AALBORG UNIVERSITY
Department of Computer Science
KDE master group

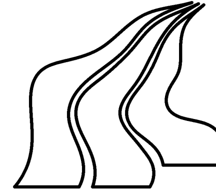
TEXT CATEGORIZATION USING HIERARCHICAL BAYESIAN NETWORK CLASSIFIERS

M.SC. PROJECT

Gytis Karčiauskas

June 2002

AALBORG UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE
FREDRIK BAJERS VEJ 7 - DK 9220 AALBORG Ø



Title:

Text Categorization Using Hierarchical Bayesian Network Classifiers

Project period:
2002 02 01 – 2002 06 06

Project group:
E1 – 121

Members:
Gytis Karčiauskas

Supervisor:
Jiří Vomlel

Pages: 45

Copies: 3

Abstract
In this paper we propose the type of Bayesian networks that we call the hierarchical Bayesian network (HBN) classifiers. We present algorithms for the construction of the HBN classifiers and test them on the Reuters text categorization test collection.

Gytis Karčiauskas

Contents

Introduction **4**

1 Related Work **5**
1.1 Bayesian Network Classifiers 5
1.1.1 Overview of Bayesian Networks 5
1.1.2 Bayesian Network Scoring Functions 6
1.1.3 A Naive Bayes Classifier 7
1.1.4 Extensions of a Naive Bayes Classifier 9
1.1.5 Other Bayesian Network Classifiers 9
1.2 Feature Clustering 10
1.3 Text Categorization 11

2 Hierarchical Bayesian Network Classifiers **13**

3 Classifier Construction Algorithms **16**
3.1 Motivation 16
3.2 Construction of a Hierarchical Bayesian Network Classifier . . 18
3.3 Construction of a Hierarchical Naive Bayes Classifier 20
3.4 General Feature Clustering Algorithm 20
3.5 Feature Clustering Algorithm using Probability Average . . . 20
3.6 Feature Clustering Algorithm using OR 23
3.7 Feature Clustering Algorithm using Independence Tests . . . 24

4 Performance Experiments **29**
4.1 Test Setup 29
4.1.1 Data Used 29
4.1.2 Text Indexing 30
4.1.3 Classifiers Tested 30
4.2 Test Results 31
4.2.1 Experiments on the Validation Data 31
4.2.2 Experiments on the Test Data 35
4.2.3 Feature Clustering Algorithms 35

5 Conclusions and Future Work **41**

List of Figures

1.1 Naive Bayes Classifier as a Bayesian Network 8

2.1 An Example HBN Classifier 13

3.1 Bayesian Network with All Features Being Parents of the Class 16
3.2 Bayesian Network with Two Hidden Variables 17
3.3 Function ConstructHBNClassifier 19
3.4 Function ConstructSubtree 19
3.5 Function ClusterFeatures 20
3.6 Function ClusterFeaturesAvg 22
3.7 Function Merge 23
3.8 Function InfoLoss 23
3.9 Function ClusterFeaturesOr 24
3.10 Function Merge (for ClusterFeaturesOr) 24
3.11 Function ClusterFeaturesDep 26
3.12 Function ImproveClustering 28

4.1 *HBN-AVG* for Class Trade 37
4.2 *HBN-OR* for Class Trade 38
4.3 *HBN-DEP* for Class Trade 39
4.4 *HBN-DEP* with $\alpha = 0$ for Class Trade 40

List of Tables

4.1 The Final Parameter Values For the HBN Classifiers 32
4.2 The Performance of *HBN-AVG* with Different Number of Features 32
4.3 The Performance of *HBN-OR* with Different Number of Features 32
4.4 The Performance of *HBN-DEP* with Different Number of Features 32
4.5 The Performance of *HNB-AVG* Compared to *HBN-AVG* . . . 33
4.6 The Performance of *HNB-OR* Compared to *HBN-OR* 33
4.7 The Performance of *HNB-DEP* Compared to *HBN-DEP* . . . 33
4.8 The Performance of *NB* with Different Number of Features . 34
4.9 The Performance of *SVM* with Different Number of Features 34
4.10 The Performance on the Test Data 36

Introduction

Text categorization, defined as the activity of labeling natural language texts with thematic categories from a predefined set ([Seb02]), is a task frequently performed by humans. Because often there are many documents in a digital form and they have to be categorized, there is a big need for automatic classifiers[1] that would perform this task or would assist humans in performing it.

