yet the number of Gaussians is determined by the training method, i.e. the feature vector size may be strongly dependent on the training method. The following approach to optimize both training method and feature vector size was used:

1. The approach starts with the classifier C with optimized state numbers (compare Subsection 7.2.1), single Gaussian distributions and the full PCA transformed feature vector.

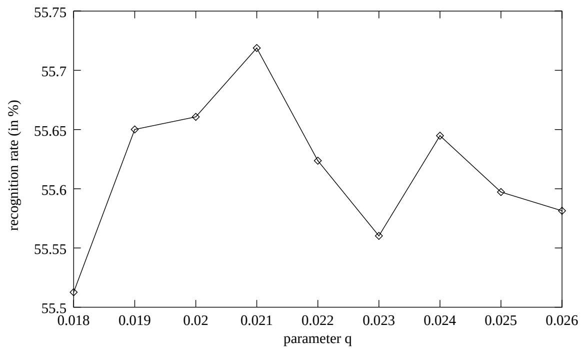
2. For varying sizes of the feature vector the system C is trained in the following manner: After each increase of the number of Gaussians a fixed number of c training iterations are done, where c is the optimal number of training iterations for the base system. The best found systems of all feature vector sizes are compared and the feature vector size which produced the best system is used in the following. This part of the approach is shown in Subsection 7.2.2.

3. The third strategy to find the optimal training method introduced in Section 3.2 was applied with the system C using best feature vector size found in the previous step. This part of the approach is shown in Subsection 7.2.3.

The idea of the above described approach was to produce a well, but suboptimal, trained systems first, to select the best feature vector size and then finally to find the optimal training method.

7.2.1 State number optimization

In this subsection the results of the application of the methods described in Section 3.1. The experimental setup (6c) in Table 6.1 was used where a full 10-fold cross validation is applied and for which very stable results may be expected. The base classifier with single Gaussian distributions was trained with 13 training iterations. (The number of 13 training iterations was



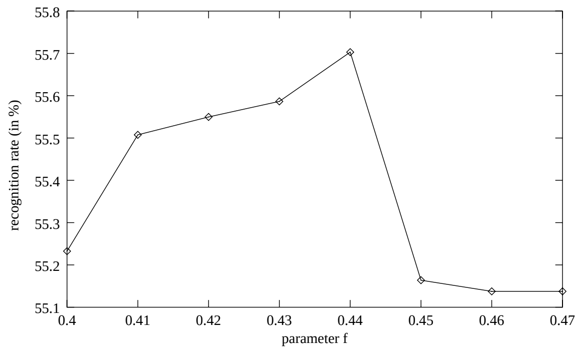


Figure 7.3: Recognition rates of the classifiers optimized by the quantile (left) and the Bakis (right) method with different values for parameters q and f on the validation set. The recognition rate of the original classifier was 51.57 %.

found to be optimal for this base classifier). A PCA transformation was applied to the feature vectors where the size of the feature vectors was not reduced.
In Figure 7.3 the recognition rates measured by the validation procedure are displayed where only values near to the optimum are shown. The recognition rate of the base classifier which used a fixed number of 14 states per HMM was 51.57 %.
The quantile method achieved a recognition rate of 55.72% with parameter $q = 0.021$, while the recognition rate of the classifier produced by the Bakis method with the best value of 0.44 for parameter f was 55.7%. As the quantile method again produced better results, the system optimized by the Bakis method will not be considered in the following. The recognition rate of the system optimized by the quantile method on the test set was 53.55 %, while the base system achieved only a recognition rate of 49.97 %. The performance increase from the base classifier to the optimized system was similar for the validation procedure (4.15 %) and test on the test set (3.58 %). This means that there is no significant overfitting effect.

7.2.2 Optimization of the feature vector size

In this subsection the determination of the optimal feature vector size is shown. The experimental setup (6b) in Table 6.1 was used. For different feature vector sizes systems with different number of Gaussians per distribution were trained. The training method was already described in the first part of the section. All number of Gaussians up to 6 were tested and the number of Gaussians was then increased until a maximum of the performance was found. The results of the experiments are shown in Figure 7.4.
The best system which achieved a recognition rate of 72.2 % used transformed feature vector of size 14, so this size will be used in the following experiments.

7.2.3 Optimization of the training method

In this section the optimization of training method using the third strategy described in Section 3.2 is presented. The experimental setup (6b) in Table 6.1 was used. The classifier to which

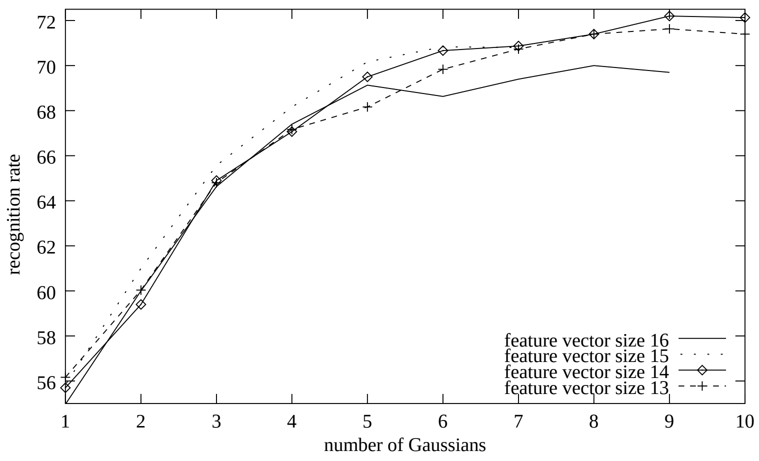


Figure 7.4: Recognition rates of the systems using different feature vector sizes

number of Gaussians											
1	2	3	4	5	6	7	8	9	10	11	12
21	18	20	23	9	15	14	14	19	19	6	19

number of Gaussians											
1	2	3	4	5	6	7	8	9	10	11	12
56.3	60.3	65.17	66.97	69.33	70.17	71.07	71.97	**72.6**	72.9	73.3	73.03

Table 7.2: Recognition rates (below) in percentages and the corresponding optimal training iterations (above) for different number of Gaussians of the optimized system

the strategy is applied has state numbers optimized by the quantile method and works with PCA transformed feature vectors of size 14 (compare Subsection 7.2.1 and 7.2.2).
In Table 7.2 the recognition rates of the classifier with increasing number of Gaussians and the number of training iterations between the increases of the number of Gaussians are shown. The recognition rates of the system are also depicted in Figure 7.5.
In future experiments the system will be trained according to the resultant optimal training strategy, but only up to and including the 9th Gaussian. The reasons for not using the system with all 11 Gaussians was already stated in Subsection 7.1.2. The number of 9 Gaussians was chosen so that there is a similar performance difference between the optimal system and the system used in the experiments for both the *geometric* and the *sub-sampling* classifier.
The performance on the test set of the system using 9 Gaussians depends on the the training approach used (compare last paragraph of Subsection 7.1.2). When using only one computer, a recognition rate of 71.57 % was obtained. In the case of the use of several computers, a recognition rate of 70.22 % was achieved[1]. In the following it will not be stated which training

[1]The large performance difference is not mainly due to the different training approaches. Some small scale experiments indicated that the performance difference may have been caused by a different order of the training files. See Subsection 8.2.14 for more information.

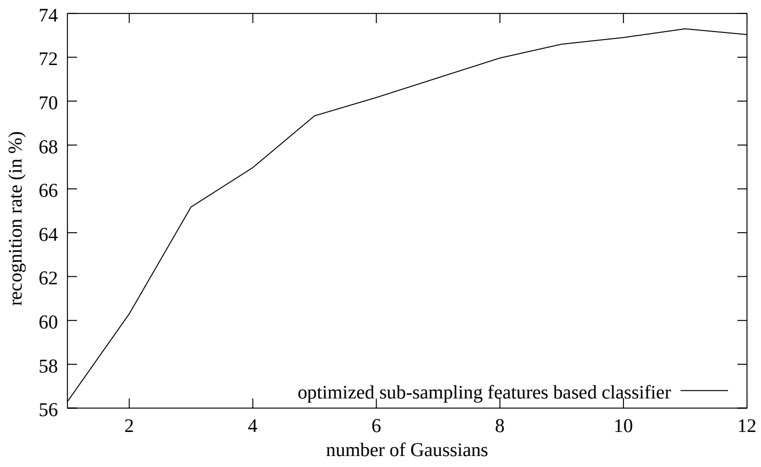


Figure 7.5: Recognition rates of the optimized system for different number of Gaussians

method	parameter	valid	test	time
validation	backward search	80.03 (79.27) %	80.48 (78.74) %	9 (17)
validation	forward search	77.57 %	78 %	31
validation	exhaustive	80.03 (79.27) %	80.48 (78.74) %	511
parameters	fisher	80.03 (78.57) %	80.48 (78.55) %	9
parameters	local	80.03 (78.17) %	80.48 (76.65) %	9
parameters	root local	80.03 (78.4) %	80.48 (79.29) %	9
simulation	best, $N = 4$	78 %	76.07 %	1.85
simulation	best, $N = 5$	78 %	76.07 %	1.9
simulation	validation, $N = 4$	80.03 (79.27) %	80.48 (78.74) %	10.85
simulation	validation, $N = 5$	80.03 (79.27) %	80.48 (78.74) %	10.9
original	all 9 features	80.03 %	80.48 %	-

Table 7.3: Results of the feature selection methods

approach was used, as this will already be indicated by the performance of the base classifier.

7.3 Feature selection for one classifier

In this section the methods which were tested in Section 6.6 are applied using the experimental setup (6a) in Table 6.1. Feature selection algorithms are not suitable for the *sub-sampling* base classifier, as this classifier uses PCA transformed features. So only the *geometric* base classifier was tested. The results are shown in Table 7.3 where the same notation as in Table 6.3 is used. The classifier using all 9 features achieved a recognition rate of 80.03 % on the validation set and 80.48 % on the test set. Most algorithms output the full set of features. The results of these algorithms in the case where the output of the full set is prohibited is given in brackets.
Unlike the experiments presented in Section 6.6 the best feature set for the validation set is the

size	p. fisher	p. local	p. local r.	simulation $N = 4/5$	average	best
1	16.07 %	16.07 %	16.07 %	40.53 %	25.93 %	40.53 %
2	50.5 %	28.03 %	45.87 %	65.9 %	45.34 %	67.07 %
3	48.7 %	48.7 %	62.3 %	77.57 %	55.73 %	77.57 %
4	58.7 %	66.2 %	73.67 %	76.2 / 76.57 %	60.2 %	76.57 %
5	72.6 %	72.23 %	75.43 %	65.77 %	66.85 %	76.6 %
6	76.9 %	73 %	76.43 %	74.83 %	72.17 %	78.3 %
7	78.2 %	75.43 %	77.23 %	78 %	76.07 %	78.7 %
8	78.57 %	78.17 %	78.4 %	79.27 %	78.03 %	79.27 %
9	80.48 %					

Table 7.4: Performance of the classifiers using the best found feature sets measured on the validation set

full set of features[2]. There are two reasons why the full feature set is best for the base classifier considered in this section:

1. By using several Gaussians per distribution, the system can more effectively model the "real" distribution of the features. When using single Gaussian distributions often the difference of the modeled distribution to the "real" distribution was so large that the inclusion of a feature lowered the performance.

2. The training method was optimized for the system using the full feature set. It can be expected that the optimal training method is different for systems using other feature sets. The proper approach would be to optimize the training method for each feature set separately, yet the time complexity of this approach is very high.

The fact that the best feature set is the full set limits the performance increases obtainable by the ensemble methods which produce classifiers using different feature sets (compare Section 7.7).
All methods except the *forward search* and the *simulation best* algorithms output the best feature set, i.e. the full set. The *forward search* got stuck in a local minimum and the *simulation best* algorithm again overestimated the small feature sets. Because of this overestimation the *simulation best* algorithm should not be used when it is expected that the best feature set could be large.
The results in Table 7.3 contain only little information of the quality of the performance estimation by the *parameters* and the *simulation* methods. In Table 7.4 the performance on the validation set of the systems using the best found feature sets for each feature set size is shown. In this table "parameters" is abbreviated to "p.". The average performance of the classifiers using the feature sets of the given size is shown in column *average*. The entry in column *best* contains the performance of the best feature set of the given size. The *simulation* method produced, in all but one case, the same result for $N = 4$ and $N = 5$.
The *p. local* method is clearly inferior, as the *p. local r.* method outperformed it for every feature vector size. The *p. local r.* method produced the best result of the *parameters* methods

[2]This is also true for the test set.

ensemble	base	combination	cn	performance	sig
Bagging	geometric	*perf. weight*	21	80.91 %	10 %
AdaBoost	geometric	*perf. weight*	14	81.89 %	0.1 %
rsm	geometric	*ga weight*	25	80.61 %	30 %
Bagging	sub-sampling	*perf. weight*	21	73.04 %	0.2 %
AdaBoost	sub-sampling	*perf. weight*	14	73.9 %	< 0.01 %

Table 7.5: Results of the classic ensemble methods. The performance of the *geometric* base classifiers is 80.36 %, while the *sub-sampling* base classifier had a performance of 71.57 %.

when considering all feature vector sizes. In 7 out of 8 cases the result of the *p. local r.* method was better than the average. The superiority of the *p. local r.* method over the average is therefore significant using a significance level of 4 %. For large feature vector sizes the *p. fisher* method is slightly better than the *p. local. r.* method. In general the *parameters* methods are good for large feature vector sizes. When taking only sizes larger than 4 into account always better results than the average were obtained for all three *parameters* methods.
The *simulation* method obtained in all but one case recognition rates better than the average. Additionally, recognition rates close to that of the best feature set were achieved in 6 out of 8 cases. The *simulation* method is much better than the *parameters* methods for small feature vector sizes, but is comparable to them for large sizes.
In summary the *simulation* method is suitable for any feature vector size, while the *p. fisher* method may produce better results for large sizes.

7.4 Classic ensemble methods

In this Section the classic ensemble methods introduced in Section 4.2 are evaluated using experimental setup (6a) in Table 6.1. The ensemble methods are only used with the combination schemes and number of classifiers found to be optimal in the last chapter.
Table 7.5 summarizes the results of the classic ensemble methods. The names of the ensemble methods used in the experiments are shown in column *ensemble* where the random subspace method is abbreviated to *rsm*. The entries in column *base* denote the used base classifiers. The performance of the *geometric* and the *sub-sampling* base classifiers are 80.36 % and 71.57 %, respectively. The used combination schemes are shown in column *combination* and the number of classifiers are indicated in column *cn*. An advanced comparison test (compare Subsection 6.3.2) was executed and the superiority of of the results of the ensemble method over the results of the base classifier was evaluated. The significance level of the superiority is given in column *sig*. Please note that the random subspace method was only executed for the *geometric* classifier, because the *sub-sampling* base classifier uses PCA transformed features and PCA and feature selection do not work well together.
All ensemble methods increased the performance of the corresponding base classifier. There is only a slight increase of the performance for random subspace method (and the increase is not statistically significant). The reason why random subspace method did not produce very good results is that the best feature set is the full set of features and that classifiers using other

		feature sets		
cn	combination	{CPW}	{WPC}	{WPC,WC}
2	score	82.23 %	81.7 %	81.73 %
2	special	82.8 %	82.4 %	82 %
3	score	82.23 %	82.6 %	82.63 %
3	special	**83** %	82.56 %	**83.17** %
3	vote best	81.16 %	82.03 %	81.83 %
4	score	82.03 %	82.63 %	82.1 %
4	special	82.93 %	**82.87** %	82.87 %
4	vote best ave	81.45 %	81.73 %	79.97 %
4	vote best max	80.53 %	81.63 %	80.06 %
4	vote best min	80.5 %	81.63 %	79.67 %
4	vote best med	80.5 %	81.77 %	79.67 %
5	score	81.73 %	81.8 %	82.33 %
5	special	82.53 %	82.43 %	82.67 %
5	vote best ave	79.73 %	80.5 %	80.57 %
5	vote best max	79.67 %	80.6 %	80.6 %
5	vote best min	79.67 %	80.47 %	80.47 %
5	vote best med	79.67 %	80.47 %	80.47%

Table 7.6: Results of the ensembles whose classifiers are trained on the clusters produced by k-means clustering with the indicated features. The performance of the base classifier is 80.03 %.

feature sets have much lower recognition rates (compare Section 7.3). The performance increase obtained by Bagging is considerable (and statistical significant using a significance level of 10 % and 0.2 %, respectively). The highest recognition rates were achieved by AdaBoost. The performance of the base classifier was increased by 1.53 % and 2.33 %, respectively.
From the results it may be concluded that AdaBoost is the most promising ensemble method as the best performances were obtained and the smallest ensembles were used.

7.5 Ensemble method using partitioning of training set

In this section results of the ensemble method using partitioning of the training set described in Section 4.3 are studied. Only the *geometric* classifier was used for the experiments. First, the ensemble method was applied using experimental setup (6b) in Table 6.1 to select some good parameters. The results are in displayed in Table 7.6. The notation is the same as in Table 6.14 shown in Section 6.21. The best results of each feature set is marked in bold.
The results of the *special* combination scheme are clearly superior. In 11 out of 12 cases the best recognition rate was obtained using this scheme. Both the *special* and the *score* combination schemes were consistently superior than the *vote best* scheme. In the following only the combination with the *special* scheme will be considered. Using this scheme the performance of the

	classifier number			
combination	2	3	4	5
score	79.53 %	79.66 %	79.92 %	79.45 %
special	79.38 %	79.81 %	80.15 %	80.11 %
vote best ave	79.53 %	79.82 %	80.25 %	79.2 %
vote best max	79.53 %	79.82 %	80.3 %	79.19 %
vote best min	79.53 %	79.82 %	80.12 %	79.16 %
vote best med	79.53 %	79.82 %	80.12 %	79.16 %

Table 7.7: Results of the ensembles whose classifiers are trained on the random clusters of equal size. The performance of the base classifier is 80.48 %.

feature set	cn	performance	sig
{CPW}	3	80.67 %	37 %
{CPW}	4	80.76 %	32 %
{WPC}	3	80.48 %	50 %
{WPC}	4	80.73 %	34 %
{WPC,WC}	3	81.07 %	14 %
{WPC,WC}	4	80.64 %	30%

Table 7.8: Results of the ensembles whose classifiers are trained on the clusters produced by k-means clustering with the indicated features. The performance of the base classifier is 80.48 %.

base classifier was increased by 1.97 % to 3.13 %. It seems that all feature sets and classifier numbers are quite suited for the ensemble method using partitioning of the training set. The results for the classifier numbers 3 and 4 were best and so only these numbers are used to test the ensemble method with experimental setup (6a) in Table 6.1.
For comparison purposes the performance when using random clustering was also tested. The random clusters had all the same size. For each classifier number the ensemble method was repeated three times and the results are averaged over the three runs. The results are shown in Table 7.7 where the notation is the same as in Table 7.6. The results of the ensembles using any combination scheme were always worse than those of the base classifier.
Table 7.8 shows the results of the ensemble method with experimental setup (6a). The ensembles were always combined by the *special* scheme. The columns *feature* and *cn* denote the used feature set and the number of classifiers, respectively. An advanced comparison test was executed for each result and the superiority of the result over the the performance of the base classifier was evaluated. The significance level of the superiority is given in column *sig*.
The performance of the ensemble method was always better or the same as the performance of the base classifier. If using the sign test to compare the results of the ensemble method to the results of the base classifier, we conclude that the ensemble method is superior using a significance level of 4 %. The obtained increases in the recognition rate are very low when compared to the results in Table 7.6. Please note that for the best configuration in Table 7.6, cluster number 3 and feature set {WPC,WC}, also the best result in Table 7.8 was achieved. If

algorithm	error/α	vote best max	perf. weight	ga weight
Bagging	-	81.1 %	80.91 %	81.13 %
AdaBoost	-	82.02 %	81.89 %	81.92 %
EBB	1.5	79.2 %	79.01 %	78.89 %
SPBE	e_1	78.77 %	78.74 %	79.38 %
SPBE	e_3	82.69 %	82.41 %	82.51 %
SPBE	e_4	82.54 %	82.6 %	81.8 %

Table 7.9: Results of the experiments with the new boosting algorithms. The recognition rate of the base classifier is 80.36 %.

the validation procedure would have been only used to find the single best configuration, and only this configuration would have been tested, a performance increase of 0.59 % in respect of the base classifier would have been achieved.
Please note that the classifiers which are trained on the clusters are using suboptimal parameter values, because those parameters were optimized for the whole training set. Such parameters are for example the number of states and the training method. Similarly, the classifiers using a subset of the feature set were trained with unoptimized parameters in the experiments in Subsection 7.3. The considered ensemble method is expected to produce better results when using classifiers with optimized parameter values.
It seems that the considered ensemble method is very good for writer dependent setups, but only achieves small performance increases for writer independent setups.

7.6 New boosting algorithms

In this section the new boosting algorithms introduced in Section 4.4 are evaluated. The setup (6a) in Table 6.1 was tested and only the *geometric* classifier was used for the experiments. The number of classifiers produced by the ensemble methods was set to 14. The results are shown in Table 7.9 where the same notation as in Table 7.9 in Section 6.22 is used. To compare the new methods to classic ensemble methods, also results of Bagging and AdaBoost are shown. Here also other combination schemes than the best found schemes of the experiments of Chapter 6 were used to allow a better comparison of the ensemble methods. The results of the combination of the classifiers by the *vote best max*, *perf. weight* and *ga weight* combination schemes are given. Please note that the *perf. weight* combination scheme found to be the best combination scheme for Bagging and AdaBoost in the experiments of Chapter 6 is slightly inferior to the two other schemes.
The recognition rate of EBB and SPBE with error function e_1 are very low. The reason for the low performance is that the classifiers produced by these ensemble methods were trained on training sets which consisted of copies of only very few original training patterns. Thus very specialized classifiers were trained. Such classifiers are best combined by the *score* combination scheme. In fact the *score* combination scheme obtained recognition rates of 80.94 % and 80.7 % for EBB and SPBE with error function e_1, respectively. Both recognition rates are much better

than the recognition rates of the other schemes and also better than the performance of the base classifier. A way to solve the problem of training the classifiers on copies of too few original training elements is described in Subsection 8.2.8.
The results of SPBE with error functions e_2 and e_3 are quite good. For all combination schemes the recognition rates were much better than those of Bagging and in 5 out of 6 cases better results than AdaBoost were obtained. The superiority of SPBE with error function e_4 over AdaBoost was evaluated for both *vote best max* and *perf. weight* combination schemes. The significance level of the superiority was 9 % and 4 %, respectively. Due to time constraints the results of SPBE with error function e_3 were not compared to the results of AdaBoost. In addition the *sub-sampling* base classifier was not used for testing the new boosting methods.
The SPBE algorithm outperformed AdaBoost for some combination schemes and error functions. On the other hand, for other parameter values, the SPBE algorithm did not produce satisfying results. Therefore it may be suitable to find the best parameter values of SPBE on a validation set, before applying the ensemble method.

