Hierarchical Mixtures of Naive Bayes Classifiers

Marco A. Wiering
marco@cs.uu.nl
Intelligent Systems Group
Utrecht University

Abstract

Naive Bayes classifiers tend to perform very well on a large number of problem domains, although their representation power is quite limited compared to more sophisticated machine learning algorithms. In this paper we study combining multiple naive Bayes classifiers by using the hierarchical mixtures of experts system. This system, which we call hierarchical mixtures of naive Bayes classifiers, is compared to a simple naive Bayes classifier and to using bagging and boosting for combining multiple classifiers. Results on 19 data sets from the UCI repository indicate that the hierarchical mixtures architecture in general outperforms the other methods.

Keywords: Naive Bayes Classifiers, Hierarchical Mixtures of Experts, Bagging, Boosting, Machine Learning.

1 Introduction

Despite their simpleness, naive Bayes classifiers (Duda and Hart, 1973) in general obtain highly competitive results compared to decision trees (Quinlan, 1993), neural networks trained with backpropagation (Rumelhart et al., 1986), instance-based learning algorithms, and other inductive learning algorithms, see (Domingos and Pazzani, 1997) for a comparison study. Recently there is a lot of interest from the automatic text categorization community to use the naive Bayes classifier because of its advantages of learning speed, simpleness, memory usage, incrementality, and good results (McCallum et al., 1998). The naive Bayes classifier (NBC) works well on a wide range of problems, and is optimal when attributes are independent given the class. However, in real data sets, the independency assumption is often violated, but Domingos and Pazzani (1997) show that even if that is clearly the case, the naive Bayes classifier may still be optimal under the zero-one loss function. E.g. NBCs can optimally learn data sets described by conjunctions or disjunctions of literals, although these domains violate the independency assumption. However, the simple NBC learns a linear discriminant function and is therefore unable to learn linearly inseparable data such as the exclusive OR problem. Some approaches to overcome this problem combine attributes (Pazzani, 1996), but when there are many attributes, the algorithm needs to be executed many times, resulting in

slow learning in case multiple attributes need to be combined. Furthermore, combining too many attributes results in large representations and worse generalization performance. Instead, we opt for an algorithm which can deal with non-linearly separable data in a more principled way.

Hierarchical models. To solve the exclusive OR problem, we can use hierarchical architectures, just like linear networks have led to multi-layer perceptrons. Our current work is similar to the hierarchical mixtures of experts (HME) algorithm (Jacobs et al., 1991; Jordan and Jacobs, 1992). The HME architecture can consist of linear networks and is still able to learn non-linear functions. Instead of using linear networks as models, we use naive Bayes classifiers. Thus, we have an architecture consisting of gating NBCs which partition the data and weight the expert NBCs predicting the class probabilities. This results in a much more powerful classifier which is able to deal with non-linearly separable data.

Combining models. There exist a number of general algorithms which also learn multiple models (classifiers) and combine them to produce the final result. One method is *stacked generalization* (Wolpert, 1992) which combines induced models from the bottom layer to the top-layer, where independent model errors are used to select models for predicting the answer to a query. Stacked generalization can be seen as a meta-theory for combining models, but it is not entirely clear how it can be used for combining NBCs. Another algorithm is *bagging* (Breiman, 1996) which learns a set of independent models by first bootstrapping the data to get a training set and then inducing a new NBC on this data set. This is then repeated a number of times. The models are then combined by using majority voting of the predicted classes. This method can improve generalization performance, but does not lead to more powerful representations. Another method which receives a lot of attention is *boosting* (Freund and Schapire, 1996; Schapire et al., 1997) which sequentially induces a set of models where the data is reweighted after inducing each new classifier. This is done so that misclassified examples get higher weight in the training data for the next classifier. By combining multiple classifiers through voting, individual errors are corrected by the other classifiers. Some experiments (Bauer and Kohavi, 1999) have shown boosting to work better than bagging with NBCs (and also with decision trees) on a variety of data sets and to improve NBC classification accuracy substantially on a number of data sets from the UCI repository (Merz et al., 1997). A problem with these methods, however, is that the single NBCs still have to be able to learn the training data, which they cannot in case of the exclusive OR problem. Although boosting theory predicts that the training data can be perfectly loaded, it cannot perfectly load all data sets with NBCs (Bauer and Kohavi, 1999). Therefore, the additional representation power when using the hierarchical mixtures of NBCs can be beneficial for particular data sets.

Contents. In section 2, we describe naive Bayes classifiers (NBCs). In section 3, we describe hierarchical mixtures of NBCs. In section 4, we compare the single NBC to bagging, boosting and using the novel hierarchical mixtures of NBCs on 19 supervised data sets from the UCI repository. In section 5, we discuss related work. Finally, section 6 concludes this paper.

