Hierarchical Mixtures of Naive Bayesian Classifiers

Marco A. Wiering

Intelligent Systems Group
Institute of Information and Computing Sciences
Utrecht University
Padualaan 14, 3508TB Utrecht, The Netherlands
marco@cs.uu.nl

Abstract. Naive Bayesian 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 Bayesian classifiers by using the hierarchical mixtures of experts system. This novel system, which we call hierarchical mixtures of naive Bayesian classifiers, is compared to a simple naive Bayesian 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.

1 Introduction

Despite their simplicity, naive Bayesian classifiers [9] in general obtain highly competitive results compared to decision trees, neural networks trained with backpropagation, instance-based learning algorithms, and other inductive learning algorithms, see [8] for a comparison study. The naive Bayesian classifier (NBC) works well on a wide range of problems with discrete and nominal data,[1] and is optimal when attributes are independent given the class. However, in real data sets, the independency assumption is often violated. Furthermore, 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 [16], 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.

[1] For continuous attributes, the data can be preprocessed which often leads to better results.

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 [11]. The HME architecture can consist of linear networks and is still able to learn non-linear functions.[2] Instead of using linear neural networks as models, we use naive Bayesian classifiers. Thus, we have an architecture consisting of gating NBCs which partition the data and weight the expert NBCs predicting the class probabilities. E.g., in text categorization one gating NBC could learn to discriminate between newsgroups such as politics and religion and an expert NBC could then discriminate between politics-guns and politics-misc etc. Thus the hierarchical mixtures of NBCs 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 algorithm is *bagging* [4] which learns a set of independent models by first bootstrapping the data to get a separate 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. Another method which receives a lot of attention is *boosting* [10, 17] 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. 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. 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 Bayesian 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. Section 5 describes related work. Finally, section 6 concludes this paper.

2 Naive Bayesian Classifiers

Naive Bayesian 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 although exact inference is still a NP-hard problem for the most general Bayesian networks. NBCs make a full independency statement which makes them very fast to train and compact to store.

[2] The HME method can also be combined with nonlinear classifiers (see e.g., [2]).

2.1 Naive Bayesian 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 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, 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, and get:

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

α is a normalization constant to sum all class probabilities given the features to 1.