7.7 Feature selection ensemble methods

In this section experiments of the feature selection ensemble methods introduced in Section 4.5 are presented. Always the experimental setup (6b) in Table 6.1 was used. The experiments were done only with the *geometric* base classifier as feature selection does not work well with the *sub-sampling* base classifier.
In Subsection 7.7.1 results of the ensemble methods using underlying feature search algorithms are studied. In Subsection 7.7.2 ensemble methods which use the results of all feature sets are tested. Finally ensemble methods which simulate the results of the patterns of the validation set are studied in Subsection 7.7.3. The notation used in this Section and abbreviations are the same as in Section 6.23.

7.7.1 Feature selection ensemble methods using feature search

In this Subsection the methods described in Subsections 4.5.1 and 4.5.2 are evaluated. The number of classifiers is dynamically selected using the method *dyn 10* introduced in Subsection 6.26.1. The results of the experiments are shown in Table 7.10. The recognition rate of the base classifier which uses the full set of features was 80.48 % on the test set and 80.03 % on the validation set. The produced ensembles were combined by the *vote best median* scheme. The entries in column f indicate the objective functions used for the feature search ensemble methods. The recognition rates on the validation set of the ensembles combined by the *vote reject* scheme are given in column *valid*. The column *test* shows the performance of the ensembles on the test set. The results on the test set of the ensemble methods set were compared to the results of the base classifier. The significance levels of the superiority of the ensemble methods are given in column *sig*. Finally *sets* denotes the number of feature sets which were validated. For comparison purposes also the results of the random subspace method are given. Please note that the recognition rate given here differs from the one in Table 7.5 as another combination scheme is used.

algorithm	f	cn	valid	test	sig	sets
b. ensemble	f_2	15	82.97 %	82.32 %	< 0.01 %	128
f. ensemble	f_2	17	81.4 %	81.8 %	1 %	212
b./f. ensemble	f_2	13	83 %	82.69 %	< 0.01 %	224
b. ensemble	f_3	21	83.1 %	82.69 %	< 0.01 %	173
f. ensemble	f_3	21	82.13 %	82.42 %	0.02 %	220
b./f. ensemble	f_3	14	83.1 %	82.51 %	< 0.01 %	163
b. ensemble	f_4	19	83 %	82.35 %	< 0.01 %	162
f. ensemble	f_4	14	81.83 %	82.51 %	< 0.01 %	199
b./f. ensemble	f_4	16	83.27 %	82.51 %	< 0.01 %	191
rsm	-	25	-	80.76 %	20 %	0

Table 7.10: Results of the feature search ensemble methods. The recognition rate of the base classifier was 80.48 % on the test set and 80.03 % on the validation set.

All ensemble methods produced very good results. The recognition rates of the methods were 1.04 % up to 1.93 % higher than that of the random subspace. In addition the methods are stable as the recognition rates are in a small range of 0.89 %. The correlation of the performance on the validation set and the performance on the test set is 0.672. This is much higher than the correlation of 0.388 measured for the experiments of Subsection 6.23.1. The higher correlation observed in the experiments in this subsection is due to the improved validation procedure. However, the results of the forward search are still slightly underestimated.
There is no significant difference between the results of the different objective functions, yet the results of the objective function f_2 seem to be slightly worse than those of the other two functions. The *b./f. ensemble* method was slightly better than the other methods in respect of both ensemble size and ensemble performance. For f_2 and f_3 the *b./f. ensemble* method produced the smallest and for f_4 the second smallest ensemble. Likewise, the performance of the *b./f. ensemble* method was best for f_2 and f_4 (together with the *f. ensemble* method) and second best for f_3. So the results indicate that using two search algorithms leads in general to better results than using only one search algorithm.

7.7.2 Feature selection ensemble methods using exhaustive search

In this subsection the ensemble methods introduced in Subsection 4.5.3 are evaluated. The number of classifiers are dynamically selected using the methods *dyn' 14* and *dyn' 5* introduced in Subsection 6.26.2. The recognition rate of the base classifier which uses the full set of features was 80.48 % on the test set and 80.03 % on the validation set. The produced ensembles were combined by the *vote best max* scheme.
The results of the ensemble methods are shown in Table 7.11. The meaning of the entries in the table is the same as in Table 7.10. In an additional column *time* the time complexity of the algorithms are given. This time complexity was measured in multiples of the time needed for the validation of a feature set. Please note that for any of the algorithm all 511 feature sets must be validated first. For comparison purposes also the results of the random subspace method are given. Please note that the recognition rate given here differs from the one in Table 7.5 as

algorithm	f	cn	valid	test	sig	time
std. exhaustive	f_1	16	83.2 %	82.84 %	< 0.01 %	< 0.2
replace exhaustive	f_1	7	82.26 %	82.05 %	< 0.01 %	1.3
std. exhaustive	f_2	17	83.67 %	82.94 %	< 0.01 %	< 0.2
replace exhaustive	f_2	7	82.67 %	82.41 %	< 0.01 %	1.5
std. exhaustive	f_3	16	83.1 %	82.48 %	< 0.01 %	< 0.2
replace exhaustive	f_3	9	83 %	82.05 %	0.02 %	1.3
std. exhaustive	f_4	16	83.5 %	82.69 %	< 0.01 %	0.4
replace exhaustive	f_4	9	83.1 %	82.44 %	< 0.01 %	4
rsm	-	25	-	80.73 %	22 %	-

Table 7.11: Results of the ensemble methods using exhaustive feature search. The recognition rate of the base classifier was 80.48 % on the test set and 80.03 % on the validation set.

another combination scheme is used.

All ensemble methods produced very good results. The recognition rate of the methods were 1.32 % up to 2.21 % higher than that of the random subspace. In addition the methods are stable as the recognition rates are in a small range of 0.89 %. The correlation of the performance on the validation set and the performance on the test set is 0.798. This value is very high, especially as the correlation measured for the experiments of Subsection 6.23.2 was -0.082. This again shows that the improved validation procedure led to a higher correlation of the performances on the validation and the test set.

There is no significant difference between the results of the different objective functions, yet the objective function f_2 produced slightly better results than the other three functions. The *std. exhaustive* method produced large ensembles with very good recognition rates, while the *replace exhaustive* method produced much smaller ensembles which had lower recognition rates.

The feature selection ensemble methods using feature search produced on average ensembles of size 16.6 with an average recognition rate of 82.42 % (compare last subsection). The average performance of the *replace exhaustive* method of 82.24 % is slightly lower than this recognition rate, but its average ensemble size of 8 is less than a half of the average ensemble size of the methods tested in the last section. On the other hand the *std. exhaustive* method obtained an average recognition rate of 82.73% by using a similar number of classifiers than the feature selection ensemble methods using feature search. Please note that both the *replace exhaustive* and the *std. exhaustive* methods require the results of the validation set of all possible feature sets.

From the results of this and the last subsection the following strategy can be deduced: If the design time should not be too high, the feature selection ensemble methods using feature search should be used. If the design time is not important and the time complexity of the recognition should be small, then the *replace exhaustive* method should be applied. If both design time and time complexity of the recognition are not critical, then the *std. exhaustive* method should be applied.

method	cn	f	N	vote best max	vote best med
b. ensemble	12	f_2	4	81.53 (2) %	81.53 (2) %
b. ensemble	23	f_2	5	83 (<0.01) %	83.03 (<0.01) %
b. ensemble	11	f_4	4	80.98 (16) %	80.98 (16) %
b. ensemble	13	f_4	5	81.8 (0.5) %	81.83 (0.2) %
b./f. ensemble	16	f_2	4	82.2 (0.05) %	82.2 (0.05) %
b./f. ensemble	13	f_3	4	81.22 (7) %	81.22 (7) %
b./f. ensemble	15	f_4	4	81.83 (0.5) %	81.7 (0.5) %
rsm	25	-	-	80.73 (22) %	80.76 (20) %

Table 7.12: Results of the heuristic version of the feature search ensemble methods. The recognition rate of the base classifier was 80.48 %.

method	cn	f	N	vote best max	vote best med
std. exhaustive	15	f_1	2	81.07 (13) %	81 (15) %
std. exhaustive	15	f_1	3	81.4 (3) %	81.4 (3) %
std. exhaustive	15	f_1	5	82.2 (0.02) %	82.2 (0.02) %
std. exhaustive	15	f_3	2	81.56 (2) %	81.4 (3) %
std. exhaustive	15	f_3	3	82.08 (0.1) %	82.11 (0.1) %
std. exhaustive	15	f_3	5	81.89 (0.5) %	81.89 (0.5) %
replace exhaustive	7	f_1	5	81.37 (4) %	81.31 (5) %
replace exhaustive	7	f_3	5	80.79 (29) %	80.58 (44) %
rsm	25	-	-	80.73 (22) %	80.76 (20) %

Table 7.13: Results of the heuristic version of the ensemble methods using exhaustive feature search. The recognition rate of the base classifier was 80.48 %.

7.7.3 Feature selection ensemble methods using simulations

In this subsection the heuristic versions of the ensemble methods tested in the previous two subsections are evaluated. The used heuristic is described in Subsection 4.5.4. The ensemble methods are abbreviated by the same terms than in the two last subsections. The number of classifiers was determined by the same methods than in the non heuristic methods. The ensembles were combined by the *vote best max* and *vote best median* combination schemes, because those schemes were found to be best in previous experiments (compare Section 6.23.3). The results of the feature search ensemble methods are shown in Table 7.12 and the results of the feature selection ensemble methods using exhaustive search are given in Table 7.13. The names of the tested methods are shown in column *method*. The number of classifiers in the ensemble is given in *cn*. The entry in column f indicates the objective function used for the ensemble method. The number N is a parameter of the method that simulates the results of the validation set (compare Subsection 3.3.3). The two last columns show the performance of the ensembles using the indicated schemes on the test set. The results of the ensemble methods on the test set were compared to the results of the base classifier. The significance levels of the superiority of the ensemble methods over the base classifier are given in brackets.

The time complexities of the methods are the same as in the experiments in Section 6.23.
Both combination schemes performed very similarly. The *replace exhaustive* method had quite a low performance. The reason for this is that this method strongly overfits on the simulated results. As the *replace exhaustive* method also has a higher time complexity than the *std. exhaustive* method, it is not suitable for the considered application and only the results of the *std. exhaustive* method will be addressed in the following. The value 2 for parameter N seems not to be suitable for the *std. exhaustive* method. The *b./f. ensemble* method produced results of similar quality than the *b. ensemble* method, although two search algorithms are used. The reason for this is that the forward search is inferior to the backward search because the used heuristic overestimates small feature sets (compare Section 6.23.3).
There is no significant difference of the results of the feature search ensemble methods and the results of the *std. exhaustive* method. It seems that the benefit of the *std. exhaustive* method of considering all feature sets was set back by a stronger overfitting effect.
All methods (but the *replace exhaustive*) produced superior results than the random subspace method. This shows that the approximated results contain useful information for selecting feature sets. The results of the methods tested in this subsection seem to be instable. Very high recognition rates were obtained, including the overall best rate of all feature selection algorithms of 83.03 %, but also quite low ones. The heuristic versions of the feature selection ensemble methods did worse than the ensemble methods using the real results, of course. On average the performance was about 0.6 % lower for the feature selection algorithms and 1 % lower for the *std. exhaustive* method.
The heuristic version of the feature search ensemble methods have the following shortcoming in respect of the non-heuristic versions: Firstly, the results are very instable. Secondly, the performance is lower. On the other hand the heuristic versions have a far lower time complexity. The time complexity of the *std. exhaustive* and the feature search ensemble methods tested in this subsection was always lower than two times the complexity of validating a feature set. The lowest time complexity of a non-heuristic method was 64 times higher.

7.8 Ensemble methods using several base classifiers

In this section ensemble methods using several base classifiers are evaluated. The experiments are done with the setup (6a) in Table 6.1. The following two base classifiers were used:

- $\mathbf{C_g}$: The *geometric* classifier (compare Section 7.1). The performance of this classifier is 80.36 % on the test set.

- $\mathbf{C_s}$: The *sub-sampling* classifier (compare Section 7.2). The performance of this classifier is 70.22 % on the test set. For the ensemble methods only using this base classifier a different training approach was used for which the base classifier has a recognition rate of 71.57 %.

The classifiers of the ensembles were combined by the *vote best max*, *perf. weight* and *ga weight* schemes. Only the results of the maximum version of the *vote best* combination scheme is given because all versions had similar performances.

method	C	cn	vote best max	perf. weight	ga. weight
Bagging	C_g	21	81.1 %	80.91 %	81.13 %
Bagging	C_s	21	73.07 %	73.04 %	72.95 %
AdaBoost	C_g	14	82.02 %	81.89 %	81.92 %
AdaBoost	C_s	14	74.3 %	73.9 %	73.77 %
Bagging	C_g,C_s	20	83 %	83.64 %	83.21 %
AdaBoost	C_g,C_s	14	83.76 %	83.79 %	84.07 %
MSPB, e_2	C_g,C_s	14	83.79 %	84.16 %	84.16 %
MSPB, e_4	C_g,C_s	14	83.18 %	83.64 %	83.36 %

Table 7.14: Results of the ensemble methods using several base classifiers. The recognition rate of the base classifiers C_g and C_s is 80.36 % and 70.22 (71.57) %, respectively.

The results of the experiments are shown in Table 7.14. The tested ensemble methods are given in column *method* where MSPB is the abbreviations for multi simple probabilistic boosting which was introduced in Subsection 4.6.3. For this ensemble method also the used error function e is given. The entries in column C denote the base classifiers used for the experiments. If there is only one classifier then the classic ensemble method was applied. The results of these ensemble methods are included for comparison purposes. If the entry contains both base classifiers and the method is not MSPB then the algorithm introduced in Section 4.6.1 was applied to extend the classic ensemble method. The entries in column *cn* are the number of classifiers produced by the ensemble methods. If there are two base classifiers then $\frac{cn}{2}$ classifiers were generated from each base classifier. Please note that for Bagging C_g,C_s only 20 instead of 21 classifiers were used so that there are the same number of classifiers generated from each base classifier.
All ensemble methods improved the performance of the corresponding base classifier for all combination schemes. The ensemble methods using both base classifiers have better recognition rates than the ensemble methods which only use one base classifier. This is quite surprising as the base classifier C_s has an over 10 % lower performance than the base classifier C_g. It seems that the increase of diversity when using classifiers from base classifier C_s had a larger impact on the ensemble performance than the inferior individual performances of those classifiers. The same phenomenon was already observed in the experiments of Subsection 6.24.1.
When considering only the ensemble methods using both base classifiers the ordering of the methods according to their performance is as follows: Bagging, MSPB e_4, AdaBoost and MSPB e_2. This is true for all three combination schemes.
To analyze the results of the ensemble methods more thoroughly, the best results of the ensemble methods are compared to each other. For each pair of ensemble methods the significance of the superiority of the better ensemble method was calculated.
The results of this calculation are shown in Table 7.15. The entries in the table are the significance levels of the superiority. Only the results of the best ensemble method using one base classifier, AdaBoost using the *geometric* base classifier, was used for the comparison.
From the results of Table 7.15 it may be concluded that there is no significant difference between the performance of Bagging and MSPB e_4. Also there is no significant difference between the performance of AdaBoost and MSPB e_2. It seems that the MSPB algorithm did not produce

	AdaBoost C_g, C_s	Bagging C_g, C_S	MSPB, e_4	AdaBoost C_g
MSPB, e_2	42 %	10 %	9 %	$<$ 0.01 %
AdaBoost C_g, C_s	-	14 %	17 %	$<$ 0.01 %
Bagging C_g, C_S	-	-	0 %	0.05 %
MSPB, e_4	-	-	-	0.1 %

Table 7.15: Significance level of the superiority of the ensemble method referred to in the first column over the ensemble method referred to in the first row

superior results to AdaBoost as it was the case in the experiments in Section 6.25. A possible reason for this was already stated in Section 7.6: The classifiers produced by these ensemble methods were trained on training sets which consisted of copies of only very few original training patterns. This was especially the case for MSPB e_4. A way to solve the problem of training the classifiers on copies of too few original training elements is described in Subsection 8.2.8.

7.9 Summary of the best results

In this section the best results obtained in the experiments in this and the last chapter are summarized. The best results of the following five categories of ensemble methods are given:

1. Classic ensemble methods (compare Section 4.2).
2. Ensemble methods using partitioning of training set (compare Section 4.3).
3. New boosting algorithms (compare Section 4.4).
4. Feature selection ensemble methods (compare Section 4.5).
5. Ensemble methods using several base classifiers (compare Section 4.6).

The best results of experimental setups (3a) and (6a) are shown in Tables 7.16 and 7.17, respectively. The first column of these tables shows the categories of the best ensemble methods. The names of these ensemble methods are given in the second column. For category two not the name of the ensemble method, but the used parameter values for the clustering are given. The used number of classifiers and combination schemes are shown in the third and fourth columns. Finally, the column *rec. rate* shows the performance of the created ensemble on the test set.
For experimental setup (3a) the largest increase in the recognition rate of 5.81 % was obtained by the b./f. feature search ensemble method. This corresponds to an error reduction of 17.21 %. For experimental setup (3a) the largest increase in the recognition rate of 3.8 % was obtained by MSPB. This increase in recognition rate corresponds to an error reduction of 19.35 %.

cat.	ensemble method	cn	combination scheme	rec. rate.
1	random subspace method	25	*ga weight*	69.35 %
2	features={WPC,WC}	5	*special*	70.83 %
3	EBB, $\alpha = 1.5$	10	*ga weight*	69.82 %
4	b./f. feature search, f_2	10	*vote best med*	72.04 %
5	MSPB, e_2	10	*vote best max*	73.55 %

Table 7.16: Information on the best tested ensemble methods for experimental setup (3a). The performance of the base classifier is 66.23 % for the first four categories. In the case of the ensemble method of the fifth category the two base classifiers had the performances of 70.92 % and 70.71 %, respectively.

cat.	ensemble method	cn	combination scheme	rec. rate.
1	AdaBoost	14	*vote best max*	82.02 %
2	features={WPC,WC}	3	*special*	81.07 %
3	SPBE, e_3	14	*vote best max*	82.69 %
4	heuristic b. feature search, f_2, $N = 5$	23	*vote best med*	83.03 %
5	MSPB, e_2	14	*perf. weight*	84.16 %

Table 7.17: Information on the best tested ensemble methods for experimental setup (6a). The performance of the base classifier is 80.48 % for the first four categories. In the case of the ensemble method of the fifth category the two base classifiers had the performances of 80.36 % and 70.22 %, respectively. Please note that for the method of category 2 an improvement from 80.03 % to 83.17 % was achieved on the validation set.

Chapter 8

Conclusions and future work

In this chapter conclusions of the experiments are drawn and future work is suggested.

8.1 Conclusions

The main goal of the thesis was the evaluation of classic and new multiple classifier systems (MCS) methods in the domain of handwriting recognition. The focus of the thesis was on ensemble methods, yet also different combination schemes were considered. As the performance of every MCS method depends on the performance of the base classifiers, a considerable amount of time has been spent on the optimization of the base classifiers. In the context of this thesis a number of new ensemble methods and combination schemes were introduced. In initial experiments with suboptimal base classifiers some well performing ensemble methods and combination schemes were selected and methods for the optimization of the base classifiers were evaluated. In addition the best values of the free parameters of the ensemble methods were estimated. Then the ensemble methods using the best found combination schemes and parameter values and optimized base classifiers were tested.

The best optimized base classifier obtained a recognition rate of 80.48 % in a handwritten word recognition task with a vocabulary of size 3,997, i.e. a 3,997 class-problem was considered. All classic ensemble methods were able to increase this performance. The best recognition rate of 82.02 % was obtained by AdaBoost. Further increases were achieved by using the ensemble methods introduced in this paper. The best performance obtained by the feature search ensemble method was 83.03 % and an ensemble method using several base classifiers even obtained a recognition rate of 84.16 %. The performance increases are not very large, but it should be taken into account that handwriting recognition is one of the most difficult tasks in the domain of pattern recognition.

In summary, it can be stated that the performance of the handwritten word recognizers can be increased by using MCS methods. This is especially the case for some of the methods introduced in this thesis. Most of the introduced MCS methods are not restricted to the domain of handwriting recognition and may also be useful for other pattern recognition problems.

8.2 Future works

In the following some new methods are outlined which may be tested in future studies. In addition, some new experiments using the methods proposed in this thesis are suggested.

8.2.1 Optimization of the parameters with tasks similar to the testing

In Chapter 6 the optimal values of the free parameters of the ensemble methods and the best combination schemes were established. Those optimal parameter values and combination schemes were then used in the experiments of Chapter 7 where another recognition task was considered and an optimized base classifier was used. An alternative approach would be to establish those optimal parameter values and combination schemes using the same recognition task and base classifier. This is because more exact values can be expected with this approach. The above mentioned approach was not used in the thesis because of its high time requirement.