2 Naive Bayes Classifiers

Naive Bayes classifiers make an independency assumption to make full Bayesian learning feasible. A representation in which full dependency is modelled between the attributes would require an exponential amount of space to store and an exponential amount of time and data to learn. Other statistical learning algorithms use a set of independency relations to construct a compact Bayesian network (Heckerman et al., 1995), although exact inference is still an NP-hard problem (Dagum and Luby, 1993). Naive Bayes classifiers make a full independency statement and this makes them very fast to train and compact to store. This means that storing a large number of examples with many features becomes an easy task with such methods. Although the full independency assumption makes the model less powerful, NBCs still tend to perform very well on real world data sets. Domingos and Pazzani (1997) analyse why this is the case, and their findings are that although the bias (component of the error for an infinite sample) of NBCs is larger than the bias of more powerful learning algorithms, the variance (component of the error due to the sample's finite size) of NBCs is smaller. Since the variance decreases with a growing number of examples, NBCs may outperform other algorithms when the data sets are quite small. Furthermore, since the discriminant power of NBCs increases with a growing number of attributes, the NBC should be particularly favoured when the sample size is small and the number of attributes is large. These are also exactly the kind of problems for which more powerful inductive learners tend to overfit the data resulting in poor generalization performance.

2.1 Naive Bayes Classifiers

The learning problem is to map a set of features $D = \{f_1, f_2, \ldots, f_n\}$ describing an instance to its correct class-label C. For this the learning algorithm first induces a model (classifier) by learning on the training data $(D^1, C^1), (D^2, C^2), \ldots, (D^T, C^T)$.

Statistical learning algorithms perform the classification by first computing class probabilities $P(C|f_1, f_2, \ldots, f_n)$ of all output classes C given the input features, and then selecting the class with maximal probability. We cannot store these probabilities directly [1], since it would require an exponential amount of storage space and the result would not be useful for generalization. Instead, we first use Bayes' rule to compute:

$$P(C|f_1, f_2, \ldots, f_n) = \frac{P(f_1, f_2, \ldots, f_n|C)P(C)}{P(f_1, f_2, \ldots, f_n)}$$

and to decrease the size of this model we use the naive Bayes hypotheses of mutual independency among the features given the class:

$$P(f_1, f_2, \ldots, f_n|C) = \Pi_i \; P(f_i|C)$$

Now we have

$$P(C|f_1, f_2, \ldots, f_n) = \alpha P(C) \; \Pi_i \; P(f_i|C)$$

[1] This would resemble root learning.

where α is a normalization constant to sum all class probabilities given the features to 1.0. Basically the naive Bayes classifier can also be seen as a product network, where the bias is the class probability and weighted inputs are now modelled as features probabilities which are determined by a tabular representation. Thus, for nominal features the simple naive Bayes classifier can learn a linear decision boundary[2], and therefore has the same representational power as a perceptron.

2.2 Learning Algorithm

The learning algorithm is simple and uses a set of counters[3] to store all information. We define:

$$P(C) = \frac{c(C)}{tot}; \quad and \quad P_i(f_i|C) = \frac{c_i(f_i, C)}{c_i(C)}$$

To deal with the problem of having unobserved (feature-value, class) pairs in the training data, we use some parametrized Laplace correction. For this, we initialize the counters to some small value γ, and sum over them to get the totals. Now on each learning example $(\{f_1, f_2, \ldots, f_n\}, C^*)$, we use the following algorithm to update the parameters:

Updating NBC($\{f_1, f_2, \ldots, f_n\}, C^*, weight$):
1) $c(C^*)$ += $weight$
2) tot += $weight$
3) For all $k = 1 \ldots n$
3a) $c_k(f_k, C^*)$ += $weight$
3b) $c_k(C^*)$ += $weight$

Here the weight will be useful for defining the forthcoming algorithms. For the single naive Bayes classifier we use a weight of 1.0. Note that the algorithm is just using frequency counting, and a small prior (γ) is used to initialize the model.

3 Hierarchical Mixtures of Naive Bayes Classifiers

The hierarchical mixtures of experts system of Jordan and Jacobs (1992) consists of a number of gating networks and expert networks. The gating networks learn to gate the predictions of experts to the top layer network which makes the final prediction. We use the same system, but now we use naive Bayes classifiers (NBCs) instead of linear neural networks as gating and expert networks.

3.1 Architecture

We will explain a 2-layer architecture. Extensions to higher layer architectures are trivial. The system consists of 1 root gating NBC m_0, N gating NBCs m_1^1

[2]For numeric attributes decision boundaries could be non-linear due to the discretization method used.