Generally there are two ways for making automatic classifiers. In a *knowledge engineering* approach, the knowledge of human experts is described as a set of rules, which are then used in the process of classification. The disadvantages of this approach are that many work needs to be done to make human knowledge explicit and for each new domain a separate formulation of the rules needs to be done manually again. In a *machine learning* approach, the classifier is built automatically from the set of the already classified instances. The major work shifts from making human knowledge explicit to creating algorithms that classify new instances based on the information about the already classified instances. Classifiers for different domains can be learned using the same algorithm.

In this paper we use machine learning methods for automatic text categorization. Particularly, we use Bayesian networks, because they seem to be suitable for modeling the uncertainty that is present in the categorization task. We propose the type of Bayesian networks that we call the *hierarchical Bayesian network* (HBN) classifiers. We test them on the Reuters text categorization test collection.

In Chapter 1 we describe the related work. In Chapter 2 we define the HBN classifiers and describe their properties. In Chapter 3 we describe our algorithms for the construction of classifiers. In Chapter 4 we give the results of the experiments performed. In Chapter 5 we give the conclusions and possible future work directions.

[1] We will use the terms “categorization” and “classification” as synonyms.

Chapter 1

Related Work

In this chapter first we review the work already done in the area of Bayesian network classifiers. Then we describe the algorithms that can be used for feature clustering. Finally we give a general overview of the machine learning in text categorization.

1.1 Bayesian Network Classifiers

In this section first we give a brief overview of Bayesian networks. Then we discuss Bayesian network scoring functions and problems related to using them for Bayesian network classifiers. After that we review the already available Bayesian network classifiers. We divide them into three groups – a naive Bayes classifier, classifiers that extend a naive Bayes, and other Bayesian network classifiers.

1.1.1 Overview of Bayesian Networks

Bayesian network, described, for example, by Jensen [Jen01], can be defined as a set of variables and a set of directed edges between variables, where each variable has a finite set of mutually exclusive states, the variables together with the edges form a directed acyclic graph, and to each variable A with parents $pa(A) = \{B_1, \ldots, B_k\}$ there is attached a conditional probability table $P(A|B_1, \ldots, B_k)$.[1] The edges between the variables model dependence relationships: an edge going from variable B to variable A means that the state of variable A depends on the state of variable B. The probability distribution over all variables $A_1, \ldots, A_n$ in a Bayesian network is $P(A_1, \ldots, A_n) = \prod_i P(A_i \mid pa(A_i))$. When using Bayesian networks as classifiers, the variables represent classes and features, and the edges indicate the relationships between them.

[1]Variable A having variable B as its parent means that there is an edge going from variable B to variable A.

In this paper we will use the following notation. θ indicates the parameters of a Bayesian network B, i.e. the conditional probability distributions for all variables A_i in B. $P_{B_\theta}(X_1, \ldots, X_k)$ indicates the probability distribution over variables $X_1, \ldots, X_k$ defined by a Bayesian network B with parameters θ. $P_{B_\theta}(x_1, \ldots, x_k)$ indicates the probability that the corresponding variables are in particular states $x_1, \ldots, x_k$. $D = \{< f_1^i, \ldots, f_n^i, c^i >, i = 1, \ldots, N\}$ indicates the training data of N cases with feature variables $F_1, \ldots, F_n$ and a class variable C.

1.1.2 Bayesian Network Scoring Functions

When learning Bayesian network classifier from training data, a *scoring function* is used for evaluation of the candidate Bayesian networks. Usually scoring functions seek for a simple Bayesian network that fits the training data (that is, the joint probability distribution over all the variables). Usually the *likelihood* of a Bayesian network B with parameters θ given training data $D = \{< f_1^i, \ldots, f_n^i, c^i >, i = 1, \ldots, N\}$ is used to measure how B fits D. The likelihood is defined as

$$L(B_\theta|D) = \prod_{i=1}^{N} P_{B_\theta}(f_1^i, \ldots, f_n^i, c^i) \ .$$

Often the *log likelihood* $LL(B_\theta|D) = \sum_{i=1}^{N} \log P_{B_\theta}(f_1^i, \ldots, f_n^i, c^i)$ is used instead of likelihood.