8.2.2 Feature ensemble methods with special HMM classifier combination

In Section 5.4 the special HMM classifier combination scheme was introduced. One of the conditions of its application was that the feature vectors input to the HMMs are the same for all classifiers. For the classifiers produced by the feature selection ensemble methods introduced in Section 4.5 and for related methods this not the case as every classifier uses a different subset of the whole feature set. Using the method outlined in this subsection the special HMM classifier combination scheme can also be applied to classifiers using different subsets of the whole feature set. The three steps of the methods are:

1. The ensemble method is applied and the classifiers are trained.
2. The HMMs of all classifiers are modified. For all features not present in the subset of the whole feature set used by the classifier a “dummy” distribution is inserted. This “dummy” distribution is a single Gaussian where the mean is set to the global mean of the feature and the variance is set to a very large number. Because the variance is very large, different values for that feature result in almost the same likelihood, i.e. the value of the feature is irrelevant. So all features not present in the subset of the whole feature set used by the classifier are irrelevant and the output class of the classifier remains the same. Note that although the output class remains the same there will be a constant offset of the score values of the modified classifier to the score values of the original classifier.
3. Because all HMMs are now modified to allow the input of the whole feature vector, the HMM classifier combination scheme as introduced in Section 5.4 may be applied.

8.2.3 Feature ensemble methods using several base classifiers

In Subsection 4.5.2 the description is given how the ensemble method using one underlying feature selection algorithm can be improved by applying several feature selection algorithms. The results of some experiments of Chapter 6 and Chapter 7 have shown that the use of several feature selection algorithms may increase the performance. The results of the experiments in

Input: A set of feature selection algorithms $S = \{S_1(f,L),\dots,S_m(f,L)\}$, where f is the objective function and L a list of subsets, which are prohibited as result; $\mathrm{BCS} = \{\mathrm{BC}_1,\dots,\mathrm{BC}_m\}$ a set of base classifiers; n is the desired number of classifiers.
Output: Classifiers $C_1,\dots,C_n$.

```
for(i:=1;i≤n;i++)
    select feature selection algorithm CS from set $S$;
    select base classifier BC from set BCS;
    fs is the set of all already used feature sets for base classifier BC;
    $\mathrm{fs_i} = \mathrm{CS(f, fs)}$ where BC is used as base classifier;
    $C_i$ is the classifier which has CB as base classifier and uses t feature set $\mathrm{fs_i}$;
return $C_1,\dots,C_n$;
```

Table 8.1: Ensemble method using several feature selection algorithms and base classifiers

Sections 6.24 and 7.8 lead to the conclusion that the use of several base classifiers may also increase the performance. It is straightforward to combine both approaches. The algorithm of such a combination is shown in Table 8.1 where the same notation as in Table 4.8 was used. The base classifiers and feature selection algorithms can be randomly selected or according to a special order.

Similarly to the above described approach, also the ensemble methods using exhaustive feature search introduced in Subsection 4.5.3 may be extended for the use of several base classifiers. In each step of these ensemble methods all possible feature sets are considered. In the extended version all possible pairs of base classifiers and feature sets are considered. If for example there are two base classifiers each with 511 possible feature sets then a total of 2·511=1022 pairs are considered per step.

8.2.4 More stable *ga weight* combination scheme

The weighted voting scheme with optimized weights introduced in Subsection 5.2.2 and abbreviated as *ga weight* scheme produced good but quite unstable results in the experiments. A way to partially solve the problem of instability is the following: Instead of applying the genetic algorithm once to get a set of weights for the classifiers, the genetic algorithm is applied several times to produce a set of weight sets. In the recognition process the weighted voting combination scheme is then applied with each of these weight sets producing several results. The final result is then determined by voting over the results of all weight sets.

The premise of the above described approach is that genetic algorithm produces weight sets of different qualities. By voting over the output produced by several such weights sets, a more stable result is expected. It may even be that the average recognition rate of the combination scheme increases. (The performance of an ensemble of classifiers combined by voting is often better than the average performance of the individual classifiers). As the calculation of the sum of weights for the classes is much faster than the recognition process of the classifiers, the additional time complexity of the above described approach is negligible.

8.2.5 Adaption of the Viterbi and Baum-Welch algorithms

The classifiers described in Chapter 2 use the Baum-Welch algorithm [76] for the training of the HMMs and the Viterbi algorithm [76] for the recognition. Both algorithms treat the transition probabilities between the states and the likelihood values of the output distributions of the states in the same way, although probabilities and likelihoods are not the same type of values (e.g. the sum of all probabilities is always 1 which is not the case for likelihoods). In text line recognition where the probabilities of the language model are used in conjunction with the likelihood values produced by he recognition of the word models a so called grammar scale factor is used [64, 93]. The probability of the language model to the power of this grammar scale factor is used for the calculation of the final score value[1]. Similarly to the grammar scale factor, a transition probability scale factor may be used, i.e. the original transition probability to the power of this factor may be used for the calculation of the score value. The optimal value for this factor must be empirically determined. Fortunately this value is likely to be independent of the other parameters of the HMMs, like the number of states, and may therefore be separately optimized.

8.2.6 Other training strategies

In [78, 79] methods to select the optimal number of states and allographs[2] for each character HMM are presented. It may be promising to apply these methods to the classifiers described in Chapter 2. The design process of the classifier would then be the following:

1. Training of the base classifier using single Gaussian distributions
2. Optimization of the number of allographs and states
3. Selection of the optimal training method (compare Section 3.2)

The step (2) and (3) can be repeated several times to improve the system.

An alternative design process is the following:

1. Setting the state numbers of the HMMs to a low value
2. Training of the classifier using single Gaussian distributions
3. Facultative: Optimization of the training method
4. Testing of the classifier on the training set in “forced alignment” mode to get the character boundaries [103]
5. Extract character images

[1]When using log likelihoods the log probability of the language model is multiplied by the grammar scale factor.

[2]In the cited papers the term writing variants is used for allographs.

6. For each character separately the optimal number of states, allographs and the best training method of its HMM are determined. For the first two parameters the methods of [78, 79] may be applied or simply a range of different values can be tested. The quality of the model can be measured in three ways:

    (a) Sum of the log likelihood values of the patterns of this character. This measure could lead to severe overfitting, as the HMMs and the patterns of the other characters are not considered. Note that exactly this value is optimized in the Baum-Welch training algorithm.

    (b) Character recognition rate. The likelihood values of all character HMMs for the validation patterns of the actual character are calculated. The number of times the likelihood of the HMM corresponding to the actual character was higher than the likelihoods of all other HMMs is counted. Note that the other HMMs are not yet optimized, so the comparison is not fair. To solve this problem the step (6) may be repeated several times.

    (c) The discriminative training [72] may be used instead of the maximum likelihood training of the Baum-Welch algorithm. In discriminative training the HMM is optimized to distinguish the patterns of the actual character from the patterns of all other characters.

8.2.7 Extension of new boosting methods for several base classifiers

In Subsection 4.6.1 an approach to extend the ensemble methods using one base classifier to ensemble methods using several base classifiers is described. The same approach is, of course, also applicable with the new boosting ensemble methods introduced in Section 4.4. As those ensemble methods produced often better results than the AdaBoost algorithm in the experiments of Chapter 6, it may be expected that the extended versions of the methods also outperform the extended version of AdaBoost.

8.2.8 Modified version of the new boosting algorithms

The new boosting algorithms introduced in Section 4.4 and their multi-base-classifiers extensions introduced in Section 4.6.2 all have the following implicit assumption: The pattern whose inclusion in the training set has the most positive impact on the probability of a pattern x to be classified correctly is the pattern x itself. This leads to the strategy that patterns which will certainly be classified correctly by the ensemble are not included in the training set, as it is more profitable to use instances of misclassified patterns instead. A reason why the above mentioned assumption may be flawed is as follows. The Baum-Welch algorithm optimizes the product of the likelihoods of the training patterns for the HMMs of the correct classes. Yet we want to maximize the differences of the likelihoods for the correct models and the likelihoods for the other models. So it may be that even when the likelihoods of the HMMs of the correct classes are high, other HMMs have higher likelihoods. It was observed that this happens often when the training set consists of instances of only few patterns. As the assumption may be flawed it

could be profitable to include also instances of patterns which will be certainly classified correctly by the ensemble in the training set. Another reason why the inclusion of instances of such patterns in the training set may be a good idea is to avoid overfitting. The overfitting effect is especially strong when training on instances of only few samples and when using Gaussian mixture distributions with many Gaussians.
The results of the new boosting ensemble methods and their multi-base-classifiers extensions were quite promising in Chapter 6, but some results in Chapter 7 were very poor. The reasons why the ensemble methods deteriorated when using the optimized base classifier are the following:

1. Gaussian mixture instead of single Gaussian distributions were used leading to a stronger overfitting.

2. The recognition rate of the optimized base classifier was high, so only few training patterns were misclassified. This led to training sets which contained instances of only few patterns as the correctly classified patterns were not included.

One solution to the above described problem is to include patterns in the training set which are correctly classified by the ensemble. The proposed modification of the ensemble methods is as follows: A parameter β is introduced. The selection probabilities $p(x)$ calculated by the ensemble methods is only used to sample a fraction of $1-\beta$ of the training set[3]. The rest of the training set is sampled from patterns correctly classified by the actual ensemble. Note that the ensemble methods introduced in this thesis are included as a special case with $\beta = 0$. The optimal value for β is application dependent and can be empirically evaluated. Alternatively the obtained recognition rates of the classifier when training on parts of the training set with different sizes may be used as guideline for choosing β.
The optimal value of β for the experimental setups used in Chapter 6 seems to be low, as good results for $\beta = 0$ were achieved. For the experimental setups used in Chapter 7 a higher value of β could lead to good results. Half and half bagging uses an approach similar to the one introduced in this subsection where $\beta = \frac{1}{2}$ (compare Subsection 4.2.5).
In Subsection 4.6.2 it was noted that different boosting methods were used for the design of the advanced boosting algorithms, but only the one based on simple probabilistic boosting produced promising results. It may be that the problem of the other algorithms was the same as the one described above and that with different values for β than 0 they produce good results.

8.2.9 Feature selection ensemble methods with other objective functions

The ensemble methods described in Section 4.5 use four objective functions. The objective functions are based on the performance of the actual ensemble combined by the *vote reject* scheme (compare Subsection 5.2.1). The values of the objective functions are equal to this performance plus an additional term in which the possibility that a wrong ensemble result becomes correct when adding more classifiers is taken into account. The *vote reject* scheme was chosen, because it is simple, robust, and there is a strong correlation of its results with the results of other voting schemes. It may yet be promising to base the object functions on the combination scheme which will be finally used to combine the classifiers during recognition. (In

[3]In effort based boosting a new value of parameter α should be used where $\alpha_{new} = \alpha_{old} \cdot \frac{1}{1-\beta}$

the experiments of this thesis the combination of the classifiers produced by the feature selection ensemble methods was always done by a version of the *vote best* combination scheme). So in the above described approach the combination scheme used during the design process and the combination scheme used for the recognition are the same.

8.2.10 Combination of ensemble methods using partitioning

In Section 4.3 the ensemble methods using partitioning of the training set, also called the partitions based ensemble methods, were introduced. Those ensemble methods divide the training set into different partitions and train the individual classifiers on one or several of these partitions. For the partitioning a clustering algorithm was used to produce clusters of words with similar writing style. In the experiments of this thesis five feature sets were used for the clustering algorithm and the cluster numbers was varied from 2 to 5. Each different combination of feature set and cluster number produced a different ensemble of classifiers. The idea of the new ensemble method is simple: Some ensembles produced by different feature sets and cluster numbers are fused to a single ensemble. This new ensemble contains a large number of diverse classifiers and may therefore be of a high quality.

8.2.11 Sequential application of ensemble methods

In Chapter 4 several ensemble methods were presented. In this subsection the approach of the sequential application of ensemble methods is discussed. The approach starts by applying the "first-order" ensemble method to the base classifier. Then to each of the produced classifiers the "second-order" ensemble method is applied producing an ensemble for each of the classifiers generated by the "first-order" ensemble method. The classifiers of all ensembles produced by the "second-order" ensemble method constitute the final ensemble.
In the following the compatibility of the different ensemble methods for the sequential application is studied. First the ensemble methods are categorized into three types:

- **sampling**: All ensemble methods which train the classifiers on training sets sampled from the original training set are of this type. Such ensemble methods are Bagging, AdaBoost and the new boosting algorithms (see Subsections 4.2.1 and 4.2.2 and Section 4.4). As the partitioning of the training set may also be regarded as a special sampling, the ensemble method using partitioning of the training set (see Section 4.3) also belongs to this type.

- **feature**: The *feature* ensemble methods include all feature selection ensemble methods and the random subspace method (see Section 4.5 and Subsection 4.2.3).

- **architecture:** The methods of this type include every ensemble method producing classifiers for which the value of a parameter is varied. The ensemble method introduced in Subsection 4.2.4 is of this type.

The compatibility of the different types of ensemble methods is shown in Table 8.2. The number in the brackets refer to the explanation given below.
The explanations for the entries given in Table 8.2 are the following:

	second order		
first order	sampling	feature	architecture
sampling	no (1)	no (2)	no (2)
feature	yes	no (1)	no (3)
architecture	yes	yes	(yes) (1)

Table 8.2: Compatibility of the different types of ensemble methods for the sequential application

1. In general two ensemble methods of the same type should not be used sequentially because by doing so the diversity of the classifiers is produced twice in the same way and the diversity of the classifiers in the final ensemble will be low. An exception to this rule is the sequential application of two *architecture* ensemble methods, because two very different parameters may be varied. E.g. the state number of the HMMs and the HMM topology could be varied when using HMM classifiers.

2. The *sampling* ensemble methods should always be used as second-order methods. The reason for this is that they reduce the size of the training data of the individual classifiers[4]. If they are used as first-order methods only a part of the original data is available to the second-order ensemble methods.

3. When using *architecture* and *feature* ensemble methods together, the *architecture* method should be used first. The reason for this is that for the application of the *feature* ensemble methods all parameters of the classifier must already be set.

In Subsection 4.6.1 an approach was proposed in which an ensemble method is applied to each classifier of a set of base classifiers. The resulting ensembles are then fused to one ensemble. This approach may be regarded as the sequential application of the "ensemble method" which created the base classifiers and another ensemble method. Of course the creation of the base classifiers is not really an ensemble method as the base classifiers were designed by hand. Yet in respect of the compatibility for the the sequential application the approach introduced in Subsection 4.6.1 has the same behavior as the *architecture* ensemble methods.
In theory, more than two ensemble methods may sequentially be applied, yet the size of the final ensemble increases exponentially with the number of ensemble methods.

8.2.12 Adaption of the confidence based classifiers reduction scheme

In Section 5.5 the confidence based classifiers reduction (CBCR) combination scheme was introduced. When using the CBCR scheme classifiers for which the confidence of the result is lower than a threshold are not used by the underlying combination scheme. A possible adaption to this scheme is to use a local threshold instead of a global one. A local threshold depends only on the confidence values of the classifiers for the considered test pattern. Such a local threshold for example is a fraction of the best confidence value obtained for the actual test pattern. In the original version of the CBCR scheme the classifiers which obtained low confidence values

[4]Here the number of different training elements is meant, i.e. not counting multiple copies of the training elements.

are not used for the combination. In the adapted version classifiers which obtained low confidence values compared to the confidence values of the other classifiers are not used. Of course it is also possible to use both a local and a global threshold. An example of the use of both thresholds is the following: Only classifiers which have a confidence higher than t_1 and higher than t_2 multiplied by the highest confidence obtained for the actual test pattern are used by the underlying combination scheme.

8.2.13 Weights and normalization parameters calculation on validation set

The weights for the weighted voting schemes introduced in Subsection 5.2.2 and the parameters of the normalization procedure introduced in Subsection 5.3.3 were determined using the training set. Instead of the training set also a validation set could be used to calculate these values. The advantage of using a validation set is that the values are no longer biased as when using the training set. An example of such a biased value is the performance of a 1-nearest neighbor classifier estimated on the training set. A 1-nearest-neighbor classifier always has a recognition rate of 100 % on the training set which is a poor estimation of the performance of the classifier on the test set.

8.2.14 Different order of training files

It was observed that different orders of the training files lead to different HMM classifiers. When using single Gaussian distributions, the differences were only small, but there were considerable differences when using Gaussian mixtures. The large performance difference of the two *subsampling* base classifiers is mainly due to the different training file orders (compare last paragraph of Subsection 7.2.2). The fact that different training file orders lead to different HMM classifiers may be exploited to create an ensemble: for each trained classifier another training file order may be used. The diversity of such classifiers is not expected to be very large, yet the individual performance of the classifiers is high, because, unlike in Bagging, the whole training set is used for training.

8.2.15 Other future work

In this subsection some other future works are listed:

- Many experiments of Chapter 7 were carried out only once. To obtain more stable results the experiments could be repeated several times.
- The IAM database which was shortly described in Section 2.4 contains far more words than used in this thesis. So experiments with larger sets of words may be carried out.
- This thesis only considers word recognition. In future work the topic of multiple classifier systems in line and sentence recognition may be addressed.

Bibliography

[1] F. Alkoot and J. Kittler. Experimental evaluation of expert fusion strategies. *Pattern Recognition Letters*, 20(11-13):1361–1369, 1999.

[2] F. Alkoot and J. Kittler. Feature selection for an ensemble of classifiers. In *Proc. of the 4th World Multiconference on Systematics, Cybernetics and Informatics, Volume VII*, Orlando, Florida, 2000.

[3] A. Allwein, R. Schapire, and Y. Singer. Reducing multiclass to binary: A uniying approach for margin classifiers. *Journal of Machine Learning Research*, 1:113–141, 2000.

[4] R. Bakis. Continuous speech recognition via centisecond acoustic states. In *Proc. of the 91. Meeting of the Acoustic Society of America*, 1976.

[5] A. Bellili, M. Gilloux, and P. Gallinari. An hybrid MLP-SVM handwritten digit recognizer. In *[37]*, pages 28–32.

[6] R. Bippus. 1-dimensional and pseudo 2-dimensional HMMs for the recognition of German literal amounts. In *[35]*, volume 2, pages 487–490.

[7] A. Brakensiek, J. Rottland, A. Kosmala, and G. Rigoll. Off-line handwriting recognition using various hybrid modeling techniques and character n-grams. In *[41]*, pages 343–352.

[8] L. Breiman. Bagging predictors. *Machine Learning 2*, pages 123–140, 1996.

[9] Leo Breiman. Half & half bagging and hard boundry points. Technical Report 534, University of California at Berkeley, 1999.

[10] H. Bunke. Recognition of cursive Roman handwriting - past, present and future. In *[38]*, volume 1, pages 448–459.

[11] H. Bunke and P. Wang, editors. *Handbook of Character Recognition and Document Image Analysis*. World Scientific, 1997.

[12] D. Charlet and D. Jouvet. Optimizing feature set for speaker verification. *Pattern Recognition Letters*, 18:873–879, 1997.

[13] Y. Chim, A. Kassim, and Y. Ibrahim. Dual classifier system for handprinted alphanumeric character recognition. *Pattern Analysis and Applications*, 1:155–162, 1998.

[14] A. Dempster, N. Laird, and D. Rubin. Maximum likelihood estimation from incomplete data. *Journal of the Royal Statistical Society (B)*, 39(1):1–38, 1977.

[15] P. A. Devijver and J. Kittler. *Pattern Recognition: A Statistical Approach.* Prentice-Hall, 1982.

[16] T. Dietterich. Ensemble methods in machine learning. In *[51]*, pages 1–15.

[17] T. Dietterich and E. Kong. Machine learning bias, statistical bias, and statistical variance of decision tree algorithms. Technical report, Departement of Computer Science, Oregon State University, 1995.

[18] T. G. Dietterich and G. Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of Artifical Intelligence Research*, 2:263–286, 1995.

[19] R. Duin and D. Tax. Experiments with classifier combination rules. In *[51]*, pages 16–29.

[20] Y. Freund. Boosting a weak learning algorithm by majority. In *Proceedings of the Third Workshop on Computational Learning Theory*, pages 202–216.

[21] Y. Freund and R. Schapire. A decision-theoretic generalisation of on-line learning and an application to boosting. *Journal of Computer and Systems Sciences*, 55(1):119–139, 1997.

[22] G. Giacinto and F. Roli. Dynamic classifier selection. In *[51]*, pages 177–189.

[23] D. Goldberg. *Genetic Algorithm in Search, Optimization and Machine Learning.* Addison-Wesley, 1989.

[24] D. Guillevic and C. Suen. HMM-KNN word recognition engine for bank cheque processing. In *[39]*, pages 1526–1529, 1998.

[25] S. Günter and Bunke. Optimization of weights in a multiple classifier handwritten word recognition system using a genetic algorithm. *submitted to Electronic Letters on Computer Vision and Image Analysis.*

[26] I. Guyon, R. Haralick, J. Hull, and I. Phillips. Data sets for OCR and document image understanding research. In *[11]*, pages 780–799.

[27] T. Ho. Adaptive coordination of multiple classifiers. In *[34]*, pages 371–384.

[28] T. Ho. The random subspace method for constructing decision forests. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 20(8):832–844, 1998.

[29] T. Ho, J. Hull, and S. Srihari. Decision combination in multiple classifier systems. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 16:66–75, 1994.

[30] G. Houle, D. Aragon, R. Smith, M. Shridhar, and D. Kimura. A multilayered corroboration-based check reader. In *[34]*, pages 495–546.

[31] T. Huang and C. Suen. Combination of multiple experts for the recognition of unconstrained handwritten numerals. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 17:90–94, 1995.

[32] D. Hull. Using statistical testing in the evaluation of retrieval experiments. In *Research and Development in Information Retrieval*, pages 329–338. 1993.

[33] J. Hull. A database for handwritten text recognition research. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 16(5):550–554, May 1994.