[3]The counter variables $c(C)$ etc. are represented as real numbers.

to m_1^N, and $N \times M$ expert networks m_2^{11} to m_2^{NM}. Have a look at figure 1 which depicts a two-layer architecture in which the gating networks have two sub-models (children).

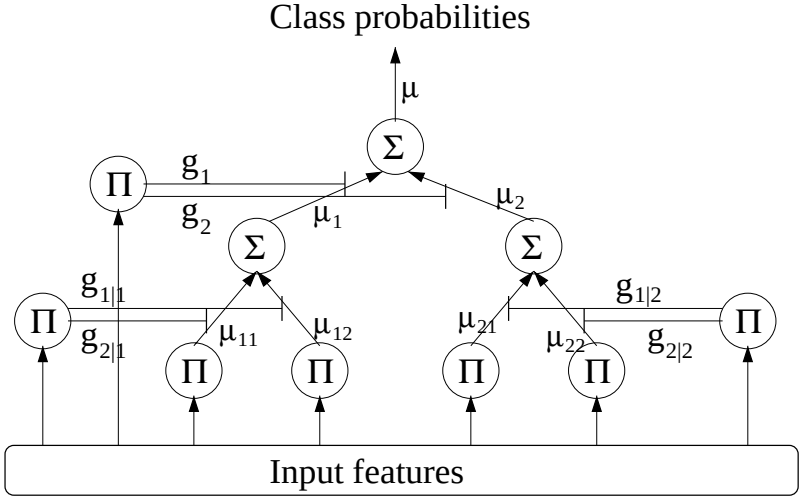

Figure 1: *The 2-layer architecture consisting of naive Bayes classifiers (shown by the product signs). The gating networks weight the outputs of their sub-models and propagate the weighted sum to the gating network one layer above. Expert networks estimate the class probabilities μ_{ij} given the features.*

Expert networks m_2^{ij} output class probabilities given the input features describing the instance D. The class probabilities can be modelled as a vector $\vec{\mu_{ij}} = (\mu_{ij}^{C_1}, \mu_{ij}^{C_2}, \ldots, \mu_{ij}^{C_K})$, where:

$$\mu_{ij}^C = P_2^{ij}(C|D) = \alpha P_2^{ij}(C)\Pi_k P_2^{ij}(f_k|C)$$

Here α is again a renormalization constant. The top-layer gating network m_0 computes the following gating values for its sub-models M_i:

$$g_i = P_0(M_i|D) = \alpha P_0(M_i)\Pi_k P_0(f_k|M_i)$$

and gating networks m_1^i compute output gating values by:

$$g_{j|i} = P_1^i(M_j|D) = \alpha P_1^i(M_j)\Pi_k P_1^i(f_k|M_j)$$

So the gating networks essentially treat the expert networks as classes.

Our architecture now consists of counters for all models. For model m_2^{ij} we use tot_2^{ij} etc. as counter variables. The complete model should be initialized with some symmetry breaking counter generator (e.g. by adding a small random value to the initialization value γ).

We want to compute the class probabilities of the root network given the input data $D = \{f_1, f_2, \ldots, f_N\}$. For this we have to compute class probabilities

by propagating the predictions of the experts to the top. The output of the complete architecture can be computed as:

$$\vec{\mu} = \sum_i g_i \sum_j g_{j|i} \vec{\mu_{ij}}$$

For training this system, the gating networks have to predict how well their sub-models perform given some input data, and let the gating weight of the best model converge to the highest value among the models.

3.2 Learning by Expectation Maximization

Expectation Maximization (Dempster et al., 1977) is a well known method for multiple model fitting in which mixture coefficients of the local mixture models are learned. The weights for selecting each model are latent variables, since they cannot be estimated directly from the data. Instead a couple of iterations can be performed in which the latent variables can be estimated by monitoring the error of individual models. The total probability of generating the output class probabilities is computed by mixing the expert class probabilities through the gating networks given the parameters:

$$P_0(C|D) = \sum_i g_i \sum_j g_{j|i} P_2^{ij}(C|D)$$

Posterior probabilities. To develop the learning algorithm, we need to compute posterior probabilities that each model has generated the right output class C^*. For this we compute:

$$h_i = \frac{g_i \sum_j g_{j|i} \hat{P_2^{ij}}(C^*|D)}{\sum_i g_i \sum_j g_{j|i} \hat{P_2^{ij}}(C^*|D)}$$

and

$$h_{j|i} = \frac{g_{j|i} \hat{P_2^{ij}}(C^*|D)}{\sum_j g_{j|i} P_2^{ij}(\hat{C^*}|D)}$$

where we use a Gaussian regression model for computing the probability that expert network m_2^{ij} generated C^*:

$$\hat{P_2^{ij}}(C^*|D) = e^{-\sigma(1.0 - P_2^{ij}(C^*|D))}$$

We could also have used other distributions such as the Bernoulli distribution, but selected the Gaussian regression model due to its general applicability to multiple classes. Furthermore, using this model gives us more influence to control the learning speed in which models start to deviate from each other. For this we can set σ which in our experiments was set to a small value (0.1) so that one model would not immediately learn much faster on data of a particular class (for which the local model may have learned a higher a-priori class probability).