Friedman et al. [FGG97] use the *minimal description length* (MDL) scoring function to evaluate the candidate Bayesian network classifiers. The MDL scoring function of a Bayesian network B with parameters θ given training data D is defined as

$$MDL(B_\theta|D) = \frac{\log N}{2}|B_\theta| - LL(B_\theta|D) \ ,$$

where $|B_\theta|$ is the number of parameters in the network. The first term in a definition of MDL represents the length of the description of B_θ, and the second term represents the suitability of B_θ for describing D. When searching through the space of possible Bayesian networks, we try to find the one with the minimal MDL score. However, a good classifier should fit best the probability of the *class* given the values of the features in the training data rather than the joint probability distribution over all the variables in the training data. Therefore, a Bayesian network selected as the best according to MDL or another non-specialized scoring function sometimes performs poorly as a classifier. To deal with this problem, the *conditional likelihood* can be used instead of likelihood. The conditional likelihood of a Bayesian network B with parameters θ given training data $D = \{< f_1^i, \ldots, f_n^i, c^i >$

$, i = 1, \ldots, N\}$ is defined as

$$CL(B_\theta|D) = \prod_{i=1}^{N} P_{B_\theta}(c^i|f_1^i, \ldots, f_n^i) \ .$$

Often the *conditional log likelihood* $CLL(B_\theta|D) = \sum_{i=1}^{N} \log P_{B_\theta}(c^i|f_1^i, \ldots, f_n^i)$ is used instead of the conditional likelihood. Then, according to Friedman et al. [FGG97], the *conditional MDL* score that would avoid the above mentioned problem of the MDL score can be derived. The conditional MDL scoring function of a Bayesian network B with parameters θ given training data D is then defined as

$$CMDL(B_\theta|D) = \frac{\log N}{2}|B_\theta| - N \cdot CLL(B_\theta|D) \ .$$

However, the conditional likelihood has one serious drawback when compared to the likelihood. For a fixed structure of a Bayesian network B, there is a closed form solution for the parameters θ that maximize $L(B_\theta|D)$. Namely, the conditional probabilities are simply equal to the frequencies of the corresponding values for the variables in the training data D. However, for the conditional likelihood no such general solution exists. Only in a case when the class variable has no children and all the parents of the class variable are feature variables, substituting the frequencies in D as the parameters of B maximizes the conditional likelihood. This means that generally when learning a Bayesian network structure, the computationally expensive methods to estimate the conditional likelihood have to be used.

1.1.3 A Naive Bayes Classifier

The most simple of Bayesian network classifiers is a *naive Bayes* classifier, described, for example, by Duda and Hart [DH73]. In a naive Bayes classifier it is assumed that the features are independent given the value of the class, that is, for the class variable C and any feature variables F_i, F_j,

$$P(F_i|F_j, C) = P(F_i|C)$$

for all possible values of F_i, F_j and C, whenever $P(C) > 0$. Figure 1.1 depicts a naive Bayes classifier as a Bayesian network, where the class variable is C and the feature variables are $F_1, \ldots, F_n$. The probabilities $P(C), P(F_1|C), \ldots, P(F_n|C)$ are estimated from the training data. When a new instance with the known values of feature variables $F_1, \ldots, F_n$ has to be classified, Bayes rule is used:

$$P(C|F_1, \ldots, F_n) = \frac{P(F_1, \ldots, F_n|C)P(C)}{P(F_1, \ldots, F_n)} \ .$$

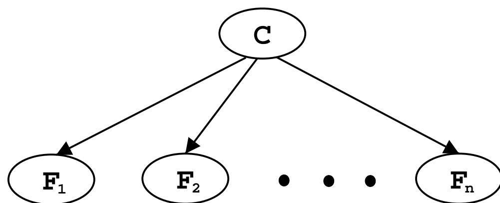


Figure 1.1: Naive Bayes Classifier as a Bayesian Network

Since $F_1, \ldots, F_n$ are assumed to be independent given C, $P(F_1, \ldots, F_n|C) = P(F_1|C) \ldots P(F_n|C)$, and the above formula becomes

$$P(C|F_1, \ldots, F_n) = \frac{P(F_1|C) \ldots P(F_n|C)P(C)}{P(F_1, \ldots, F_n)} \ . \tag{1.1}$$

The probabilities in the numerator are known, and the probability in the denominator does not depend on the class value, so it need not be calculated.