[34] J. Hull and S. Taylor, editors. *Document Analysis System II.* World Scientific, 1998.

[35] *Proc. of the 4th Int. Conference on Document Analysis and Recognition*, Ulm, Germany, 1997.

[36] *Proc. of the 5th Int. Conference on Document Analysis and Recognition*, Bangalore, India, 1999.

[37] *Proc. of the 6th Int. Conference on Document Analysis and Recognition*, Seattle, USA, 2001.

[38] *Proc. of the 7th Int. Conference on Document Analysis and Recognition*, Edinburgh, Scotland, 2003.

[39] *Proc. of the 14th Int. Conference on Pattern Recognition*, Brisbane, Australia, 1998.

[40] S. Impedovo, P. Wang, and H. Bunke, editors. *Automatic Bankcheck Processing.* World Scientific, Singapore, 1997.

[41] *Proc. of the 7th Int. Workshop on Frontiers in Handwriting Recognition*, Amsterdam, The Netherlands, 2000.

[42] *Proc. of the 8th Int. Workshop on Frontiers in Handwriting Recognition*, Niagara-on-the Lake, Ontario, Canada, 2002.

[43] A. Jain, P. Duin, and J. Mao. Statistical pattern recognition: A review. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 22(1):4–37, 2000.

[44] A. Kaltenmeier, T. Caesar, J. Gloger, and E. Mandler. Sophisticated topology of hidden Markov models for cursive script recognition. In *Proc. of the Second Int. Conference on Document Analysis and Recognition, Tsukuba Science City, Japan*, pages 139–142, 1993.

[45] H. Kang. Experimental results on the construction of multiple classifiers recognizing handwritten numerals. In *[37]*, pages 1026–1030.

[46] G. Kaufmann, H. Bunke, and M. Hadorn. Lexicon reduction in an HMM-framework based on quantized feature vectors. In *[35]*, volume 2, pages 1097–1101.

[47] G. Kim, V. Govindaraju, and S. Srihari. Architecture for handwritten text recognition systems. In S.-W. Lee, editor, *Advances in Handwriting Recognition*, pages 163–172. World Scientific, 1999.

[48] M. Kirby. *Geometric Data Analysis: An Empirical Approach to Dimensionality Reduction and the Study of Patterns.* John Wiley and Sons, New York, 2001.

[49] J. Kittler, R. Duin, and M. Hatef. On combining classifiers. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 20:226–239, 1998.

[50] J. Kittler, P. Pudil, and P. Somol. Advances in statistical feature selection. In S. Singh, N. Murshed, and W. Kropatsch, editors, *Advances in Pattern Recognition - ICAPR*, pages 425–434, 2001.

[51] J. Kittler and F. Roli, editors. *Proc. of the First Int. Workshop on Multiple Classifier Systems*, Cagliari, Italy, 2000. Springer.

[52] J. Kittler and F. Roli, editors. *Proc. of the Second Int. Workshop on Multiple Classifier Systems*, Cambridge, UK, 2001. Springer.

[53] S. Krishnan, K. Samudravijaya, and P. Rao. Feature selection for pattern classification with gaussian mixture models: A new objective criterion. *Pattern Recognition Letters*, 17:803–809, 1996.

[54] A. Krogh and J. Vedelsby. Neural networks ensembles, cross validation, and active learning. In *Advances in Neural Information Processing Systems*, volume 7, pages 231–238. MIT Press, 1995.

[55] A. Kundu. Handwritten word recognition using hidden Markov model. In *[11]*, pages 157–182.

[56] L. Lam, Y. Huang, and C. Suen. Combination of multiple classifier decisions for optical character recognition. In *[11]*, pages 79–101.

[57] L. Lam and C. Suen. Structural classification and relaxation matching of totally unconstrained handwritten zip-code numbers. *Pattern Recognition*, 21(1):19–31, 1988.

[58] L. Lam and C. Suen. Optimal combinations of patterns. *Pattern Recognition Letters*, 16:945–954, 1995.

[59] Di V. Lecce, G. Dimauro, A. Guerriero, S. Impedovo, G. Pirlo, and A. Salzo. A perturbation-based approach for multi-classifier system design. In *[41]*, pages 553–558.

[60] J. Lee, J. Kim, and J. H. Kim. Data driven design of HMM topology for on-line handwriting recognition. In *[41]*, pages 239–248.

[61] L. Lee and N. Gomes. Disconnected handwritten numeral image recognition. In *[35]*, volume 2, pages 467–470.

[62] Y. Lu and P. Shi. An implementation of postal numerals segmentation and recognition system for chinese business letters. In *[36]*, pages 725–728.

[63] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the 5th Berkeley Symposium on Mathematical statistics and probability*, volume 1, pages 281–297, Berkeley, 1967.

[64] U. Marti. *Offline Erkennung handgeschriebener Texte.* PhD thesis, University of Bern, Switzerland, 2000.

[65] U. Marti and H. Bunke. Using a statistical language model to improve the performance of an HMM-based cursive handwriting recognition system. *Int. Journal of Pattern Recognition and Artificial Intelligence*, 15:65–90, 2001.

[66] U. Marti and H. Bunke. The IAM-database: An English sentence database for off-line handwriting recognition. *Int. Journal of Document Analysis and Recognition*, 5:39–46, 2002.

[67] S. Marukatat, T. Artieres, P. Gallinari, and B. Dorizzi. Rejection measures for handwriting sentence recognition. In *[42]*, 2002.

[68] K. Maruyama and Y. Nakano. Recognition method for cursive Japanese word written in latin characters. In *[41]*, pages 133–142.

[69] M. P. Narendra and K. Fukunaga. A branch and bound algorithm for feature subset selection. *IEEE Trans. on Computers*, 26(9):917–922, 1977.

[70] D. Niedermann and M. Bruhin. Clustering of handwriting. Technical report, University Bern, 2001. (In German).

[71] H. Nishimura, M. Kobayashi, M. Maruyama, and Y. Nakano. Off-line character recognition using HMM by multiple directional feature extraction and voting with bagging algorithm. In *[36]*, pages 49–52.

[72] R. Nopsuwanchai and D. Povey. Discriminative training for HMM-base offline handwritten character recognition. In *[38]*, pages 114–118.

[73] D. Partridge and W. Yates. Engineering multiversion neural-net systems. *Neural Computation*, 8(4):869–893, 1996.

[74] P. Pudil, J. Novovičovà, and J. Kittler. Floating search methods in feature selection. *Pattern Recognition Letters*, 15(11):1119–1125, 1994.

[75] L. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proc. of the IEEE*, 77(2):257–285, 1989.

[76] L. Rabiner and B. Juang. *Fundamentals of Speech Recognition.* Prentice Hall, 1993.

[77] F. Rahman and M. Fairhurst. Serial combination of multiple experts: A unified evaluation. *Pattern Analysis and Applications*, 2:292–311, 1999.

[78] M. Schambach. Determination of the number of writing variant with an HMM base cursive word recognition system. In *[38]*, pages 119–123.

[79] M. Schambach. Model length adaption of an HMM base cursive word recognition system. In *[38]*, pages 109–113.

[80] R. Schapire. The strength of weak learner. *Machine Learning*, 5(2):197–227, 1990.

[81] R. Schapire. Using output codes to boost multiclass learning problems. In *Proc. of the 14th Int. Conference on Machine Learning*, pages 313–321, 1997.

[82] R. Schapire. The boosting approach to machine learning. An overview. In *Proc. of the MSRI Workshop on Nonlinear Estimation and Classification*, pages 149–172, 2002.

[83] R. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37(3):297–336, 1999.

[84] A. Senior and A. Robinson. An off-line cursive handwriting recognition system. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 20(3):309–321, March 1998.

[85] J. Simon. Off-line cursive word recognition. *Proc. of the IEEE*, 80(7):1150–1161, Jul 1992.

[86] P. Somol and P. Pudil. Oscillating search algorithms for feature selection. In *Proc. of 15th Int. Conference on Pattern Recognition*, pages 406–409, 2000.

[87] A. Spanjersberg. Combination of different systems for the recognition of handwritten digits. In *Proc. of the Second Int. Joint Conference on Pattern Recognition*, pages 208–209, Copenhagen, Denmark, 1974.

[88] S. Srihari. Handwritten address interpolation: A task of many pattern recognition problems. *Int. Journal of Pattern Recognition and Artificial Intelligence*, 14:663–674, 2000.

[89] M. Srinivas and L. Patnaik. Genetic algorithms: A survey. *Computer*, pages 17–26, 1994.

[90] C. Suen and L. Lam. Multiple classifier combination methodologies for different output level. In *[51]*, pages 52–66.

[91] C. Suen, C. Nadal, R. Legault, T. Mai, and L. Lam. Computer recognition of unconstrained handwritten numerals. *Proc. of the IEEE*, 80:1162–1180, 1992.

[92] H. Teulings and L. Schomaker. Unsupervised learning of prototype in cursive script recognition. In S. Impedovo and J. Simon, editors, *From Pixels to Features III: Frontiers in Handwriting Recognition*, pages 61–73. 1992.

[93] A. Vinciarelli. *Offline Cursive Handwriting: From Word to Text Recognition*. PhD thesis, University of Bern, Switzerland, 2003.

[94] A. Vinciarelli. A survey on off-line cursive script recognition. *Pattern Recognition*, 35(7):1433–1446, 2003.

[95] A. Vinciarelli and S. Bengio. Writer adaptation techniques in HMM based off-line cursive script recognition. *Pattern Recognition Letters*, 23(8):905–916, June 2002.

[96] A. Vinciarelli, S. Bengio, and H. Bunke. Offline recognition of large vocabulary cursive handwritten text. In *[38]*, volume 2, pages 1101–1105.

[97] W. Wang, A. Brakensiek, A. Kosmala, and G. Rigoll. Multi-branch and two-pass HMM modeling approaches for off-line cursive handwriting recognition. In *[37]*, pages 231–235.

[98] W. Wang, A. Brakensiek, and G. Rigoll. Combination of multiple classifiers for handwritten word recognition. In *[42]*, pages 117–122.

[99] C. Wenzel, S. Baumann, and T. Jager. Advances in document classification by voting of competitive approaches. In *[34]*, pages 385–405.

[100] R. Wilkinson, J. Geist, S. Janet, P. Grother, C. Burges, R. Creecy, B. Hammond, J. Hull, N. Larsen, T. Vogl, and C. Wilson. The first census optical character recognition systems conf. #nistir 4912. Technical report, The U.S. Bureau of Census and the National Institute of Standards and Technology, Gaithersburg, MD, 1992.

[101] K. Yamasaki. Automatic allograph categorization based on stroke clustering for on-line handwritten japanese character recognition. In *[39]*, pages 1150–1152.

[102] S. Young, J. Jansen, J. Odell, D. Ollason, and P. Woodland. *The HTK Hidden Markov Model Toolkit Book*. Entropic Cambridge Research Laboratory, http://htk.eng.cam.ac.uk/, 1995.

[103] M. Zimmermann and H. Bunke. Hidden markov model length optimization for handwriting recognition systems. In *[42]*, pages 369–374.

[104] M. Zimmermann and H. Bunke. Automatic segmentation of the IAM off-line database for handwritten English text. In *Proc. of the 16th Int. Conference on Pattern Recognition*, volume 4, pages 35–39, Quebec, Canada, 2002.

[105] M. Zimmermann, J. Chappelier, and H. Bunke. Parsing n-best lists of handwritten sentences. In *[38]*, volume 1, pages 572–576.

Appendix A

Appendix

A.1 Figures and Tables

In this section some figures and tables of the experiments are displayed. These figures and tables are shown in the appendix because their inclusion in the sections in which they are referred to would have severely hampered the readability.

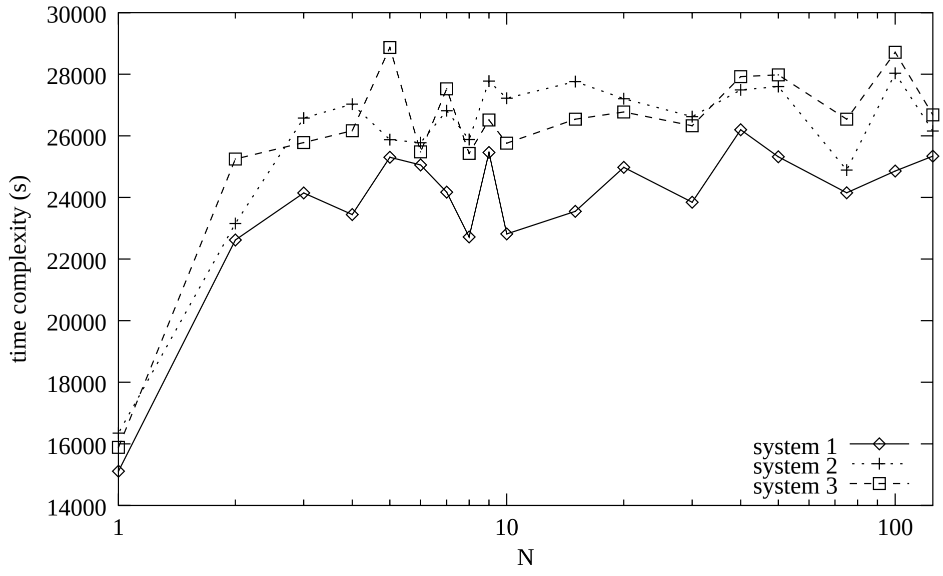


Figure A.1: Time complexity of N-best recognition in seconds

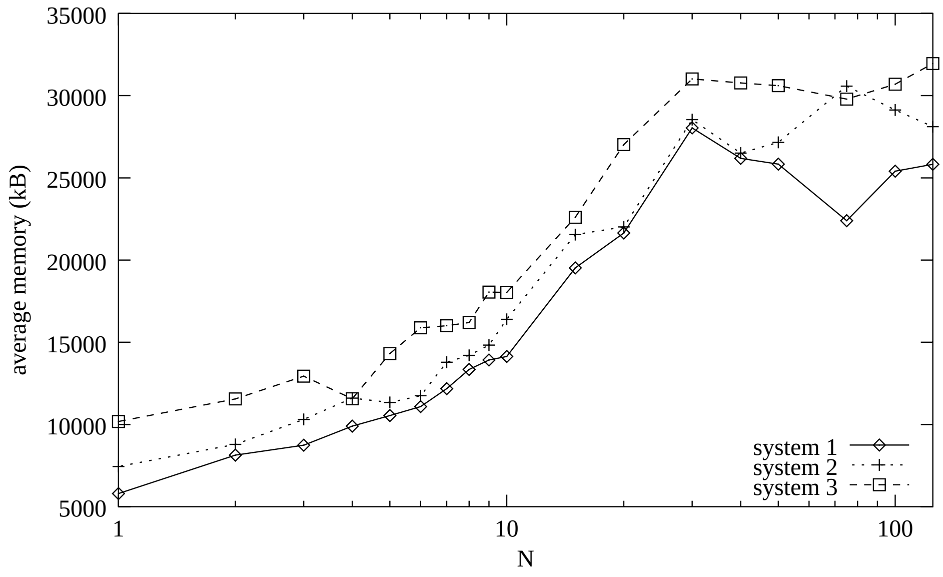


Figure A.2: Memory complexity of N-best recognition

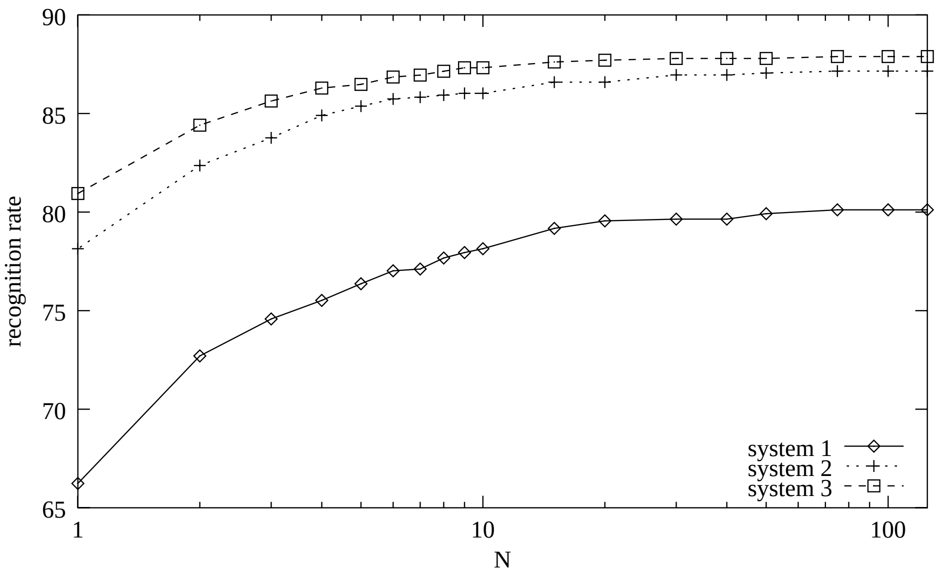


Figure A.3: Recognition rate of the classifiers outputting N-best lists

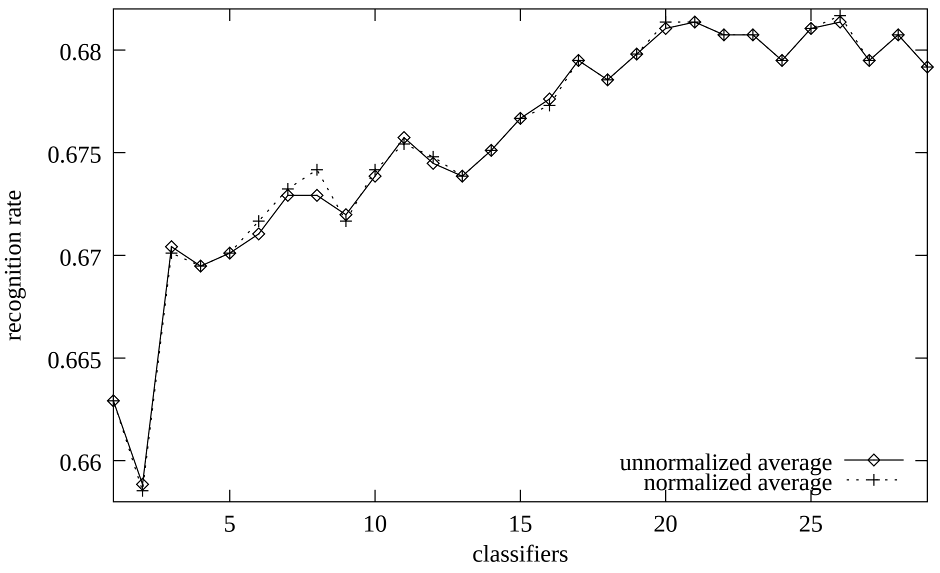


Figure A.4: Comparison of the normalized and unnormalized *vote best average* combination schemes for Bagging

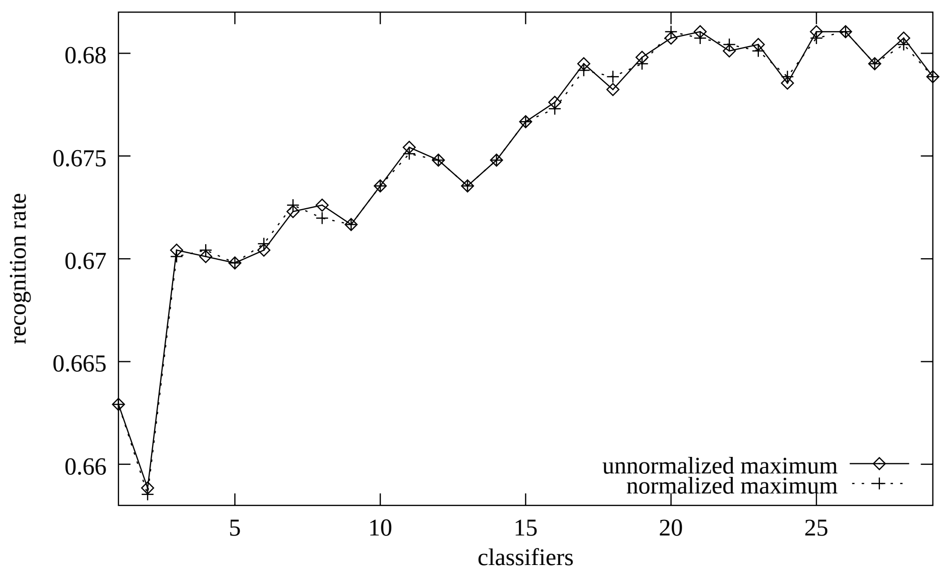


Figure A.5: Comparison of the normalized and unnormalized *vote best maximum* combination schemes for Bagging

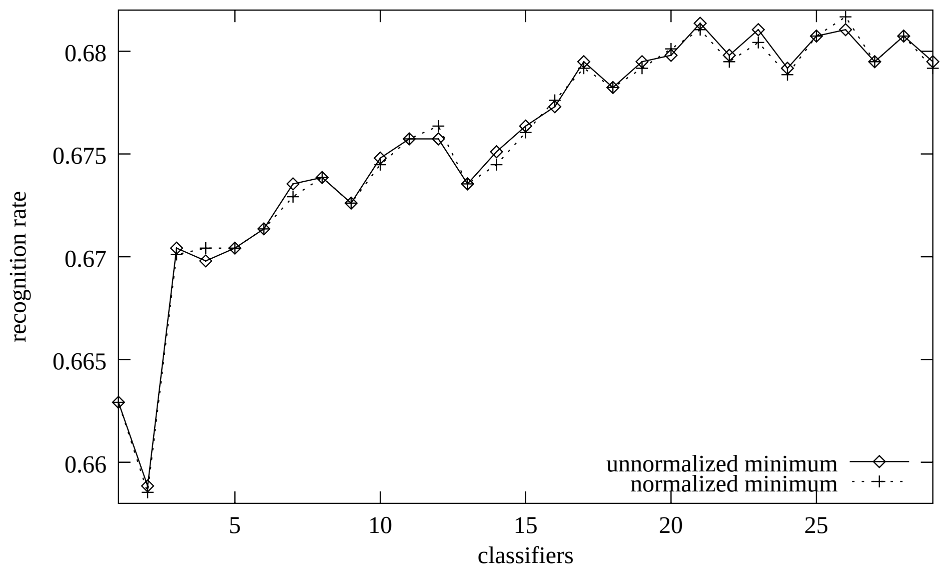


Figure A.6: Comparison of the normalized and unnormalized *vote best minimum* combination schemes for Bagging