We first compute the posterior values (Expectation step), and then we update the gating models so that the best model will get a higher weight on the

example, and we update the class probabilities of experts to the real class according to their posterior probabilities (Maximization step). We will not use gradient descent learning here, since we expect it to learn slow due to the product networks (the gradient would be computed by multiplying all $P(f_i|C)$ values, some of which may be very small). Instead we use the naive Bayes classifier update scheme. Note that we perform the EM step after each example, thus we have an online stochastic learning algorithm. Also, since we use the naive Bayes classifier, the algorithm does not really maximize the probability of generating the correct class label, but rather makes a small step to increase this probability. The algorithm is therefore a generalized EM or GEM algorithm (Jordan and Jacobs, 1992).

Updating the expert models. After having computed the class probabilities for each model and having computed the posterior probabilities for all models (except the root model), we can adapt the models.

We update the expert networks using the NBC updating scheme. Here expert network m_2^{ij} has parameters

$$P_2^{ij}(C) = \frac{c_2^{ij}(C)}{tot_2^{ij}}; \quad and \quad P_{2k}^{ij}(f_k|C) = \frac{c_{2k}^{ij}(f_k, C)}{c_{2k}^{ij}(C)}$$

We update the counter variables given an example $X = (D, C^*)$ by using the NBC updating scheme. To do this we call **Update-NBC**$(D, C^*, h_i h_{j|i})$ for each model m_2^{ij}. Thus, the weight of the update equals the posterior probability that the expert network could have generated the correct class. Updating in this way, causes expert networks with the largest posterior probability ($h_i h_{j|i}$) to learn the example fastest and to bias its function more to this example. All expert networks learn on each example.

Updating the gating models. For updating the gating networks, we make use of the best predictive sub-model as the desired output of the classifier, so that the update causes this model to be selected with a higher probability. The best sub-model M_b has the largest probability of generating C^*. For the top-layer model we update the model parameters by calling: **Update-NBC**$(D, M_b, 1.0)$. Thus, the best sub-model is now the correct class, and the weight of updating towards this model on this example is 1.

For the sub-gating networks, we multiply the learning weight of 1.0 by the posteriori probability h_i to obtain the learning weight. We again compute the best sub-model of each sub-gating network, and call this M_b. Then we update the parameters of model m_1^i by calling: **Update-NBC**(D, M_b, h_i).

Solving the exclusive OR problem. Before running experiments on real world data sets, we first analysed whether the hierarchical mixtures of NBCs was able to learn the exclusive OR problem and the 4-bit parity problem. Learning the exclusive OR problem was no problem at all for a one layer architecture — it was always able to load the training patterns. For learning the 4-bit parity problem, we had to use higher layer architectures. The smallest architecture which can represent the 4-bit parity problem is a 4-layer architecture in which there are always two submodels for each gating network. However, this minimal model could not learn the 4-bit parity problem with the parameters we used.

Then we tried 6 and 7-layer architectures, and these were able to reliably learn the 4-bit parity problem. Thus, these experiments showed that the hierarchical mixtures of NBCs is able to learn non-linearly separable data sets.

4 Experiments

We have tested the hierarchical mixtures of naive Bayes classifiers on 19 data sets from the UCI repository. We preprocessed continuous (and nominal data with large values) by using the mean and standard deviance and computing significance classes using 1 standard deviation as a separator between two feature values.[4] The data sets are given below in table 1.

Data Set	**Nr. of Classes**	**Nr. of Features**	**Nr. of Instances**
Abalone	14	9	4177
W. Breast Cancer	2	9	699
Car	4	6	1728
Chess kr-vs-kp	2	36	3196
Contraceptive	3	9	1473
Ecoli	8	7	336
Glass	8	9	214
Hepatitis	2	19	155
Housing	5	13	506
Ionosphere	2	34	351
Iris	3	4	151
Liver Bupa	2	6	345
Pima Indians	2	8	768
Segmentation	7	19	210
Servo	5	4	167
Soybeans	20	35	675
Spam	2	57	4601
Vote	2	16	435
Yeast	10	8	1484

Table 1: *The nineteen data sets from the UCI repository.*

Experimental setup. We compare the hierarchical mixtures of naive Bayes classifiers (HM) to the simple naive Bayes classifier, bagging and boosting. For the HM architectures, we used a single layer architecture consisting of 4 expert networks, and a 2-layer architecture consisting of 2×2 expert networks. We performed experiments with bagging and boosting in which the number of models was 10. We did not explore whether using more models (e.g.