2.2 Learning Algorithm

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

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

Here *tot* counts the total number of examples in the training data, and $c(C)$ counts the number of examples belonging to class C. The value $c_i(f_i, C)$ denotes the number of examples belonging to class C which have f_i as the value of the i^{th} attribute. 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) $c_k(f_k, C^*)$ += $weight$
```

Here the weight will be useful for defining the forthcoming algorithms. The $+=$ operator refers to assignment plus addition. For the single naive Bayesian 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] The counter variables $c(C)$ etc. are represented as real numbers.

3 Mixtures of Naive Bayesian Classifiers

The hierarchical mixtures of experts system of Jordan and Jacobs (1992) consists of a number of gating neural networks and expert neural networks. The gating networks learn to gate the predictions of experts to the top layer network which makes the final prediction. The expert networks will specialize on a particular subspace of the full input space, whereas the gating networks learn which expert performs best on a given example. We use a similar system, but now we use naive Bayesian classifiers (NBCs) instead of linear neural networks as gating and expert models. Also for training the gating and expert networks we devised different update algorithms. Like the original HME architecture, the learned hierarchy of NBCs can mirror the hierarchical structure of the classification scheme.

3.1 Architecture

We will explain a 2-layer hierarchical mixtures of naive Bayesian classifiers architecture. Extensions to higher layer architectures are trivial. The system consists of 1 root gating NBC m_0, N first-layer gating NBCs m_1^1 to m_1^N, and $N \times M$ expert NBCs m_2^{11} to m_2^{NM}. Have a look at figure 1 which depicts a two-layer architecture in which the gating models have two sub-models (so here N and M are both 2).

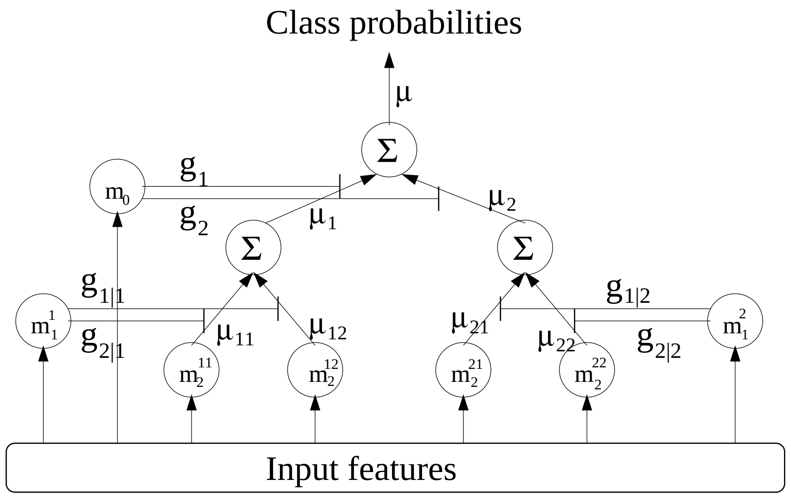

Fig. 1. *The 2-layer architecture consisting of naive Bayesian classifiers. The gating NBCs (m_1^1 and m_1^2) weight the outputs of their sub-models (by values g) and propagate the weighted sum to the gating NBC one layer above. Expert NBCs estimate the class probabilities μ_{ij} given the features.*

Expert NBCs m_2^{ij} output class probabilities given the input features describing the instance D. The class probabilities given by expert m_2^{ij} can be modelled as a vector $\boldsymbol{\mu_{ij}} = (\mu_{ij}^{C_1}, \mu_{ij}^{C_2}, \ldots, \mu_{ij}^{C_L})$, 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. $P_2^{ij}(C|D)$ denotes the probability expert m_2^{ij} assigns to class C given the data-item D. The top-layer gating NBC m_0 computes the following gating values for its sub-models M_i:

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

The gating values can be seen as probabilities that each submodel gives a correct answer. These are then used to weight the answers of the submodels and to propagate the weighted sum to one higher level.

Gating NBCs m_1^i compute output gating values by:

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

So the gating NBCs essentially treat their submodels as classes; they try to classify an instance as the best performing sub-model. Here in this case the M_{ij} refer to the expert networks m_{ij}^2. Given an example described by a set of features, the gating NBCs tell which of its submodels is likely to give the best class-prediction. In this way the best expert is used more for classifying that example. Of course, the goal of the whole system is to learn for what kind of examples which of the submodels perform best; this will be done by expectation maximization as described later.

Our architecture 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). This is needed so that the gating networks and all experts are not completely homogeneous so that the experts will be able to specialize on different parts of the input space. We want to compute the class probabilities of the root model 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 is:

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

$$P_0(C|D) = \sum_i P_0^i(M_i|D) \sum_j P_1^i(M_{ij}|D) P_2^{ij}(C|D)$$

Thus, $\mu^C = P_0(C|D)$. For training this system, the gating models have to predict how well their sub-models perform given some input data, and let the gating weight of the best model given an example converge to the highest value among the models.

3.2 Learning by Expectation Maximization

Expectation Maximization [6] 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.

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

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

and

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

where we use a Gaussian regression model for computing the probability that expert m_2^{ij} generated the correct class label 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.

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 with a learning rate according to their posterior probabilities (Maximization step). Note that we perform the EM step after each example, thus we have an online stochastic learning algorithm. Also, since we use a NBC, the algorithm does not really maximize the probability of generating the correct class, but rather makes a small step to increase this probability. The algorithm is thus a generalized EM (GEM) algorithm [11].

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 counter variables of expert NBCs m_2^{ij} 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 NBC m_2^{ij}. Thus, the weight of the update equals the posterior probability that the expert NBC could have generated the correct class. Updating in this way, causes expert NBCs 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 NBCs learn on each example.

Updating the gating models. For updating the gating NBCs, 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 output (class), and the weight of updating towards this best model on this example is 1.

For the sub-gating NBCs, 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 NBC m_1^i, and call this M_{ib}. Then we update the parameters of model m_1^i by calling: **Update-NBC(**D, M_{ib}, h_i).

Solving the exclusive OR problem. Before running experiments on real world data sets, we first did some experiments to verify whether the hierarchical mixtures of NBCs was able to learn the exclusive OR problem. Learning the exclusive OR problem was no problem at all for a one layer architecture (i.e. it has one gating NBC and two expert NBCs) — it was always able to learn to correctly classify the four training patterns. Thus, the hierarchical system can learn to classify non linearly separable data.

4 Experiments

We have tested the hierarchical mixtures of NBCs on 19 data sets from the UCI repository. We preprocessed continuous (and nominal data with large values) by using the mean and standard deviation and computing significance classes using 1 standard deviation as a separator between two feature values.