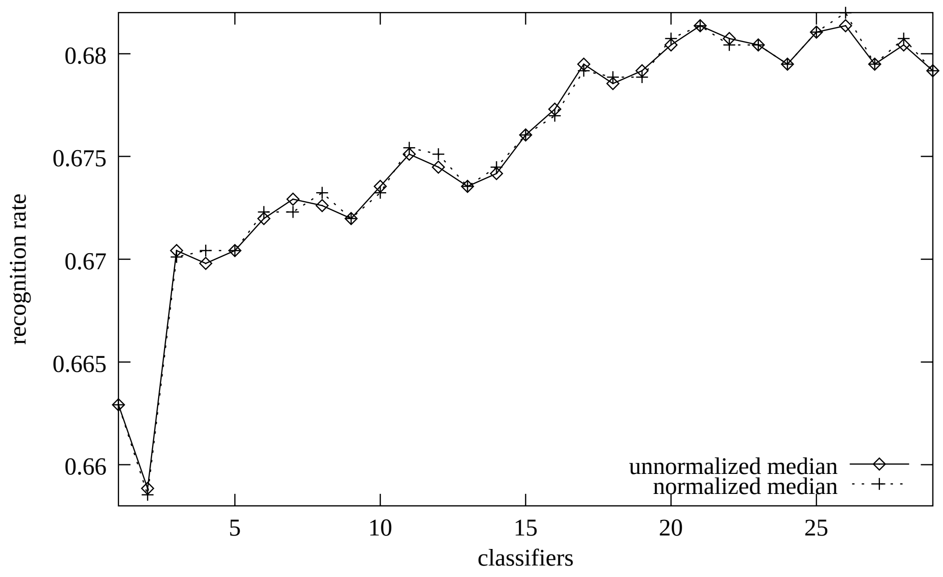


Figure A.7: Comparison of the normalized and unnormalized *vote best median* combination schemes for Bagging

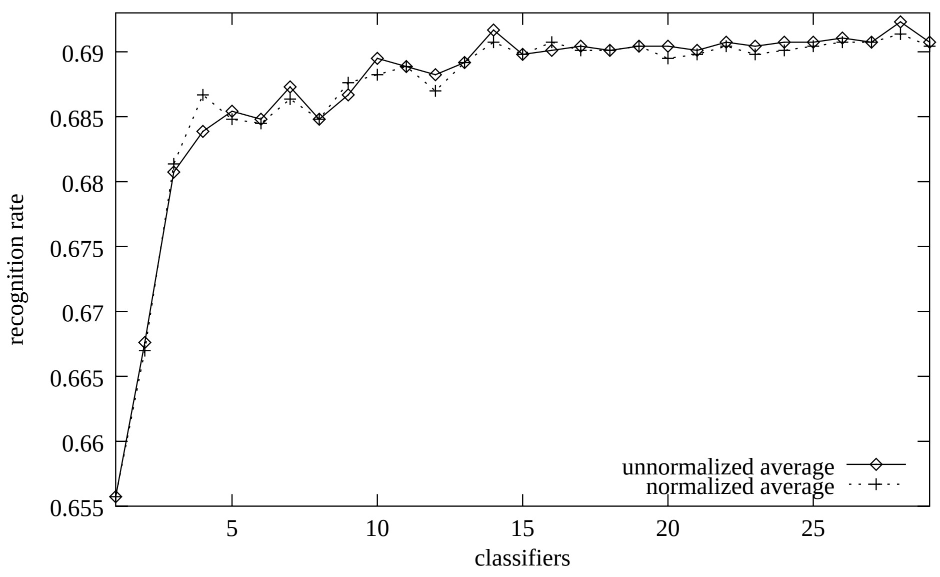


Figure A.8: Comparison of the normalized and unnormalized *vote best average* combination schemes for AdaBoost

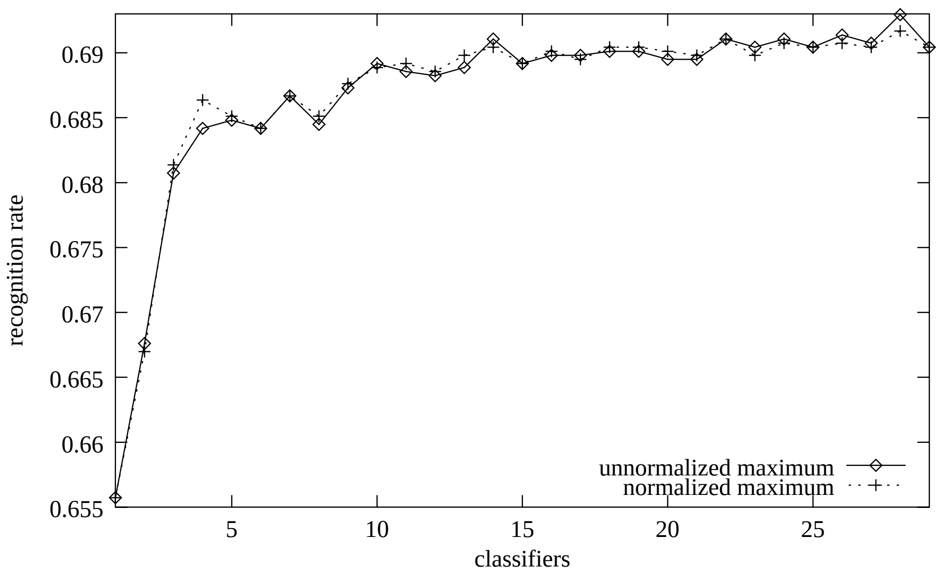


Figure A.9: Comparison of the normalized and unnormalized *vote best maximum* combination schemes for AdaBoost

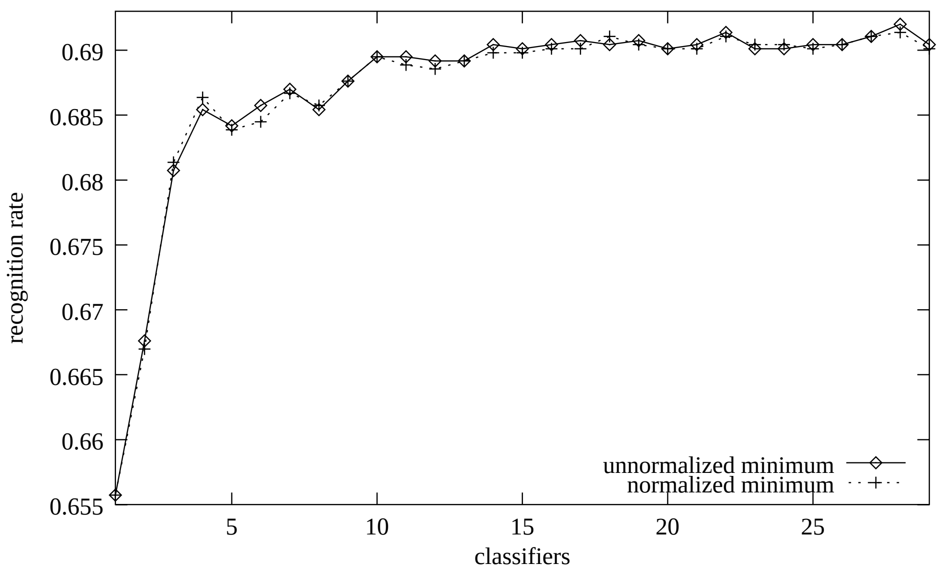


Figure A.10: Comparison of the normalized and unnormalized *vote best minimum* combination schemes for AdaBoost

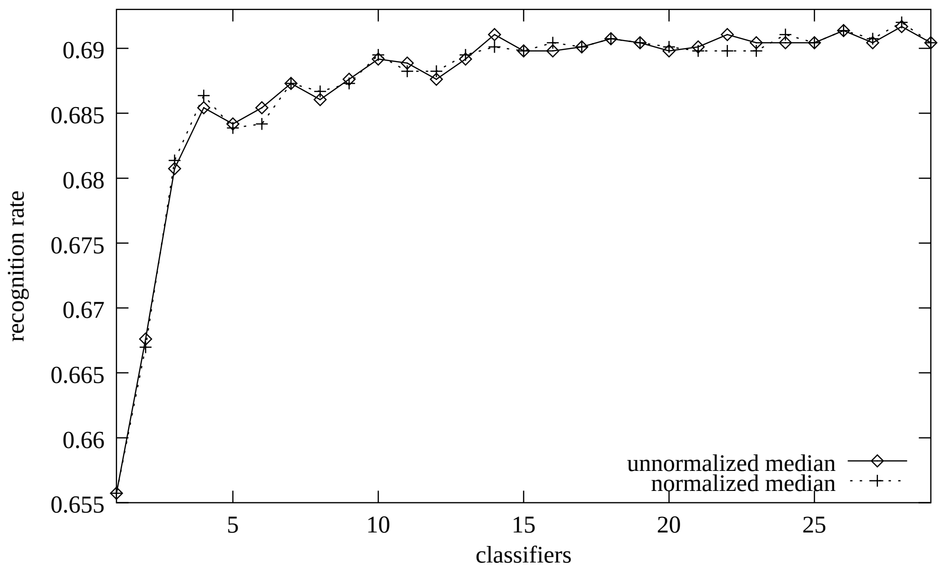


Figure A.11: Comparison of the normalized and unnormalized *vote best median* combination schemes for AdaBoost

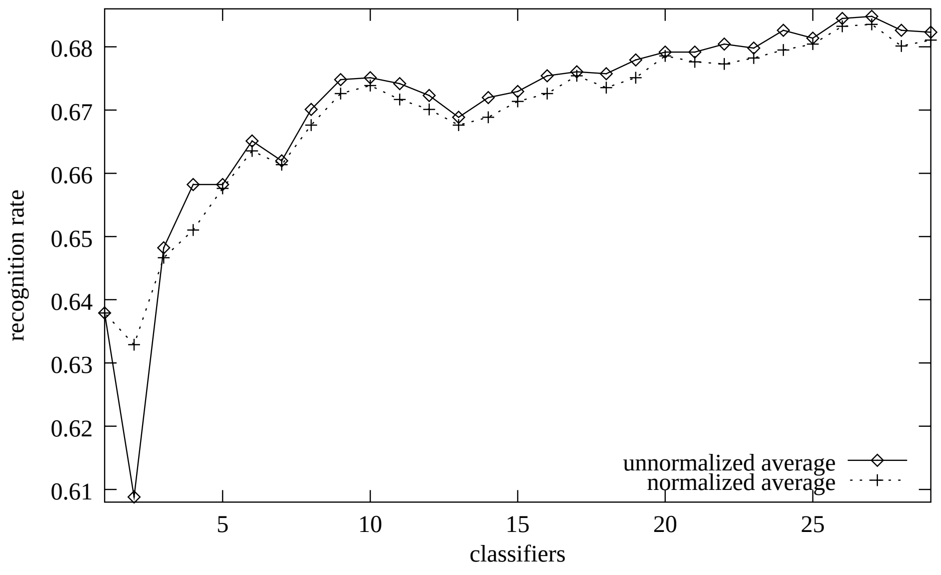


Figure A.12: Comparison of the normalized and unnormalized *vote best average* combination schemes for the random subspace method

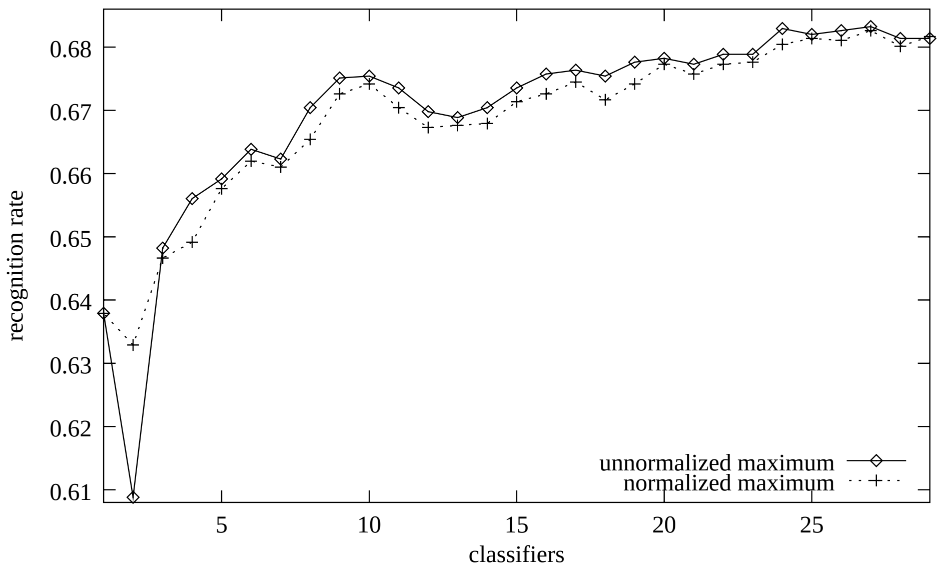


Figure A.13: Comparison of the normalized and unnormalized *vote best maximum* combination schemes for the random subspace method

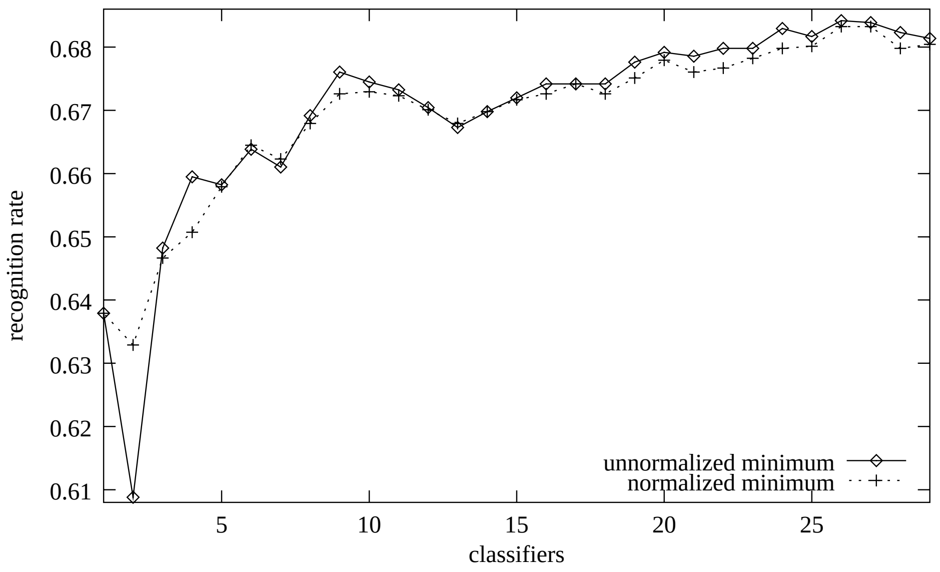


Figure A.14: Comparison of the normalized and unnormalized *vote best minimum* combination schemes for the random subspace method

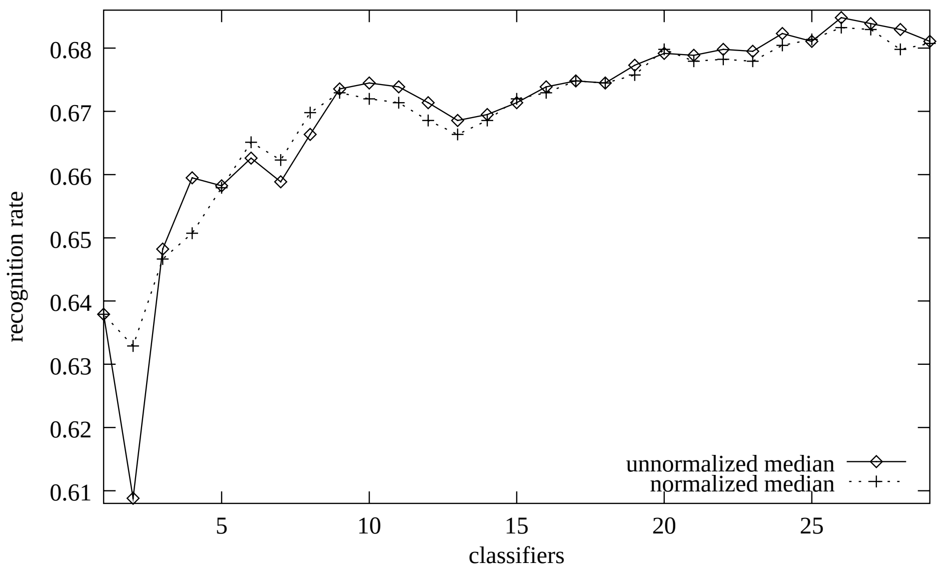
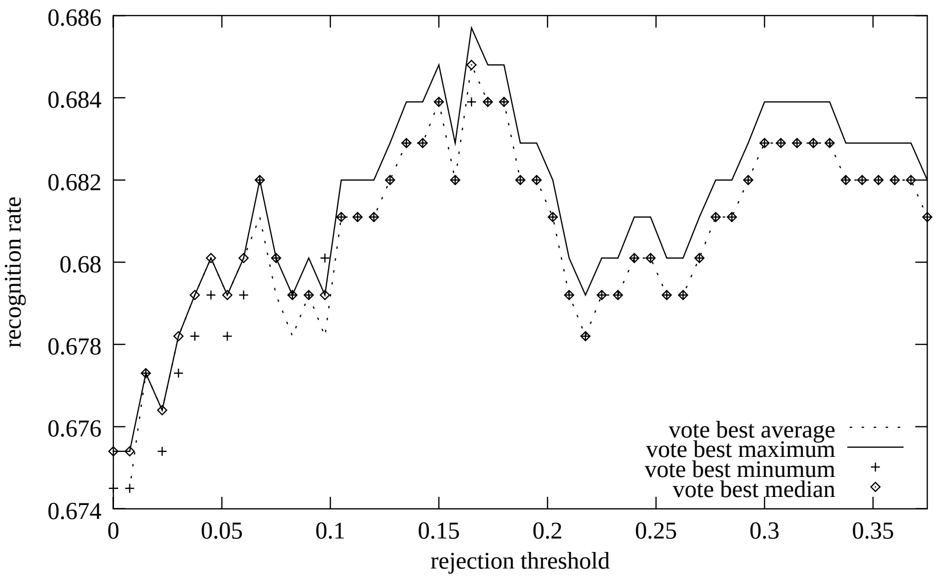


Figure A.15: Comparison of the normalized and unnormalized *vote best median* combination schemes for the random subspace method



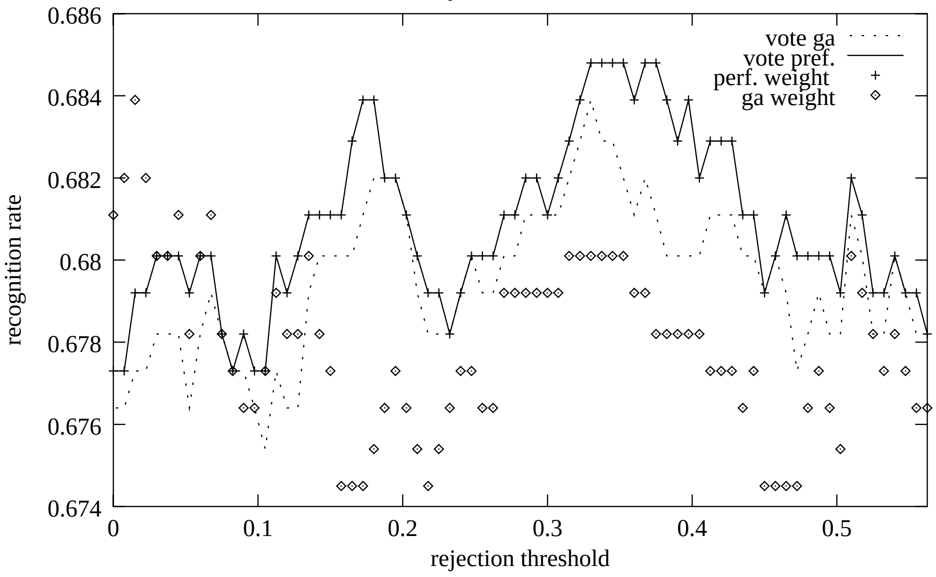


Figure A.16: Performance of the confidence based classifiers reduction combination scheme using different underlying combination schemes for Bagging