[4]Comparing our results to published results of possibly better approaches for discretizing the features such as entropy-based discretization, shows that there is usually only a slight decrease in learning performance. See also the comparison study in (Domingos and Pazzani, 1997). This will not affect our current comparison study, however.

50) worked better for two reasons: (1) The hierarchical systems would be much smaller and therefore faster to use, (2) We also did not use results of larger hierarchical architectures which may sometimes have worked better. Finally, we expect the sign of significant differences between the methods to remain the same in case a much larger number of models would have been used.

Our algorithms for bagging and boosting were similar to the ones described in (Bauer and Kohavi, 1999), but differed in some aspects. For bagging, we did not always bootstrap a new data set which was equal in size to the original data set. We rather experimented with using percentages of the original data set, and found that sometimes bagging worked best when only 20% was used for each data set. Most often, however, we used about 90% of the original data set size for bootstrapping (with replacement).

For boosting, Bauer and Kohavi (1999) used bootstrapping on the original data set in case the error of a classifier was larger than 50%. Instead, we reweighted the original data set with values between 0.9 and 1.1, and used the new reweighted data set for learning the next classifier. Thus instead of resampling we used reweighting, which should not differ a lot.

We performed 50 simulations per data set in which always half of the data set was used for learning and the other half was used for testing. We used 5 EM iterations for each hierarchical system, in which during 1 iteration the complete training data was learned in an online fashion. We kept all learning parameters constant for all data sets: $\gamma = 0.1 + rand(0, 0.01)$, $\sigma = 0.1$.

Test results. Table 2 shows the test results on the 19 data sets. The table indicates the percentages of correct classifications with the standard deviance, and significance of the results. Here (++) indicates a significant improvement (t-test, $p < 0.01$) and (+) a significant improvement ($p < 0.05$) compared to the simple NBC. The win-loss row indicates how often the mixtures of NBC, bagging or boosting significantly ($p < 0.05$) work better or worse than the simple naive Bayes classifier. The average error reduction (Bauer and Kohavi, 1999) is computed by first computing the error reduction $\frac{(e_a - e_b)}{e_a}$, where e_a is the error of the simple NBC, for each data set and then computing the average.

The results show that the hierarchical mixtures of NBCs significantly outperform the simple NBC on 9 data sets and loses on 2 data sets. Furthermore, they increase the average accuracy with more than 1%, and reduce the average error with about 7%. Although the differences may seem quite small, they are significant, and for some data sets the simple NBC already seems to reach the highest possible test performance[5], so that it is difficult to improve on this. However, for particular data sets the improvements are quite large and for some of these data sets we found that larger HM architectures even worked better.

When we examine bagging, we can see that it sometimes works better than the NBC, but as many times works worse (especially for data sets with few features), so there is no real improvement in combining bagging with NBCs in general. This can be expected, since NBCs are quite stable classifiers, so that combining multiple classifiers is not so effective, and can sometimes even reduce

[5] In other comparison studies with other learning algorithms, there also seems to be the same maximal accuracy for these particular data sets.