Often on a naive Bayes classifier *smoothing*, described, for example, by Friedman et al. [FGG97] is used. When no smoothing is used, the probability that feature variable F_i is in state f given class c is estimated simply by counting the occurrences in the training data:

$$P(F_i = f|C = c) = \frac{n}{N} \ ,$$

where N is the number of cases in the training data in which C is in state c, and n is the number of cases in the training data in which C is in state c and F_i is in state f. If $n = 0$ and thus $P(F_i = f|C = c) = 0$, then the whole numerator in Equation 1.1 becomes zero, and the probability $P(C = c|F_1, \ldots, F_n)$ for class c is zero. That is, F_i being in state f causes probability for class c to be zero no matter how large the other probabilities in the numerator from Equation 1.1 would be. When smoothing is used, $P(F_i = f|C = c)$ is estimated using the formula

$$P(F_i = f|C = c) = \frac{n + N_0}{N + mN_0} \ ,$$

where N_0 is a small constant and m is the number of states for the variable F_i. In other words, it is assumed that before seeing training data the variable F_i has been observed to be in each of its states N_0 times when the class was c. This guarantees that $P(F_i = f|C = c)$ is always above zero.

Obviously, in reality the assumption that the features are independent given the value of the class is violated very often. However, the performance of a naive Bayes classifier is often very good and comparable to the performance of much more sophisticated classifiers, as shown by Domingos and Pazzani [DP97].

1.1.4 Extensions of a Naive Bayes Classifier

Large part of the work done in the area of the Bayesian network classifiers deals with the extensions of a naive Bayes classifier. Since a naive Bayes classifier with its simplifying assumption about a conditional feature independence performs well, it is natural to hope that Bayesian classifiers that allow features to depend on each other would perform even better.

Friedman et al. [FGG97] extend a naive Bayes classifier by allowing each feature variable to have at most one other feature as a parent in addition to a class variable. Such a network is called a *tree-augmented naive Bayes* (TAN) classifier, because feature variables together with edges between them can form a tree. The authors present an algorithm that efficiently computes augmented edges for a TAN classifier from the training data. The learned network maximizes the likelihood of a TAN classifier given training data. Even more general extension of a naive Bayes classifier is to allow the features to form unrestricted Bayesian networks. Such a network is called an *augmented naive Bayes* (ANB) classifier. However, the number of possible network structures in this case is very large. Friedman et al. use a greedy heuristic search that tries to minimize the MDL score.

Cheng and Greiner [CG01] also propose algorithms for learning an ANB classifier. For learning a network structure they use conditional independence based algorithms rather than algorithms that seek a structure that maximizes a scoring function. Using statistical tests, the conditional independence relationships between the features are found. These relationships are used as constraints to construct a Bayesian network.

Keogh and Pazzani [KP99] propose an algorithm that they call *SuperParent* for learning a TAN classifier. The edges between feature variables are added one at a time based on the predictive accuracy of the candidate networks. Zhang and Ling [ZL01] extend this work by proposing an algorithm called *StumpNetwork*. They exploit the idea that often the dependence between the attributes tends to cluster into groups. The construction of the classifier is again based on the predictive accuracy of the candidate networks.

Langley and Sage [LS94] propose a *selective naive Bayes* classifier. Here a subset from the initial set of features is selected to be used by a naive Bayes classifier.

1.1.5 Other Bayesian Network Classifiers

Friedman et al. [FGG97] use a greedy search to find an unrestricted Bayesian network that minimizes MDL score. Cheng and Greiner [CG01] use conditional independence tests for learning unrestricted Bayesian network classifiers. These two approaches are similar to the corresponding algorithms for learning ANB classifiers from the previous section.