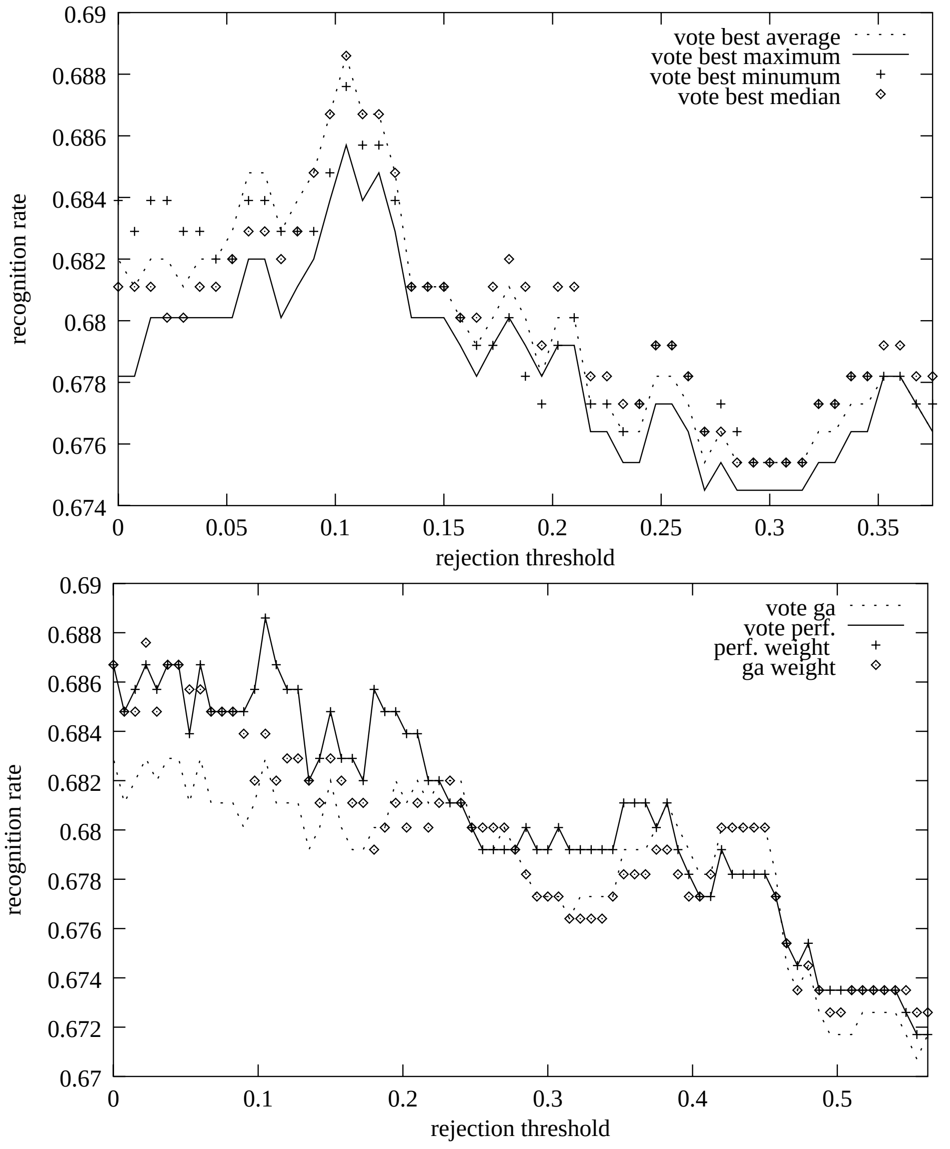
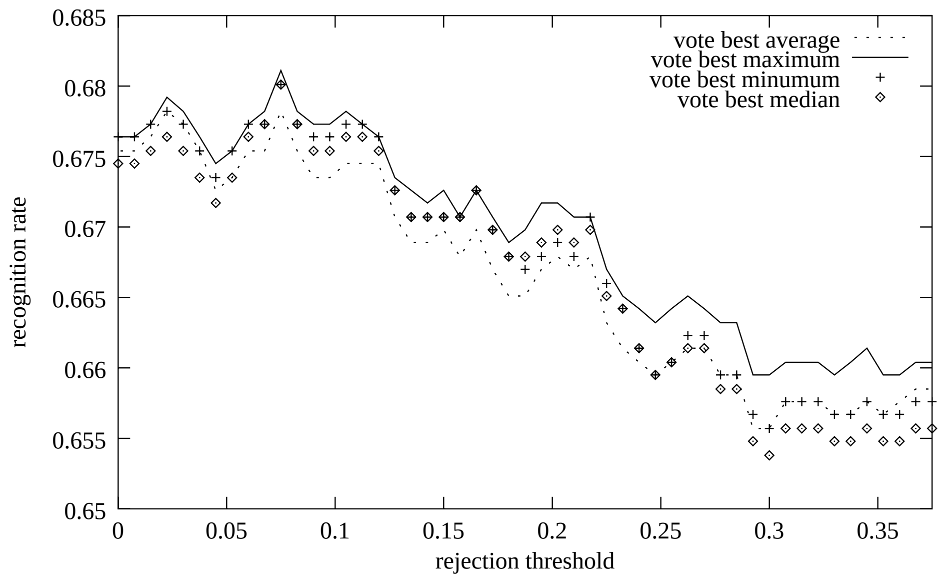


Figure A.17: Performance of the confidence based classifiers reduction combination scheme using different underlying combination schemes for AdaBoost



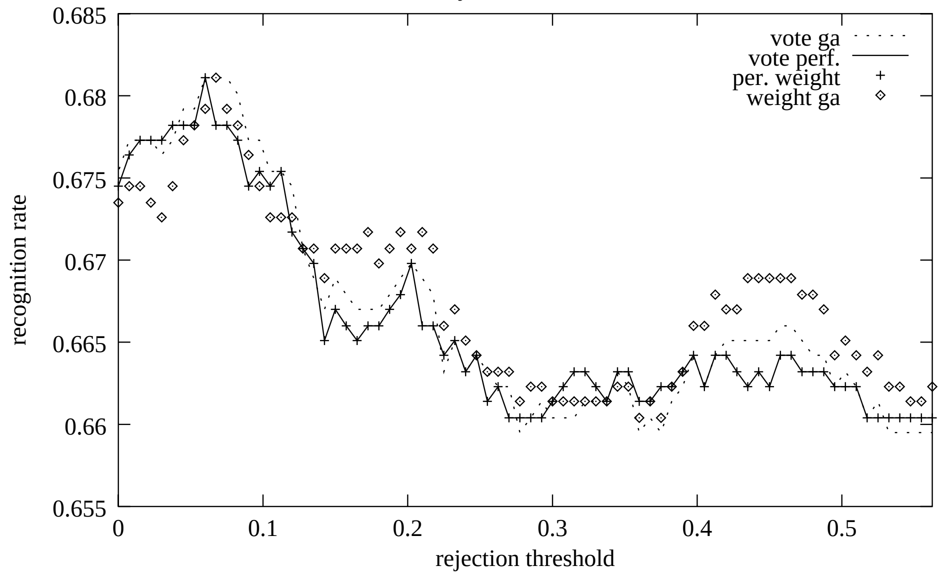
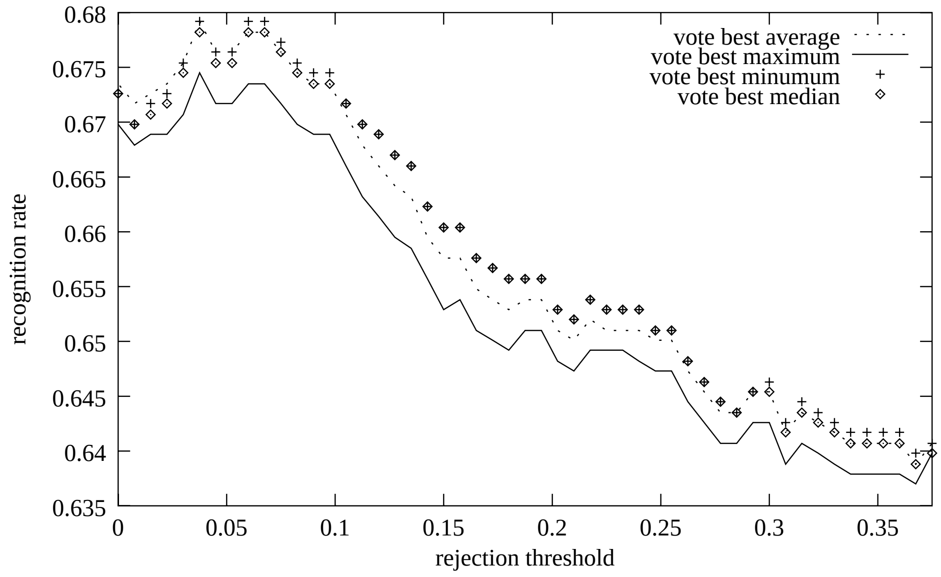


Figure A.18: Performance of the confidence based classifiers reduction combination scheme using different underlying combination schemes for the random subspace method



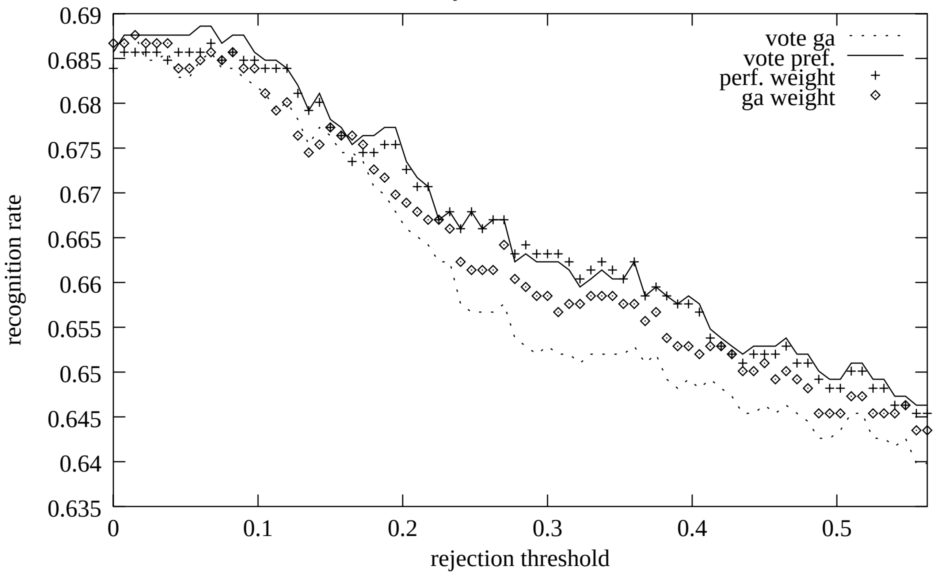


Figure A.19: Performance of the confidence based classifiers reduction combination scheme using different underlying combination schemes for architecture variation

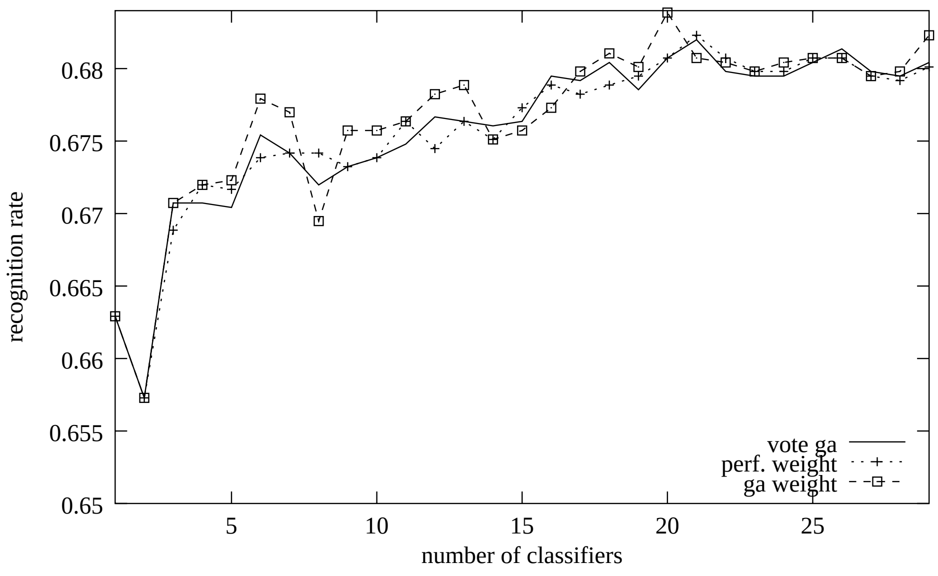


Figure A.20: Results of the three combination schemes using weights for Bagging

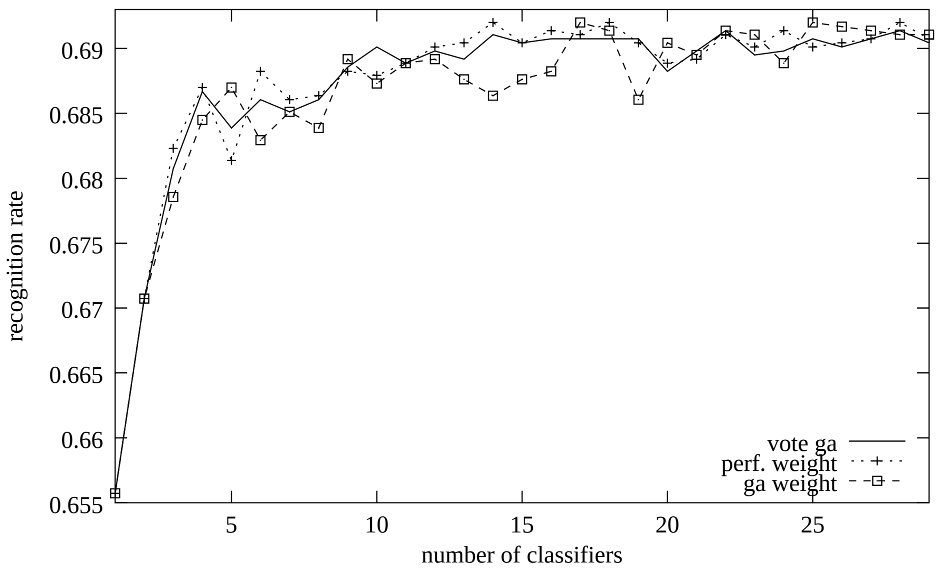


Figure A.21: Results of the three combination schemes using weights for AdaBoost

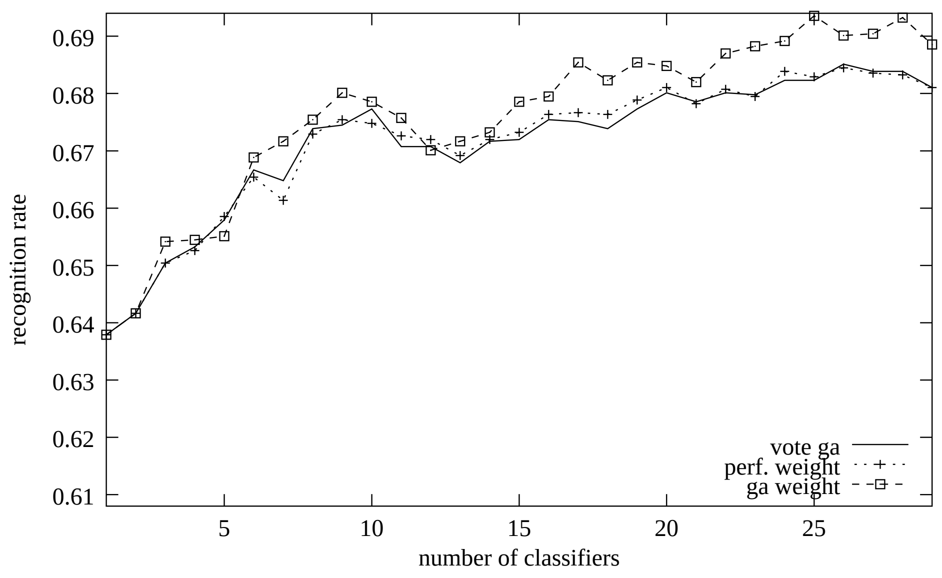


Figure A.22: Results of the three combination schemes using weights for the random subspace method

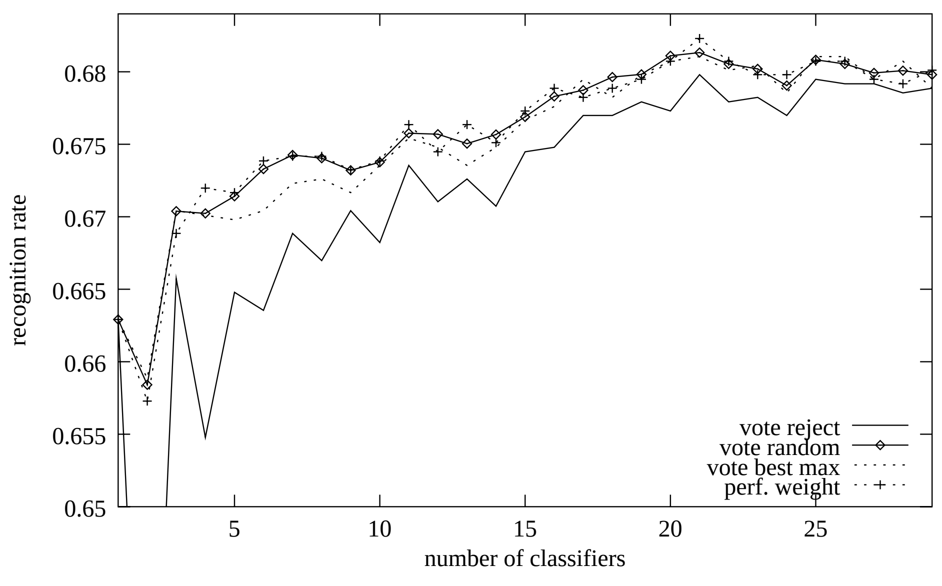


Figure A.23: Results of different combination schemes for Bagging

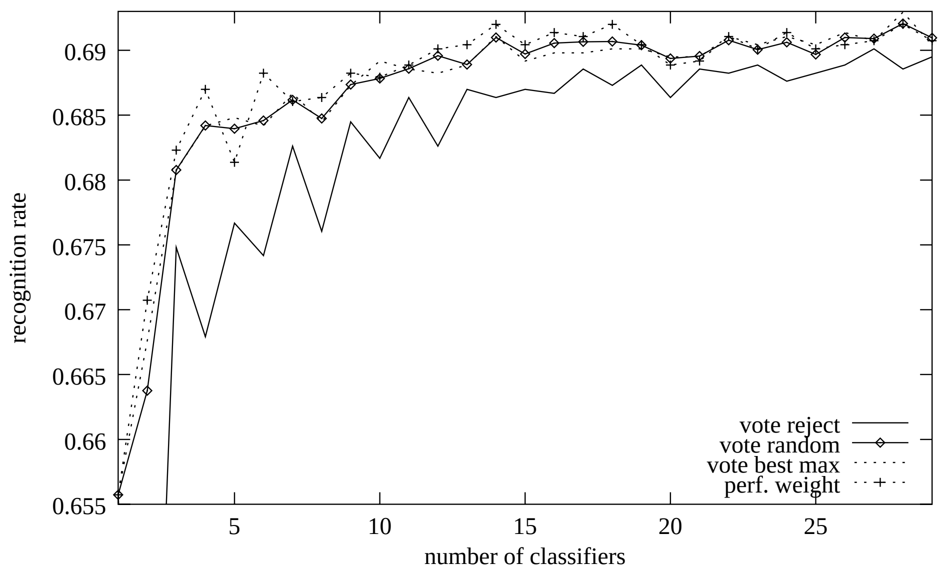


Figure A.24: Results of different combination schemes for AdaBoost

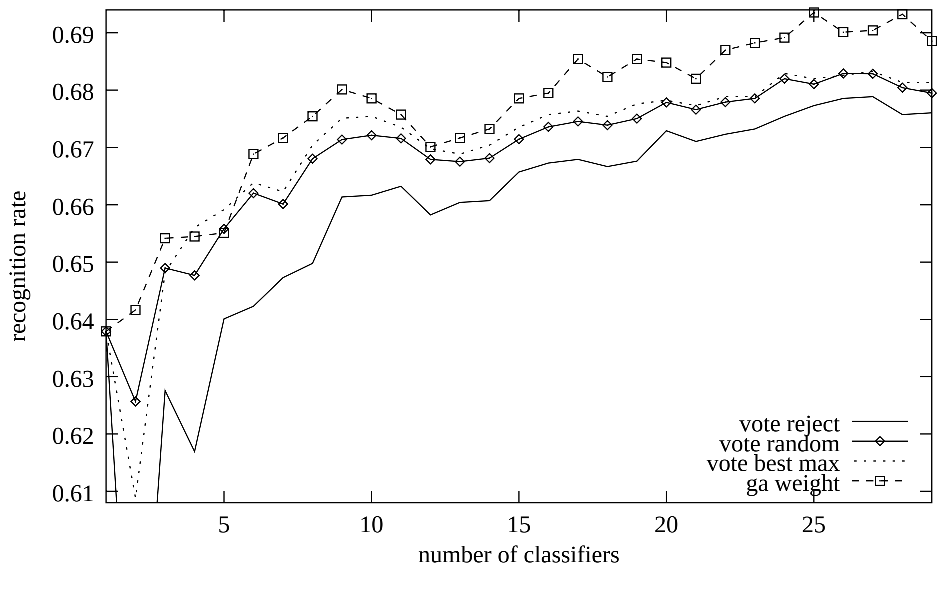


Figure A.25: Results of different combination schemes for the random subspace method