Data Set	**NBC**	**1-4 HM**	**2-2 HM**	**Bagging**	**Boosting**
Abalone	68.6±1.2	$71.8 \pm 1.3^{++}$	$71.7 \pm 1.0^{++}$	$68.9 \pm 1.2^{=}$	$68.5 \pm 1.5^{=}$
Breast Cancer	97.2±0.6	$97.0 \pm 0.7^{=}$	$96.6 \pm 0.8^{--}$	$97.3 \pm 0.7^{=}$	$95.8 \pm 0.9^{--}$
Car	84.8±1.6	$89.4 \pm 1.2^{++}$	$88.3 \pm 1.6^{++}$	$83.3 \pm 1.6^{--}$	$89.9 \pm 1.2^{++}$
Chess	87.1±1.1	$91.6 \pm 1.8^{++}$	$92.7 \pm 1.7^{++}$	$87.2 \pm 1.5^{=}$	$94.5 \pm 0.8^{++}$
Contraceptive	51.4±1.2	$51.8 \pm 1.4^{=}$	$51.5 \pm 1.5^{=}$	$50.9 \pm 1.6^{=}$	$51.0 \pm 1.5^{=}$
Ecoli	73.8±2.8	$73.1 \pm 3.8^{=}$	$73.5 \pm 3.5^{=}$	$73.8 \pm 3.2^{=}$	$73.3 \pm 3.2^{=}$
Glass	48.5±5.1	$51.0 \pm 5.3^{+}$	$51.9 \pm 5.2^{++}$	$50.9 \pm 4.9^{+}$	$51.0 \pm 5.7^{+}$
Hepatitis	85.5±2.8	$83.2 \pm 3.6^{--}$	$82.8 \pm 3.5^{--}$	$84.4 \pm 3.2^{=}$	$82.2 \pm 3.6^{--}$
Housing	59.3±2.3	$63.5 \pm 3.8^{++}$	$67.7 \pm 2.5^{++}$	$61.4 \pm 3.5^{++}$	$59.7 \pm 2.7^{=}$
Ionosphere	90.0±1.8	$91.3 \pm 1.4^{++}$	$91.0 \pm 2.2^{+}$	$90.1 \pm 1.5^{=}$	$90.2 \pm 2.3^{=}$
Iris	90.2±3.5	$90.1 \pm 2.9^{=}$	$90.1 \pm 3.5^{=}$	$89.2 \pm 2.6^{=}$	$90.0 \pm 2.4^{=}$
Liver Bupa	60.0±3.0	$60.8 \pm 3.0^{=}$	$60.3 \pm 3.1^{=}$	$58.4 \pm 2.9^{--}$	$60.5 \pm 3.1^{=}$
Pima Indians	75.0±1.4	$74.2 \pm 2.3^{-}$	$75.0 \pm 1.6^{=}$	$75.2 \pm 2.0^{=}$	$73.3 \pm 2.1^{--}$
Segmentation	78.7±4.0	$79.3 \pm 5.6^{=}$	$79.7 \pm 6.4^{=}$	$78.6 \pm 4.8^{=}$	$77.8 \pm 5.4^{=}$
Servo	82.3±4.2	$83.0 \pm 3.8^{=}$	$82.1 \pm 3.3^{=}$	$80.2 \pm 4.9^{-}$	$82.6 \pm 3.7^{=}$
Soybeans	89.5±2.2	$91.6 \pm 2.4^{++}$	$91.5 \pm 2.6^{++}$	$90.1 \pm 1.9^{=}$	$91.3 \pm 1.9^{++}$
Spam	83.3±0.6	$84.4 \pm 0.5^{++}$	$84.5 \pm 0.6^{++}$	$84.4 \pm 0.5^{++}$	$82.7 \pm 0.6^{--}$
Vote	90.6±1.7	$92.7 \pm 1.7^{++}$	$93.3 \pm 2.3^{++}$	$90.4 \pm 1.5^{=}$	$94.1 \pm 1.5^{++}$
Yeast	56.6±1.1	$57.1 \pm 1.4^{=}$	$57.0 \pm 1.3^{=}$	$56.2 \pm 1.5^{=}$	$56.5 \pm 1.5^{=}$
Average :	76.4	77.7	77.9	76.4	77.1
Av. error red.	-	7.0	6.9	-1.0	3.4
Sign. Win-loss :	-	9 : 2	9 : 2	3 : 3	5 : 4

Table 2: *The Training results on the 19 data sets.*

learning performance since less data of the original data set may be effectively used for learning each classifier.

Boosting outperforms the NBC significantly in a number of domains such as Car, Chess, and Vote[6], but on many other data sets does not lead to an improvement. In some domains, boosting even results in a larger error. We have found that this is caused by 2 problems: (1) Boosting sometimes leads to overfitting the data, where the training data is perfectly loaded, but generalization performance is reduced (which happened for e.g. Breast Cancer and Hepatitis), (2) Boosting has problems with some domains such as Spam, because after inducing 1 classifier, the next one always has a weighted error sum larger than 50% on the reweighted data. Therefore, boosting on such data sets does not lead to a collaboration between voting classifiers, but stand-alone classifiers are learned.

If we look at the domains in which boosting outperforms the simple NBC, we observe that the hierarchical mixtures systems also outperform the simple NBC. This is remarkable and is probably caused by the fact that for these domains a smaller error on the training data also means a smaller error on the test data. Since boosting and the hierarchical mixtures always reduce the training error

[6]A comparison with other published results shows that decision trees often outperform NBCs in these domains.

compared to the simple NBC, their generalization performance depends on the actual domain (and the limited training data we used). Boosting improves the average accuracy, but performs on average less well than the hierarchical mixtures systems.

We also experimented with boosting hierarchical mixtures of NBCs. Although, for some domains this worked very well, the average accuracy for all data sets was the same as for the hierarchical mixtures of NBCs alone. Finally, we also tried to learn the gating values after a number of expert networks were learned by boosting. Since boosting weights each induced classifier by their average error, this does not indicate for what kind of data the classifier works well. Learning to weight these classifiers for each example might therefore be useful. The preliminary experiments indicated that using the hierarchical mixtures of NBCs after learning each classifier by boosting, did not result in improved average performance compared to boosting, however.