Experimental setup. We compared the hierarchical mixtures of naive Bayesian classifiers (HM) to the simple naive Bayesian classifier, bagging and boosting. For the HM architectures, we used a single layer architecture consisting of 4 expert NBCs (which we call 1-4 HM), and a 2-layer architecture consisting of 2 $\times$ 2 expert NBCs (like the one shown in figure 1). We did not try to optimize the architecture or learning parameters for the different datasets. We performed experiments with bagging and boosting in which the number of models was 10. 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: $\gamma = 0.1 + rand(0, 0.01)$, $\sigma = 0.1$.

Test results. Table 1 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 ($p < 0.01$, $p < 0.05$) compared to the simple NBC. The win-loss row indicates how often the mixtures of NBCs, bagging or boosting significantly ($p < 0.05$) work better or worse than the simple naive Bayesian classifier. The average error

Table 1. *The Training results on the 19 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	90.9±0.4	$91.0 \pm 0.5^{=}$	$91.1 \pm 0.1^{=}$	$90.6 \pm 0.5^{=}$	$90.2 \pm 0.7^{--}$
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 :	76.8	78.1	78.3	76.7	77.5
Av. error red.	-	6.7	6.6	-1.5	3.2
Sign. Win-loss :	-	8 : 2	8 : 2	3 : 3	5 : 4

reduction [1] 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 8 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[4], so that it is difficult to improve on this. We can see that this for example holds for the medical diagnosis data sets (Breast Cancer, Ecoli, Hepatitis, Liver Bupa, Pima Indians) where the mutual independency assumption is not (or hardly) violated, since the disease causes the symptoms directly. 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.

Boosting outperforms the NBC significantly in a number of domains such as Car, Chess, and Vote, but on many other data sets does not lead to an improve-

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

ment. In some domains, boosting results in a larger error. Boosting improves the average accuracy, but performs on average less well than the hierarchical system.

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 NBCs, however.

5 Related Work

There have been a number of approaches to extend the naive Bayesian classifier or to combine models. Domingos (2000) used 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 [3]. 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 Bayesian 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 Bayesian 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 Bayesian 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 Bayesian 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 Bayesian 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 Bayesian classifier.

Stewart (1998) developed an algorithm which includes hidden variables to the naive Bayesian 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 [5] 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. The experiments show that the algorithm outperforms a large number of other algorithms such as the hierarchical mixtures of experts and the simple NBC.

Monti and Cooper (1999) describe a Bayesian network classifier that combines a finite mixture model and a naive Bayes model. The finite mixture model is a model which uses a hidden variable which is used as the cause for both the features and the classification. In this case, the classification is not directly treated as the cause for the inputs as in normal naive Bayes classifiers. The described novel combination allows the hidden variable and the classification to cause the input features. This combination is still a (normal) Bayesian network, however, since it does not allow to compute a weighted sum of the classification probabilities returned by the expert NBCs where the weights are given by gating NBCs.

6 Conclusion

We introduced the hierarchical mixtures of naive Bayesian classifiers which is based on the hierarchical mixtures of experts system. All gating and expert models are naive Bayesian classifiers, and the classical naive Bayes updating scheme is extended for training the hierarchical system. We have shown that the hierarchical extension can learn to classify non linearly separable data, which a simple naive Bayesian classifier cannot. In the experiments we compared the novel hierarchical system to the flat naive Bayesian classifier 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 Bayesian classifiers in general outperforms the other tested learning methods. 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 study combining variants of the HME architecture with other algorithms such as support vector machines.

References

1. E. Bauer and R. Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning*, 36:105 – 142, 1999.
2. C. M. Bishop. *Neural Networks for Pattern Recognition.* Clarendon-Press, Oxford, 1995.
3. L. Breiman. Arcing classifiers. *The annals of statistics*, 26:801–849, 1998.
4. Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.
5. C.K. Chow and C.N. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory, IT-14*, 3:462–467, 1968.
6. A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series "B"*, 39:1–38, 1977.

7. P. Domingos. Bayesian averaging of classifiers and the overfitting problem. In P. Langley, editor, *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 223–230. San Francisco, CA: Morgan Kaufmann, 2000.
8. Pedro Domingos and Michael J. Pazzani. On the optimality of the simple Bayesian classifier under zero-one loss. *Machine Learning*, 29(2-3):103–130, 1997.
9. R.O. Duda and P.E. Hart. *Pattern classification and scene analysis.* New York: John Wiley and Sons, 1973.
10. Yoav Freund and Robert E. Schapire. Experiments with a new boosting algorithm. In *Proceedings of the thirteenth International Conference on Machine Learning*, pages 148–156. Morgan Kaufmann, 1996.
11. M. I. Jordan and R. A. Jacobs. Hierarchies of adaptive experts. In J. E. Moody, S. J. Hanson, and R. P. Lippmann, editors, *Advances in Neural Information Processing Systems 4*, pages 985–993. Morgan Kauffmann, 1992.
12. Ron Kohavi. 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, 1996.
13. A. McCallum, R. Rosenfeld, Tom Mitchell, and A. Ng. Improving text classification by shrinkage in a hierarchy of classes. In *Proceedings of the 1998 International Conference on Machine Learning*, July 1998.
14. M. Meila and M.I. Jordan. Learning with mixtures of trees. *Journal of Machine Learning Research*, 1:1–48, 2000.
15. S. Monti and G. F. Cooper. A Bayesian network classifier that combines a finite mixture model and a naive Bayes model. In *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence (UAI-99)*, pages 447–456, 1999.
16. M. Pazzani. Searching for dependencies in Bayesian classifiers. In D. Fisher and H.J. Lenz, editors, *Learning from data: Artificial intelligence and statistics V*, pages 239–248, 1996.
17. Robert E. Schapire, Yoav Freund, Peter Bartlett, and Wee Sun Lee. 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, 1997.
18. B. Stewart. Improving performance of naive Bayes classifiers by including hidden variables. In J. Mira and A.P. Del Pobil, 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, 1998.
19. Kai Ming Ting and Zijian Zheng. Improving the performance of boosting for naive Bayesian classification. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 296–305, 1999.
20. Zijian Zheng. Naive Bayesian classifier committees. In *Proceedings of the tenth European Conference on Machine Learning*, pages 196–207. Berlin: Springer-Verlag, 1998.
21. Zijian Zheng, Geoffrey I. Webb, and Kai Ming Ting. 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, 1999.