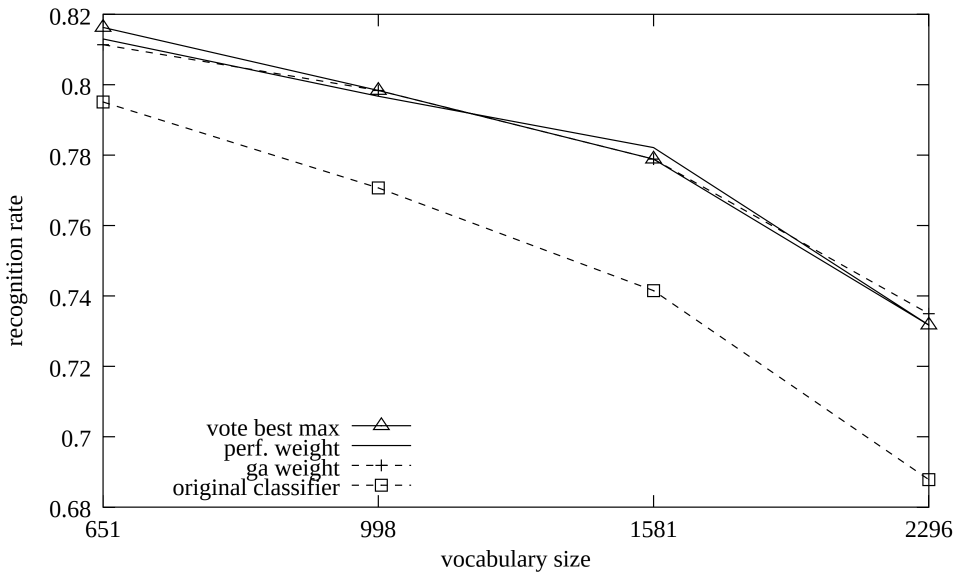


Figure A.26: Performance of Bagging for different vocabulary sizes

method	characters																		
	“	’	:	.	,	(	)	#	;	*	-	0	1	2	3	4	5	6	7
Quantile	13	12	10	5	6	17	23	46	18	71	21	26	13	28	25	30	25	22	32
Bakis	9	12	6	4	6	10	10	53	8	26	20	18	15	19	19	16	19	16	28
	8	9	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q
Quantile	17	25	19	21	19	21	15	13	19	29	11	18	20	10	26	19	16	19	16
Bakis	14	16	18	17	14	19	16	13	16	17	10	15	16	9	25	18	15	16	17
	r	s	t	u	v	w	x	y	z	A	B	C	D	E	F	G	H	I	J
Quantile	11	14	14	18	20	28	23	20	32	24	27	22	22	26	13	21	22	11	24
Bakis	12	13	13	18	16	23	16	18	20	18	24	17	23	19	24	20	18	10	17
	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z			
Quantile	27	19	28	29	29	22	60	21	19	28	27	26	27	63	20	22			
Bakis	18	15	23	22	21	22	22	21	18	22	19	16	26	24	17	19			

Table A.1: State numbers of the character models obtained by the Quantile and Bakis methods

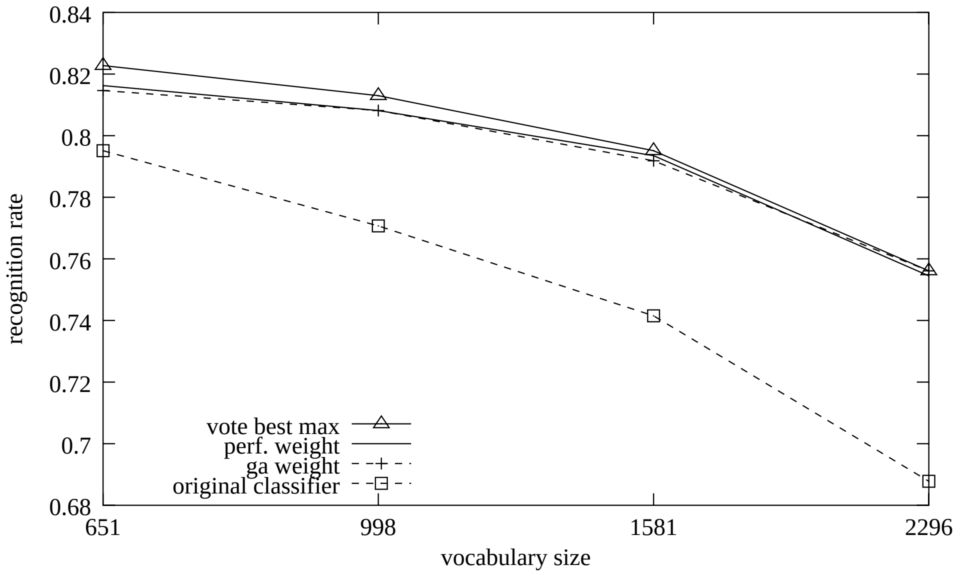


Figure A.27: Performance of AdaBoost for different vocabulary sizes

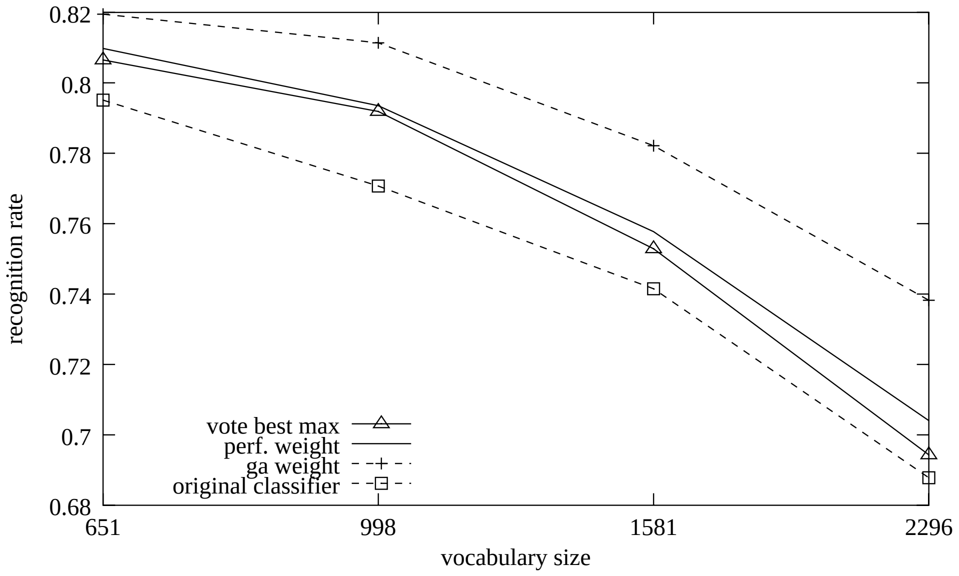


Figure A.28: Performance of the random subspace method for different vocabulary sizes

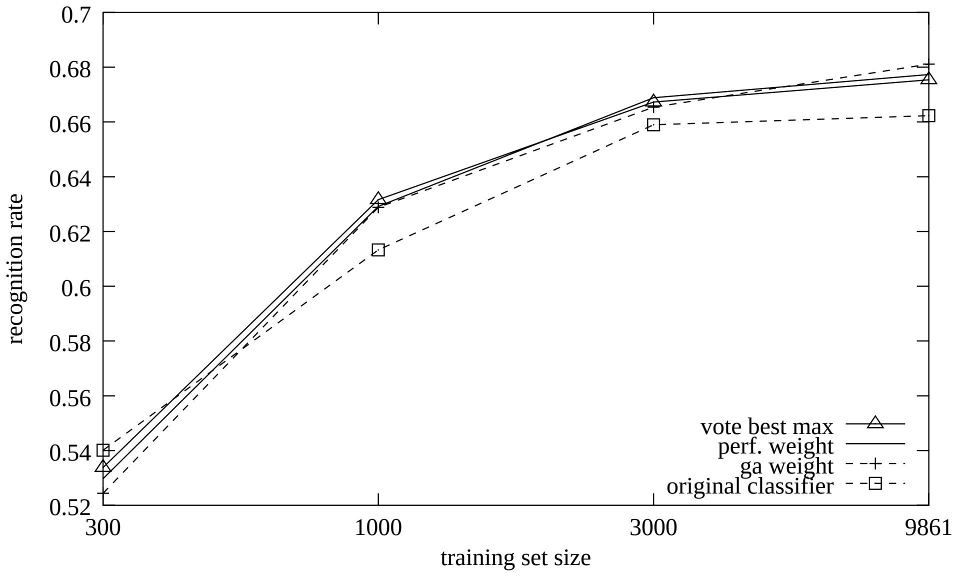


Figure A.29: Performance of Bagging for different training set sizes

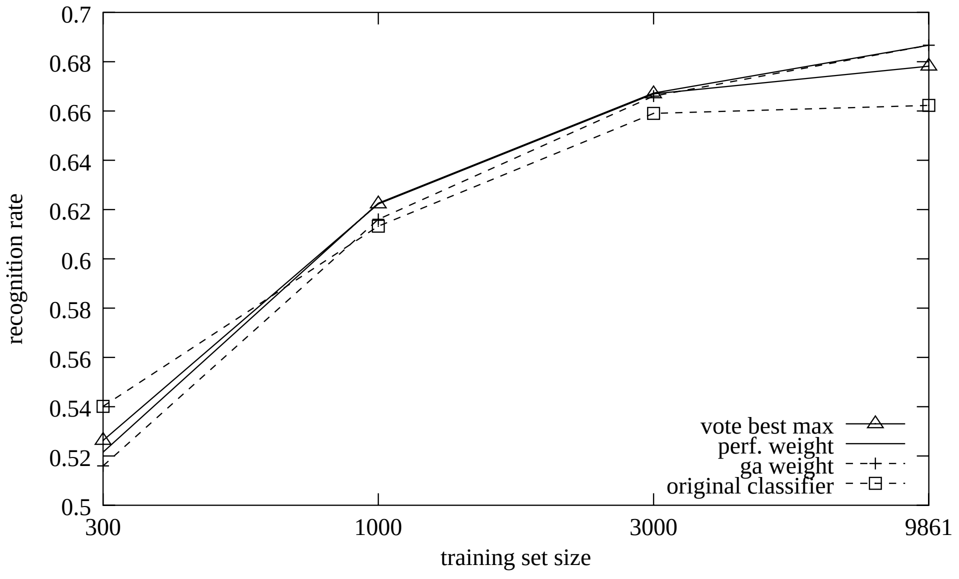


Figure A.30: Performance of AdaBoost for different training set sizes

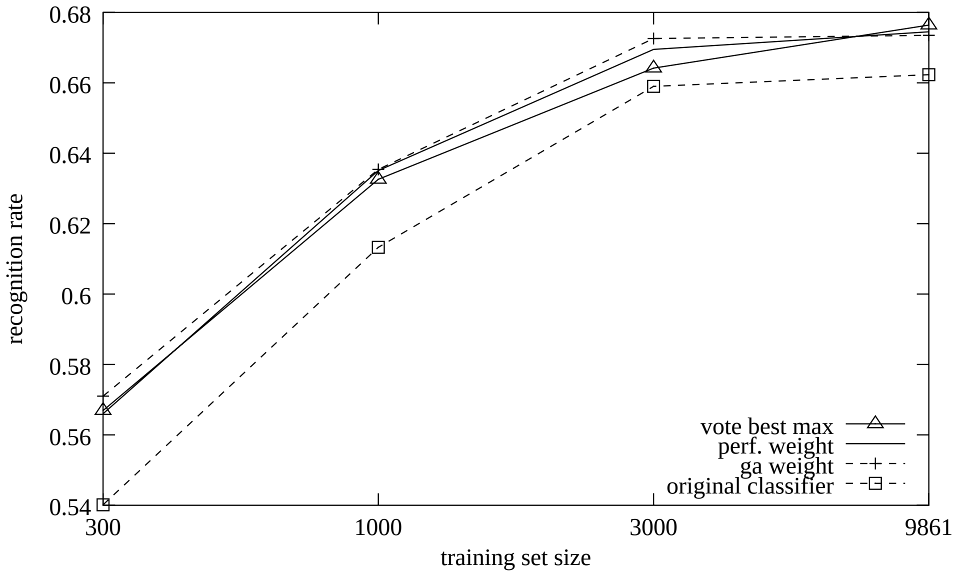


Figure A.31: Performance of the random subspace method for different training set sizes