Kontkanen et al. [KMT01] use a mixture of diagnostic Bayesian network

classifiers. In a diagnostic Bayesian network, all the edges connected to the class variable are arriving edges from feature variables. The authors use a mixture of classifiers, where different networks have different sets of parents for the class node. The sets of features to be used in networks from a mixture are found based on the predictive accuracy of the different mixtures.

1.2 Feature Clustering

While there are many general data clustering algorithms (see, for example, Han and Kamber [HK01]), we are aware only of several algorithms for feature clustering for classification.

Baker and McCallum [BM98] describe an algorithm for clustering words into groups specifically for the benefit of document classification. The main idea behind their algorithm is that similar words are those for which the distributions of the class given those words are similar. When similar words are assigned to the same cluster, all those words in the cluster are treated as the same feature. When features w_t and w_s are assigned to the same cluster, the probability of the class variable C given the new feature $w_t \vee w_s$ is defined as

$$P(C|w_t \vee w_s) = \frac{P(w_t)}{P(w_t) + P(w_s)} P(C|w_t) + \frac{P(w_s)}{P(w_t) + P(w_s)} P(C|w_s) \ . \quad (1.2)$$

In the word clustering algorithm, initially clusters with one word in each of them are created. Then clusters are repeatedly joined, each time joining two clusters w_t and w_s that minimize the average of the Kullback-Leibler divergence to the mean, defined as

$$\begin{array}{c} P(w_t) \cdot D_{KL}(P(C|w_t) \parallel P(C|w_t \vee w_s)) \\ +P(w_s) \cdot D_{KL}(P(C|w_s) \parallel P(C|w_t \vee w_s)) \ , \end{array} \quad (1.3)$$

where $D_{KL}(P(C|w_a) \parallel P(C|w_b)) = \sum_{j=1}^{|C|} P(c_j|w_a) \log \frac{P(c_j|w_a)}{P(c_j|w_b)}$. The authors test a naive Bayes classifier that uses clusters constructed by their algorithm as features. The accuracy stays similar as in the case of using single words as features, while the number of features is reduced up to three orders of magnitude.

Slonim and Tishby [ST01] present essentially the same algorithm for word clustering by using an *information bottleneck* framework [TPB99] as a theoretical basis for it. The clusters are repeatedly joined, each time joining two clusters of the current partition into a single new cluster in a way that locally minimizes the loss of mutual information about the class variable. With the mutual information between the random variables X and Y defined as $I(X,Y) = \sum_{x \in X, y \in Y} P(x)P(y|x) \log \frac{P(y|x)}{P(y)}$, the algorithm each time joins the clusters w_t and w_s that minimize

$$I(w_t, C) + I(w_s, C) - I(w_t \vee w_s, C) \ . \quad (1.4)$$

It can be shown that expressions 1.3 and 1.4 are equal. Slonim and Tishby also test a naive Bayes classifier that uses clusters constructed by their algorithm as features. When there was few training data, the classification accuracy improved up to 18% compared to the case of using single words as features.

1.3 Text Categorization

There has been much research done in the area of the automatic text categorization. A survey of machine learning approaches to text categorization is given by Sebastiani [Seb02]. First, for a classifier to be able to work with a text, an *indexing* of the text has to be done. Usually this is done by representing the text as a vector of *feature* weights, where features are the words that appear or do not appear in the text. Usually feature weights take values from the interval [0; 1]. In a special case of binary features there are two possible values: 1 for the presence and 0 for the absence of the feature in the text. Often before the indexing the *stemming* (i.e. taking the words that have the same word stem as the same feature) and the removal of *function words* (i.e. topic-neutral words such as articles, prepositions) is performed. Even after this preprocessing the number of features is often too large for a machine learning algorithm because of the too long computation time and *overfitting*. By overfitting we mean the adapting of a classifier to the particular instances in the training data instead of making generalizations about the classes. Therefore a *dimensionality reduction*, where the number of features used by the classifier is reduced, is performed. Dimensionality reduction can be either *local*, where for each class a separate set of features is chosen, or *global*, where the set of features chosen is the same for all the classes. Also dimensionality reduction can be performed by either *feature selection*, where the set of the new features is a subset of the set of all the original features, or *feature extraction*, where the new features are obtained by combinations or transformations of the original ones.