5 Related Work

There have been a number of approaches to extend the naive Bayes classifier or to combine models. Domingos (2000) describes Bayesian model averaging, where first a set of classifiers are induced, and then weights for combining the models are estimated by computing the error probability of each classifier. This method does not use different weights for different examples, however. The experiments showed that this often led to overfitting the data.

Bauer and Kohavi (1999) used the NBC and combined it with bagging, boosting and some variants such as arcing (Breiman, 1998). They showed that bagging NBCs could slightly improve the results on the data sets they used, and that boosting NBCs significantly reduced the test error. Our experiments show much less advantage for using bagging and boosting, but this may be caused by the fact that Bauer and Kohavi used different data sets with much more examples (all data sets they used had at least 1000 examples). Furthermore, naive Bayes classifiers are stable learning algorithms, and that is why we cannot expect a great benefit from using bagging. We also found that boosting sometimes leads to overfitting the data, where the algorithm could perfectly load the training data, but an increase in test error occurred.

Kohavi (1996) studies using decision trees with naive Bayes classifiers at the leave nodes (NBTree). The experiments showed that the combination worked better than either algorithm alone. Ting and Zheng (1999) also combined decision tree learning with naive Bayes classifiers at the leave nodes, but found that inducing trees with more than one node, worked less well than the simple NBC alone. Then they applied boosting to the NBC and to NBTree, and found that boosting NBCs did not result in any improvement of the average accuracy over all data sets they used. Boosting NBTree worked very well, however, and significantly outperformed the simple NBC. They explain these results by the fact that NBTree increases instability (the bias is smaller and the variance is larger) so that boosting may result in better performance. It would be interesting to compare the boosted NBTree to the hierarchical mixtures of NBCs

described in this paper, or to combine both algorithms.

Zheng (1998) uses a committee of naive Bayes classifiers in which each different NBC has a different subset of attributes. His method selects attributes so that attributes used by one classifier which performs well on the data set are also used with higher probability by the next classifier. The results show that the committee can significantly outperform the simple NBC on particular data sets from the UCI repository. These committees cannot learn to classify non-separable data sets, however.

Zheng, Webb and Ting (1999) developed lazy Bayesian rules, a classifier system which evaluates test examples in a lazy way. Instead of building a general classifier on the training data, the training data is stored in memory, and if an example needs to be classified a new classifier is constructed. This is done by using a conjunctive rule on attribute values. Different conjunctive rules are constructed and from the training data which obey the rule, a naive Bayes classifier in constructed. To choose among the possible conjunctive rules, N-fold cross validation is used. The lazy Bayesian rules system is shown to outperform the simple NBC and performs on average as well as boosting decision trees. Since for each test example, a new classifier should be induced, the method uses more computation time, however, than boosting 100 decision trees in case many test examples need to be classified.

McCallum et al. (1998) use a hierarchical model of NBCs for text classification problems. The hierarchy which was used came from the used internet provider (e.g. Yahoo), and a form of expectation maximization was used to fit a set of mixture coefficients to select sub-models responsible for generating a document. Furthermore, they used shrinkage as a statistical technique to deal with expert NBCs which receive only few examples. The experiments on three real-world data sets showed improved performance compared to the simple naive Bayes classifier.

Stewart (1998) developed an algorithm which includes hidden variables to the naive Bayes classifier. The latent variables are learned by a maximum likelihood algorithm, but he does not use hidden variables to select or combine models. Instead, the hidden variables are used to approximate the joint distribution of a set of variables. This method outperforms the simple NBC on some data sets from the UCI repository.

Meila and Jordan (2000) describe an algorithm which learns mixture coefficients for combining a set of tree distributions. Tree distributions (Chow and Liu, 1968) are special cases of graphical models in which both parameter and structure learning are tractable. The mixture-of-trees model provides an effective generalization of tree distributions in which different dependencies between the variables can be modelled by different trees. Like graphical models, this method can be used for density estimation and classification, but due to its wider applicability, the mixture coefficients were not conditioned on the input of an example, which may contain many unknown values.[7] The experiments

[7]Note that the hierarchical mixtures of NBCs still works if particular features values are unknown. In general NBCs are quite robust against missing values, which are usually just not used in the classification and learning process.

show that the algorithm outperforms a large number of other algorithms such as the hierarchical mixtures of experts and the simple NBC.