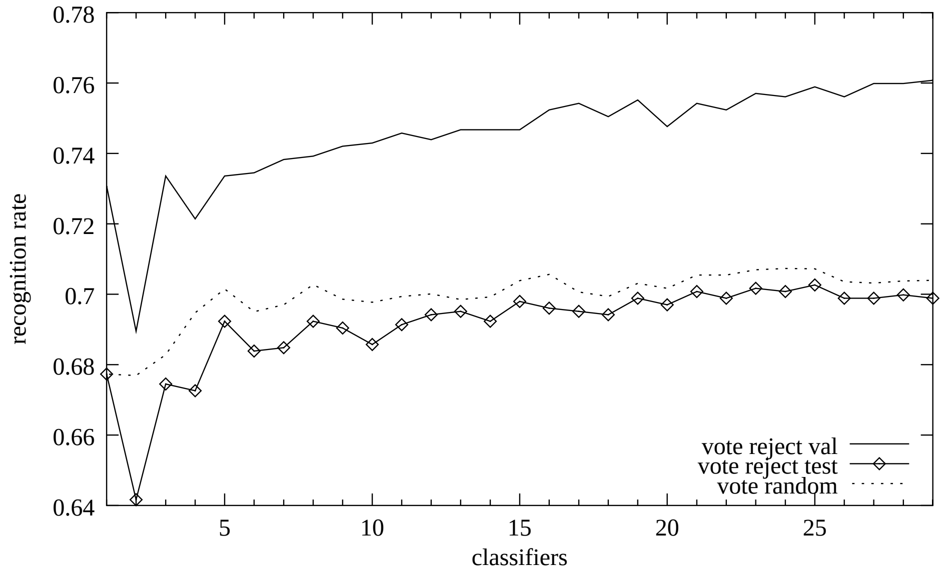


Figure A.32: Results of the *b. ensemble* method using objective function f_2

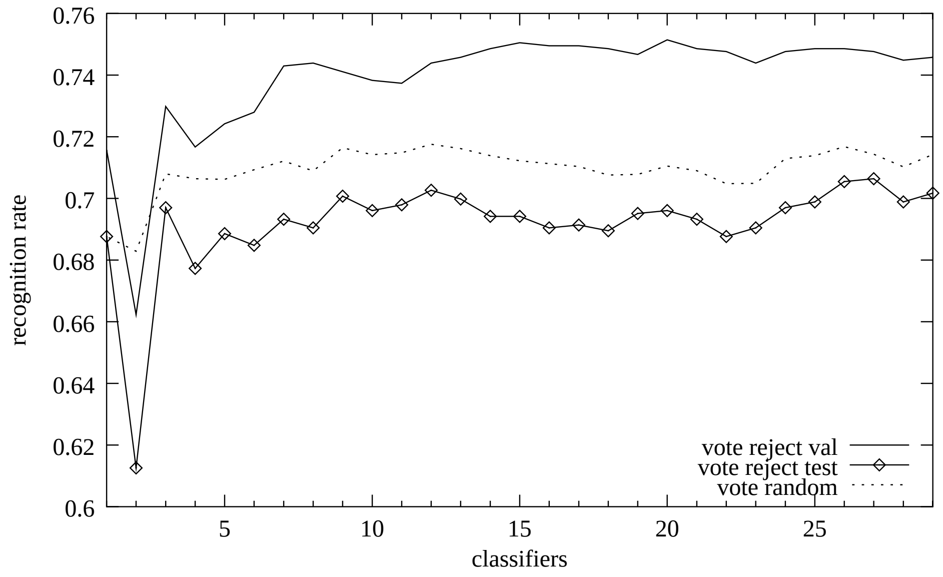


Figure A.33: Results of the *f. ensemble* method using objective function f_2

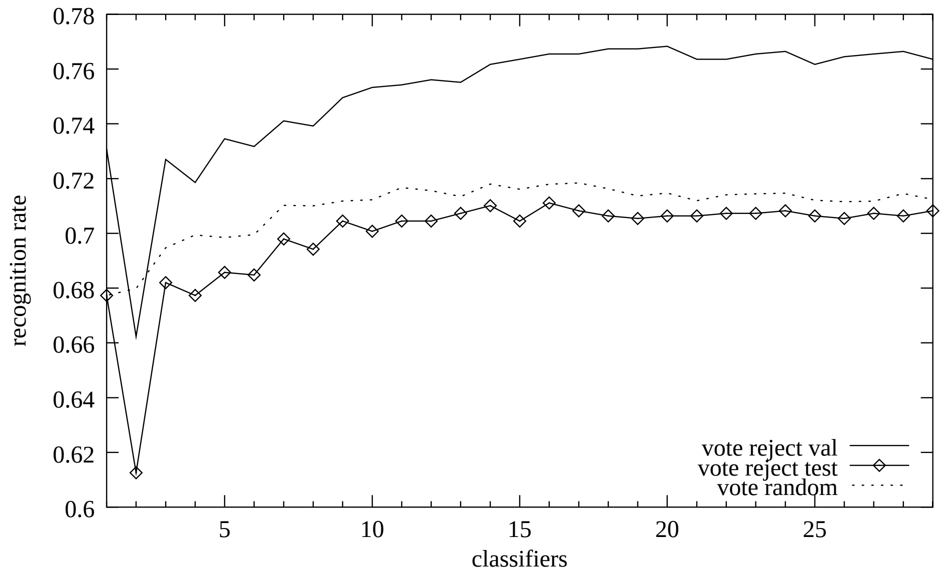


Figure A.34: Results of the *b./f. ensemble* method using objective function f_2

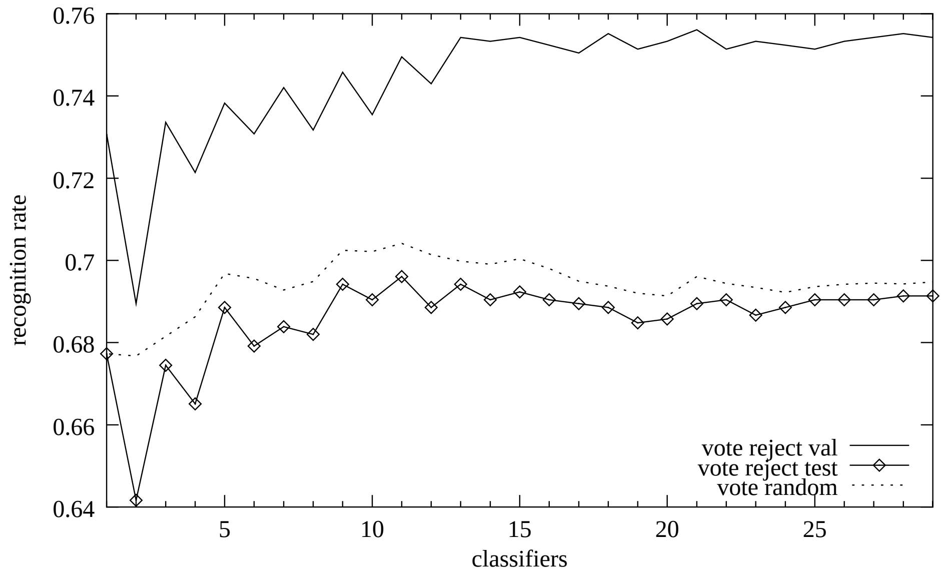


Figure A.35: Results of the *b. ensemble* method using objective function f_3

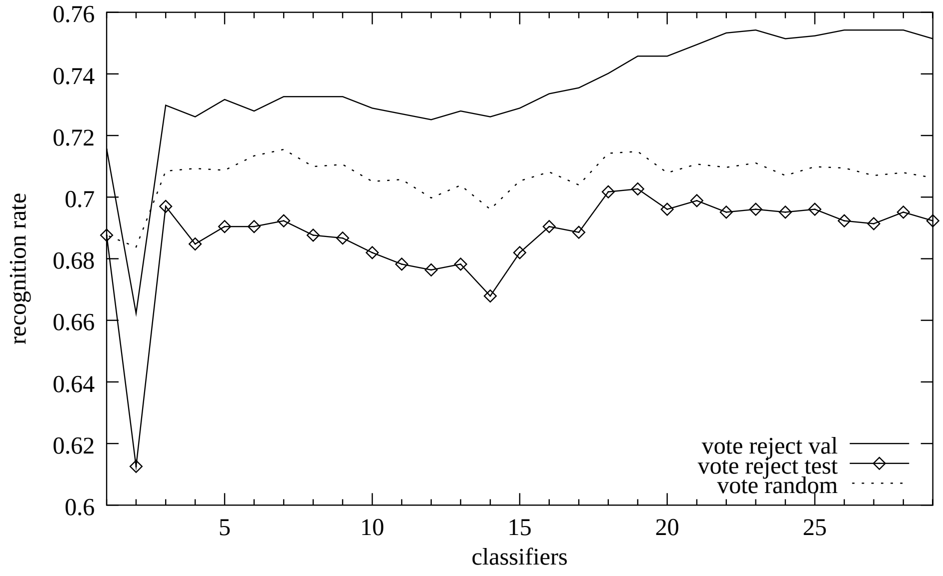


Figure A.36: Results of the *f. ensemble* method using objective function f_3

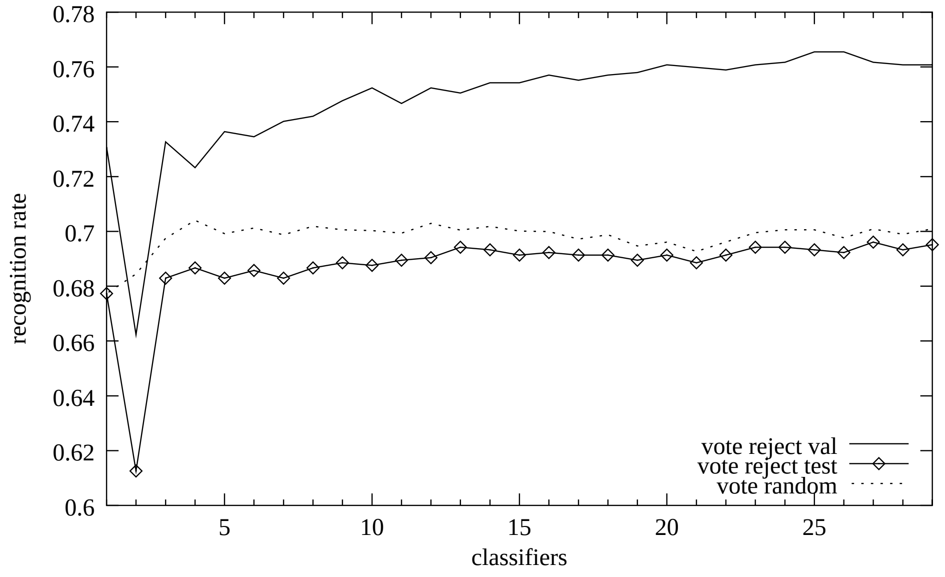


Figure A.37: Results of the *b./f. ensemble* method using objective function f_3

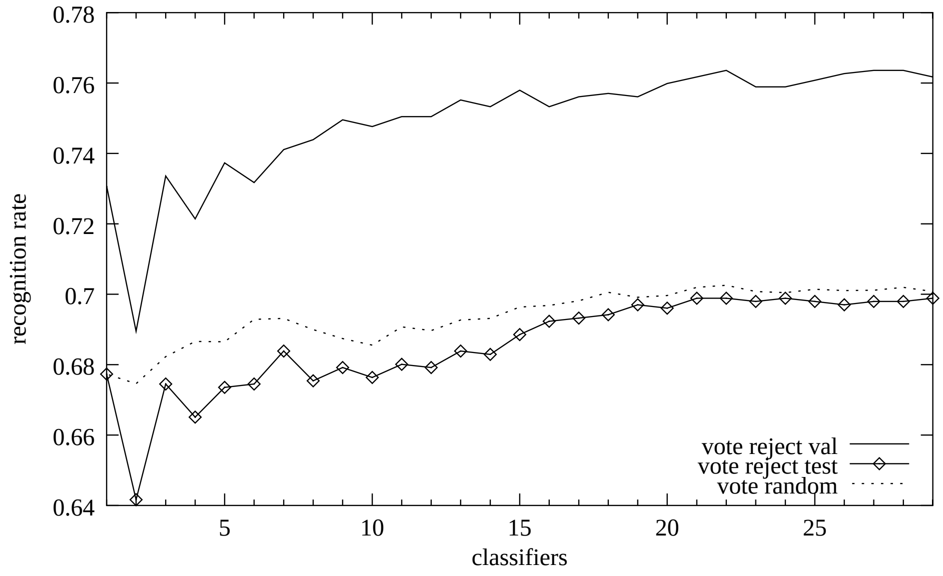


Figure A.38: Results of the *b. ensemble* method using objective function f_4

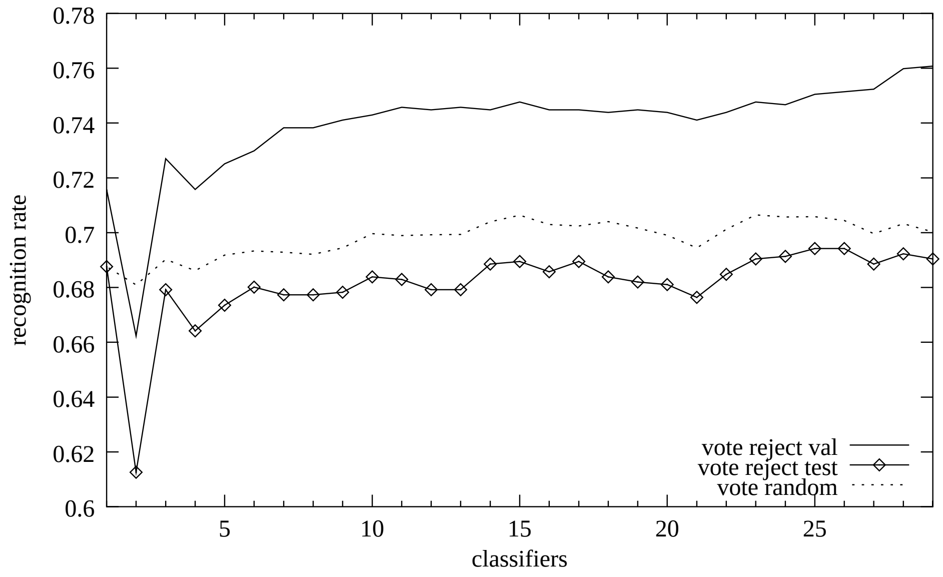


Figure A.39: Results of the *f. ensemble* method using objective function f_4

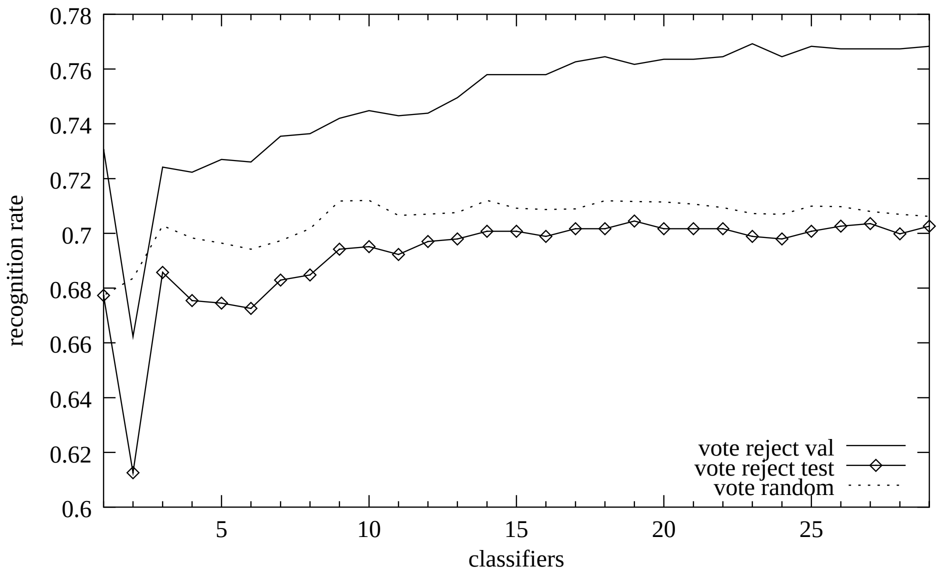


Figure A.40: Results of the *b./f. ensemble* method using objective function f_4

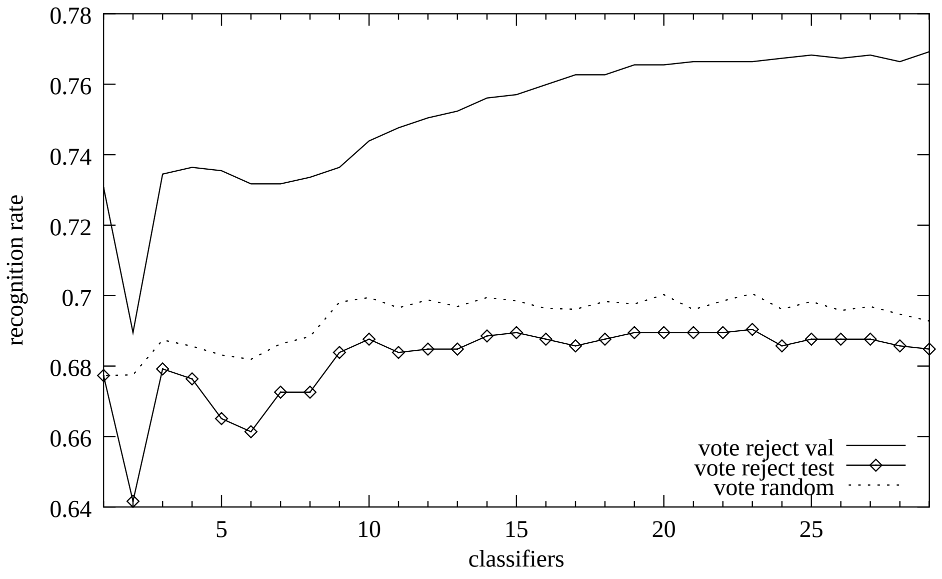


Figure A.41: Results of *std. exhaustive* using objective function f_1

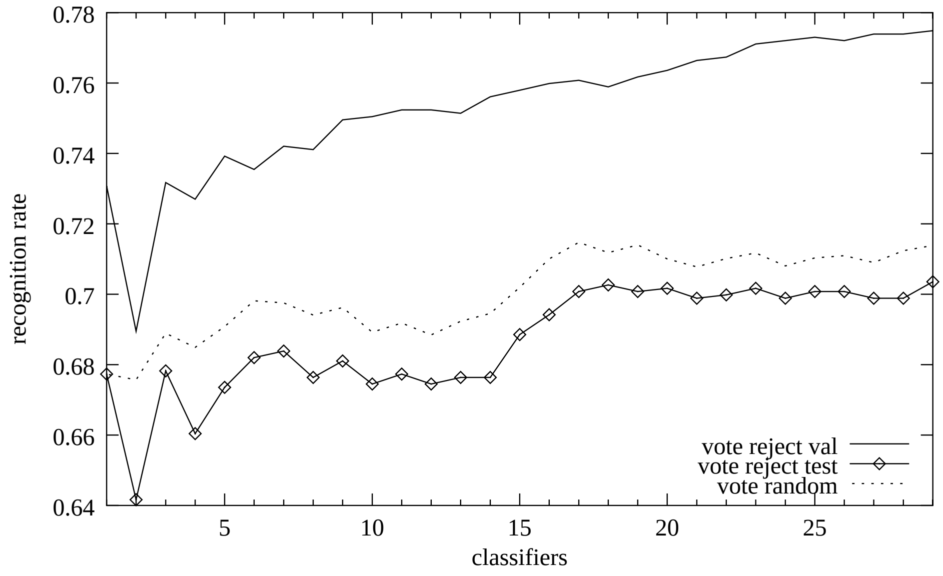


Figure A.42: Results of *std. exhaustive* using objective function f_2

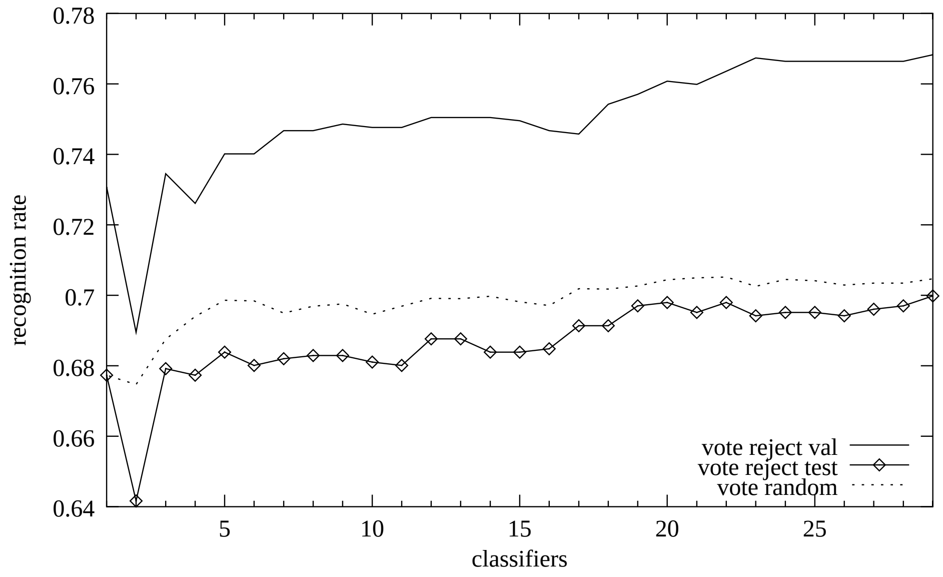


Figure A.43: Results of *std. exhaustive* using objective function f_3

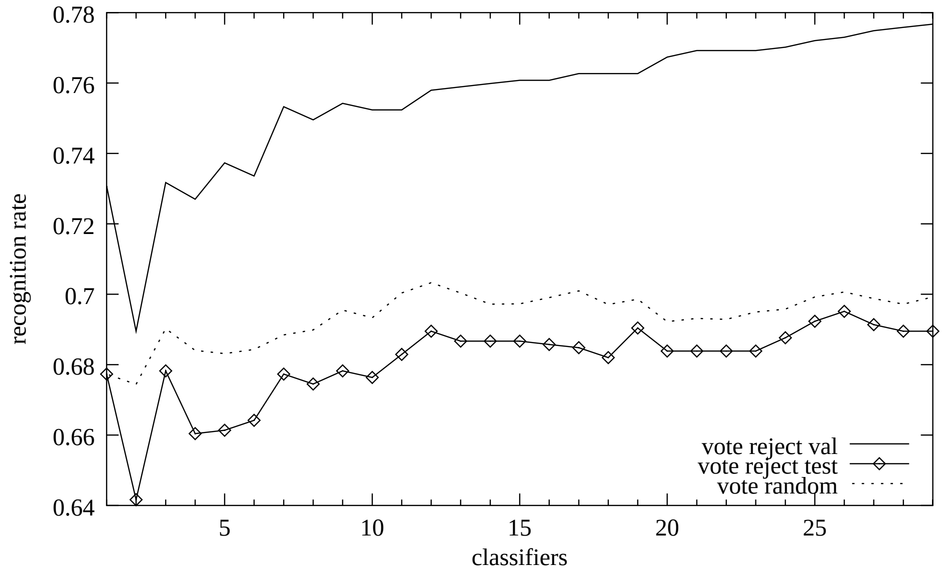


Figure A.44: Results of *std. exhaustive* using objective function f_4

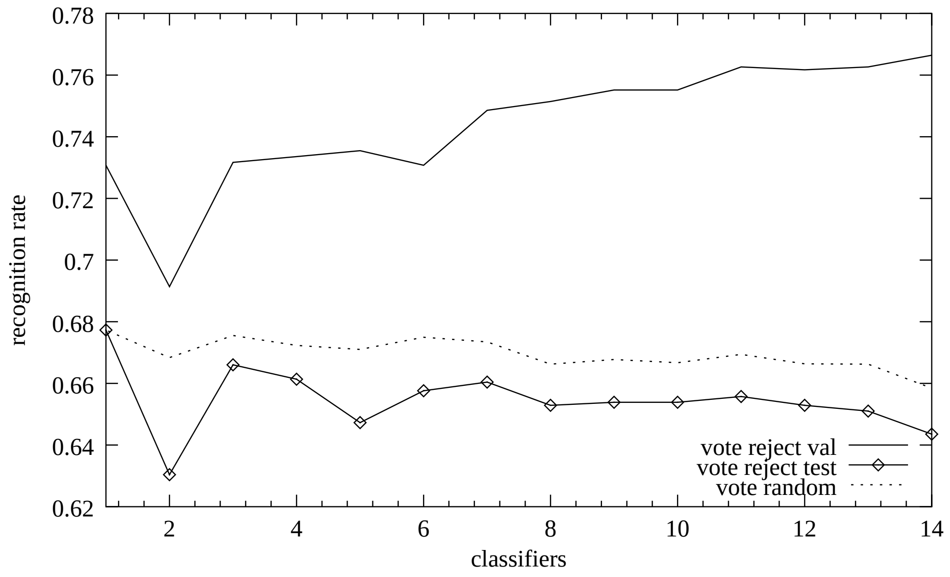


Figure A.45: Results of *replace exhaustive* using objective function f_1

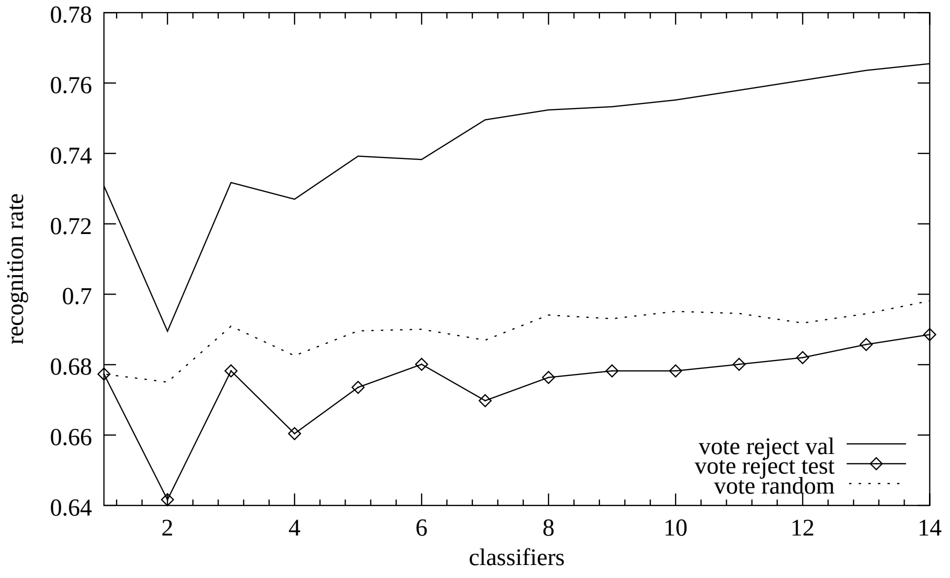


Figure A.46: Results of *replace exhaustive* using objective function f_2

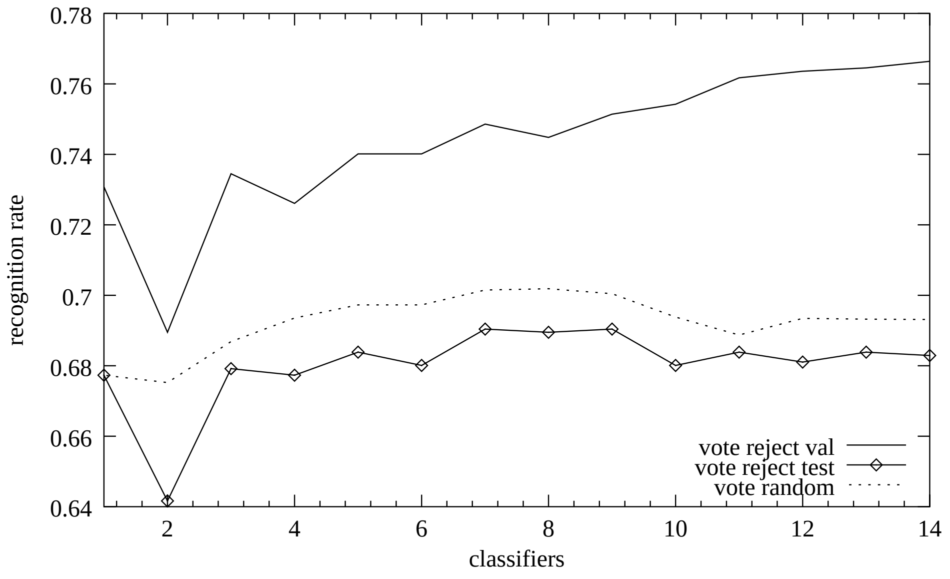


Figure A.47: Results of *replace exhaustive* using objective function f_3

Curriculum Vitae

1976	Born in Unterseen, Switzerland on August 9
1983 - 1987	Elementary school Ringgenberg
1987 - 1992	Secondary school Interlaken
1992 - 1996	Gymnasium Interlaken
1996 - 2000	Study of computer science at the University of Bern including secondary subjects Mathematics and Physics
1998	Participation in a student exchange program in the USA
2000	M. Sc. in computer science at the university of Bern
2001-2003	Research and lecture assistant and PhD student in the research group of Computer Vision and Artificial Intelligence of the University of Bern

Publications relating to diploma thesis

- H. Bunke, S. Günter, and X. Jiang. Towards bridging the gap between statistical and structural pattern recognition: Two new concepts in graph matching. In S. Singh, N. Murshed, and W. Kropatsch, editors, *Advances in Pattern Recognition - ICAPR 2001*, pages 1 - 11, 2001.

- S. Günter and H. Bunke. Validation indices for graph clustering. In J.-M. Jolion, W. Kropatsch, and M. Vento, editors, *Proc. of the 3rd IAPR-TC15 Workshop on Graph-based Representations in Pattern Recognition*, pages 229 - 238, 2001.

- S. Günter and H. Bunke. Weighted mean of a pair of graphs. *Computing*, 67 (3): 209-224, 2001.

- S. Günter and H. Bunke. Self-organizing map for clustering in the graph domain. *Pattern Recognition Letters*, 23: 401 - 417, 2002.

- H. Bunke and S. Günter. Adaptive self-organizing map in the graph domain. In H. Bunke and A. Kandel, editors, *Hybrid Methods in Pattern Recognition*, World Scientific, pages 61 - 74, 2002.

- S. Günter and H. Bunke. Validation indices for graph clustering. *Pattern Recognition Letters*, 24 (8): 1107-1113, 2003.

Publications relating to PhD thesis

- S. Günter and H. Bunke. Generating classifier ensembles from multiple prototypes and its application to handwriting recognition. In F. Roli and J. Kittler, editors, *Proc. of the 3rd Int. Workshop on Multiple Classifier Systems*, pages 179 - 188, Cagliari, Italy. Springer, 2002.

- S. Günter and H. Bunke. Creation of classifier ensembles for handwritten word recognition using feature selection algorithms. In *Proc. of the 8th Int. Workshop on Frontiers in Handwriting Recognition*, pages 183 - 188, Niagara-on-the-Lake, Canada, 2002.

- S. Günter and H. Bunke. A new combination scheme for HMM-based classifiers and its application to handwriting recognition. In *Proc. of the 16th Int. Conference on Pattern Recognition*, volume 2, pages 332 - 337, Quebec, Canada, 2002.

- S. Günter and H. Bunke. Ensembles of classifiers for handwritten word recognition. *International Journal of Document Analysis and Recognition*, 5(4): 224 - 232, 2003.

- S. Günter and H. Bunke. New boosting algorithms for classification problems with large number of classes applied to a handwritten word recognition task. In T. Windeatt and F. Roli, editors, *Proc. of the 4th Int. Workshop on Multiple Classifier Systems*, pages 326 - 335, Guildford, United Kingdom. Springer, 2003.

- S. Günter and H. Bunke. Fast feature selection in an HMM-based multiple classifier system for handwriting recognition. In B. Michaelis and G. Krell, editors, *Proc. of the 25th DAGM Symposium*, pages 289 - 296, Magdeburg, Germany. Springer, 2003.

- S. Günter and H. Bunke. Optimizing the number of states, training iterations and Gaussians in an HMM-based handwritten word recognizer. In *Proc. of the 7th Int. Conference on Document Analysis and Recognition*, volume 1, pages 472 - 476, Edinburgh, Scotland, 2003.

- S. Günter and H. Bunke. Off-line cursive handwriting recognition - On the influence of training set and vocabulary size in multiple classifier systems. In *Proc. of the 11th Conference of the International Graphonomics Society*, pages 196-199, Scottsdale, Arizona, USA, 2003.