One of the ways to perform feature selection is to use an *information gain* method, described, for example, by Yang and Pedersen [YP97]. Speaking generally, information gain for a feature F measures the number of bits of information obtained for the prediction of class C by knowing the state of F. It is calculated as $IG(F) = E(C) - E(C|F)$, where E denotes the entropy.

When the features are defined and the texts are indexed based on those features, the classifiers can be constructed. Dumais et al. [DPHS98] provide a comparison of different automatic learning algorithms for text categorization. Linear Support Vector Machines have been found better than decision trees, Bayesian networks, and naive Bayes classifiers.

The effectiveness of text classifiers is usually measured in terms of *precision* (π) and *recall* (ρ). As defined by Sebastiani [Seb02], precision is the

conditional probability that if a random document is assigned to a particular class, this decision is correct. Recall is defined as the conditional probability that if a random document ought to be assigned to a particular class, this decision is taken. If we denote the number of true positive, false positive, and false negative classifications as TP, FP, and FN, then the precision and recall are computed as

$$\pi = \frac{TP}{TP + FP} , \qquad \rho = \frac{TP}{TP + FN} .$$

There are two ways of measuring precision and recall for multiple classes. In *microaveraging*, π and ρ are computed by taking ratios of the corresponding total values of TP, FP, and FN. In *macroaveraging*, first "local" values of π and ρ for each class are computed. The final values are obtained by simply taking averages of these. These two methods may give quite different results, because the ability of a classifier to behave well on the classes with few positive instances is emphasized much more by macroaveraging than by microaveraging.

When measuring effectiveness, usually we want one number that combines precision and recall to be reported. The commonly used approach is to report a *breakeven* point – the value at which π equals ρ.[2]

[2]The values of π and ρ change as we change the value of threshold that specifies the probability of a document belonging to a particular class that must be exceeded for the document to be assigned to that class. Increasing the threshold increases precision and decreases recall, while decreasing the threshold increases recall and decreases precision for the class.

Chapter 2

Hierarchical Bayesian Network Classifiers

In this chapter first we define what do we mean by a hierarchical Bayesian network classifier. After that we give propositions that relate conditional likelihood to the likelihood of a hierarchical Bayesian network classifier.

Definition A *hierarchical Bayesian network* (HBN) classifier with feature variables $F_1, \ldots, F_n$ and a class variable C is a Bayesian network with the following tree structure: C is the root, $F_j (j = 1, \ldots, N)$ are the leaves, and hidden variables $H_1, \ldots, H_k (k \geq 0)$ are the non-leaf nodes of the tree. All arcs in the Bayesian network are going towards the root.

In Figure 2.1 we give an example of the HBN classifier with class variable C, feature variables $F_1, \ldots, F_7$, and hidden variables H_1, H_2, H_3.

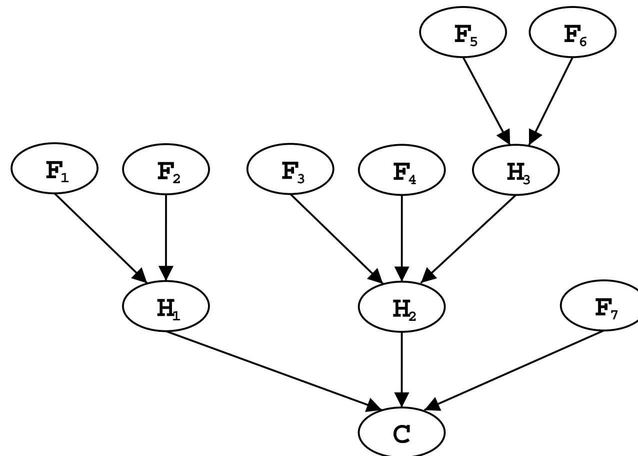


Figure 2.1: An Example HBN Classifier

Given by its structure, the probability distribution over all variables for a HBN classifier B with parameters θ is

$$\begin{aligned} & P_{B_\theta}(C, H_1, \ldots, H_k, F_1, \ldots, F_n) = \\ & = \quad P_{B_\theta}(C \mid pa(C)) \cdot \left(\prod_{i=1}^{k} P_{B_\theta}(H_i \mid pa(H_i)) \right) \cdot \left(\prod_{i=1}^{n} P_{B_\theta}(F_i) \right) \ . \end{aligned}$$

As mentioned in Section 1.1.2, maximizing likelihood rather than conditional likelihood of a Bayesian network classifier can lead to a poor performance. Bellow we prove that for a HBN classifier this is not the case. This will allow us to learn the parameters for a HBN classifier by trying to maximize its likelihood.