6 Conclusion

We introduced the hierarchical mixtures of naive Bayes classifiers which is based on the hierarchical mixtures of experts system where all networks are naive Bayes classifiers. We have shown that the hierarchical extension can learn to classify non linearly separable data, which a simple naive Bayes classifier cannot. In the experiments we compared the novel hierarchical system to the flat naive Bayes classier and two other techniques for combining multiple classifiers — bagging and boosting. The experimental results on 19 data sets from the UCI repository show that the hierarchical mixtures of naive Bayes classifiers in general outperforms the simple naive Bayes classifier, and also achieves better average results than bagging and boosting with naive Bayes classifiers. In our current work, the hierarchical architecture had to be designed a-priori. In future work we want to study growing architectures online using cross-validation to test the appropriateness of an architecture. In this way we want to circumvent using architectures which can underfit or overfit the learning data and thus perform poorly on the test data. Finally, we want to combine variants of the HME architecture with other algorithms such as decision trees, K-nearest neighbors, locally weighted regression, and support vector machines.

References

Bauer, E. and Kohavi, R. (1999). An empirical comparision of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning*, 36:105 – 142.

Breiman, L. (1996). Bagging predictors. *Machine Learning*, 24(2):123–140.

Breiman, L. (1998). Arcing classiers. *The annals of statistics*, 26:801–849.

Chow, C. and Liu, C. (1968). Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory, IT-14*, 3:462–467.

Dagum, P. and Luby, M. (1993). Approximating probabilistic inference in bayesian belief networks is NP–hard. *Artificial Intelligence*, 60:141–153.

Dempster, A., Laird, N., and Rubin, D. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series "B"*, 39:1–38.

Domingos, P. (2000). Bayesian averaging of classifiers and the overfitting problem. In Langley, P., editor, *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 223–230. San Francisco, CA: Morgan Kaufmann.

Domingos, P. and Pazzani, M. J. (1997). On the optimality of the simple Bayesian classifier under zero-one loss. *Machine Learning*, 29(2-3):103–130.

Duda, R. and Hart, P. (1973). *Pattern classification and scene analysis*. New York: John Wiley and Sons.

Freund, Y. and Schapire, R. E. (1996). Experiments with a new boosting algorithm. In *Proceedings of the thirteenth International Conference on Machine Learning*, pages 148–156. Morgan Kaufmann.

Heckerman, D., Geiger, D., and Chickering, D. (1995). Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20:197–243.

Jacobs, R. A., Jordan, M. I., Nowlan, S. J., and Hinton, G. E. (1991). Adaptive mixtures of local experts. *Neural Computation*, 3(1):79–87.

Jordan, M. I. and Jacobs, R. A. (1992). Hierarchies of adaptive experts. In Moody, J. E., Hanson, S. J., and Lippmann, R. P., editors, *Advances in Neural Information Processing Systems 4*, pages 985–993. Morgan Kauffmann.

Kohavi, R. (1996). Scaling up the accuracy of Naive-Bayes classifiers: a decision-tree hybrid. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, pages 202–207.

McCallum, A., Rosenfeld, R., Mitchell, T., and Ng, A. (1998). Improving text classification by shrinkage in a hierarchy of classes. In *Proceedings of the 1998 International Conference on Machine Learning*.

Meila, M. and Jordan, M. (2000). Learning with mixtures of trees. *Journal of Machine Learning Research*, 1:1–48.

Merz, C., Murphy, P., and Aha, D. (1997). UCI repository of machinelearning databases.

Pazzani, M. (1996). Searching for dependencies in bayesian classifiers. In Fisher, D. and Lenz, H., editors, *Learning from data: Artificial intelligence and statistics V*, pages 239–248.

Quinlan, J. (1993). *C4.5 Programs for machine learning*. San Mateo, CA: Morgan Kaufmann.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning internal representations by error propagation. In *Parallel Distributed Processing*, volume 1, pages 318–362. MIT Press.

Schapire, R. E., Freund, Y., Bartlett, P., and Lee, W. S. (1997). Boosting the margin: a new explanation for the effectiveness of voting methods. In *Proceedings of the fourteenth International Conference on Machine Learning*, pages 322–330. Morgan Kaufmann.

Stewart, B. (1998). Improving performance of naive bayes classifiers by including hidden variables. In Mira, J. and Pobil, A. D., editors, *Methodology and Tools in Knowledge-Based Systems, 11th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, IEA/AIE-98, Volume I. Lecture Notes in Computer Science, Vol. 1415.*, pages 272–280. Springer.

Ting, K. M. and Zheng, Z. (1999). Improving the performance of boosting for naive bayesian classification. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 296–305.

Wolpert, D. (1992). Stacked generalization. *Neural Networks*, 5:241–259.

Zheng, Z. (1998). Naive bayesian classifier committees. In *Proceedings of the tenth European Conference on Machine Learning*, pages 196–207. Berlin: Springer-Verlag.

Zheng, Z., Webb, G. I., and Ting, K. M. (1999). Lazy Bayesian rules: a lazy semi-naive Bayesian learning technique competitive to boosting decision trees. In *Proc. 16th International Conf. on Machine Learning*, pages 493–502. Morgan Kaufmann, San Francisco, CA.