Proposition 1 *Let $D = \{< f_1^i, \ldots, f_n^i, c^i >, i = 1, \ldots, N\}$ be a training data. Let B be a HBN classifier with feature variables $F_1, \ldots, F_n$ and a class variable C. Then for any parameters θ_1, θ_2 that satisfy $P_{B_{\theta_1}}(F_j) = P_{B_{\theta_2}}(F_j), \forall j = 1, \ldots, n$:*

$$L(B_{\theta_1}|D) > L(B_{\theta_2}|D) \iff CL(B_{\theta_1}|D) > CL(B_{\theta_2}|D) \ .$$

Proof. By definition, we have

$$\begin{aligned} L(B_{\theta_1}|D) &= \prod_{i=1}^{N} P_{B_{\theta_1}}(f_1^i, \ldots, f_n^i, c^i) \\ &= \left(\prod_{i=1}^{N} P_{B_{\theta_1}}(c^i | f_1^i, \ldots, f_n^i) \right) \cdot \left(\prod_{i=1}^{N} P_{B_{\theta_1}}(f_1^i, \ldots, f_n^i) \right) \\ &= CL(B_{\theta_1}|D) \cdot \prod_{i=1}^{N} P_{B_{\theta_1}}(f_1^i, \ldots, f_n^i) \ . \end{aligned}$$

Since variables $F_1, \ldots, F_n$ have no common parents (i.e. they are independent), $P_{B_{\theta_1}}(f_1^i, \ldots, f_n^i) = P_{B_{\theta_1}}(f_1^i) \cdot \ldots \cdot P_{B_{\theta_1}}(f_n^i), \forall i = 1, \ldots, N$. So,

$$\begin{aligned} L(B_{\theta_1}|D) &= CL(B_{\theta_1}|D) \cdot \left(\prod_{i=1}^{N} P_{B_{\theta_1}}(f_1^i) \cdot \ldots \cdot P_{B_{\theta_1}}(f_n^i) \right) \\ &= CL(B_{\theta_1}|D) \cdot K \ , \end{aligned}$$

where $K = \prod_{i=1}^{N} P_{B_{\theta_1}}(f_1^i) \cdot \ldots \cdot P_{B_{\theta_1}}(f_n^i)$ does not depend on parameters that specify conditional probabilities for non-feature variables in B. Because of the assumption $P_{B_{\theta_1}}(F_j) = P_{B_{\theta_2}}(F_j), \forall j = 1, \ldots, n$, we also have that $K = \prod_{i=1}^{N} P_{B_{\theta_2}}(f_1^i) \cdot \ldots \cdot P_{B_{\theta_2}}(f_n^i)$. Thus we can write

$$L(B_{\theta_2}|D) = CL(B_{\theta_2}|D) \cdot K \ .$$

■

Proposition 2 *Let* $D = \{< f_1^i, \ldots, f_n^i, c^i >, i = 1, \ldots, N\}$ *be a training data. Let* B *be a HBN classifier with feature variables* $F_1, \ldots, F_n$ *and class variable* C*. Then*

$$\theta \text{ maximize } L(B_\theta|D) \Rightarrow \theta \text{ maximize } CL(B_\theta|D) \ .$$

Proof. It is known (see, for example, Friedman et al. [FGG97]) that for θ that maximize $L(B_\theta|D)$ it holds that $P_{B_\theta}(F_j) = P_D(F_j), \forall j = 1, \ldots, N$, with P_D defined as

$$P_D(f_j = k) = \frac{1}{N}\sum_{i=1}^{N} 1_{f_{jk}}(f_j^i), \forall j = 1, \ldots, n, \forall k = 1, \ldots, |F_j| \ ,$$

where $1_{f_{jk}}(f_j^i) = \begin{cases} 1, & \text{if } f_j^i = k \\ 0, & \text{else} \end{cases}$.

On the other hand, $CL(B_\theta|D)$ does not depend on parameters of $P_{B_\theta}(F_j)$, because $CL(B_\theta|D) = \prod_{i=1}^{N} P_{B_\theta}(c^i|f_1^i, \ldots, f_n^i)$, and in probability $P_{B_\theta}(c^i|f_1^i, \ldots, f_n^i)$ all F_j are instantiated. So, we can have $P_{B_\theta}(F_j) = P_D(F_j)$. Together with Proposition 1, this proves Proposition 2.

■

Chapter 3

Classifier Construction Algorithms

In this chapter first we give the motivation for using the HBN classifiers and choosing the presented methods for learning them. Then we describe our algorithms for the construction of classifiers.

3.1 Motivation

Usually, in texts there are many dependencies between features that represent words. When constructing Bayesian network classifiers, the common approach to deal with feature dependencies is to extend a naive Bayes classifier, as discussed in Section 1.1.4. Another approach is to allow the class variable to have parents. In the extreme case, we would have a model where all the feature variables $F_1, \ldots, F_n$ are the parents of the class variable C, as depicted in Figure 3.1. In this case, the dependencies between the features

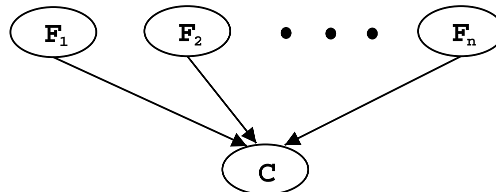


Figure 3.1: Bayesian Network with All Features Being Parents of the Class

are modeled. But an obvious problem with such a model is that the size of conditional probability table for variable C grows exponentially with n, and the number of parameters needed to specify $P(C|F_1, \ldots, F_n)$ can quickly become larger than the number of cases in training data. One of the possible solutions is to introduce hidden variables that have feature variables

as parents and the class variable as a child. For example, if we have 20 features, but we do not want to have variables with more than 10 parents, we can introduce two hidden variables H_1 and H_2, as shown in Figure 3.2. In general case, we can have a hierarchy of hidden variables (i. e., hidden

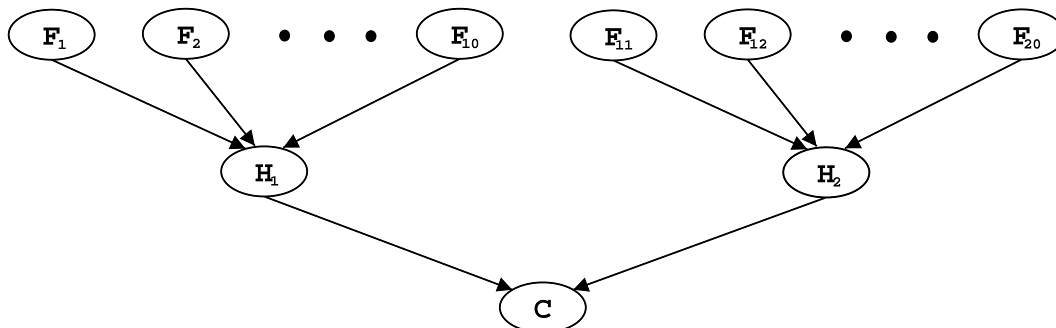


Figure 3.2: Bayesian Network with Two Hidden Variables

variables having other hidden variables as parents). That is, we can have the hierarchical Bayesian network classifiers, as defined in Chapter 2. Since in our text classification problem the features are binary, all the feature variables in our HBN classifiers have two states. For simplicity, we deal only with HBN classifiers where hidden variables have two states, and the maximum number of parents for any variable is a parameter that we call a *branching factor*.

Ideally, we would like to have a fast algorithm that computes the maximal conditional log likelihood for a given HBN classifier structure. Then, for a given training data, we could perform a search among the HBN classifier structures trying to find the one that minimizes the conditional minimal description length score, described in Section 1.1.2. However, we do not have a closed form solution for the parameters that maximize the conditional log likelihood for a given HBN classifier structure. So, approximate and computationally expensive methods have to be used. To compute the parameters for a given HBN classifier structure we use the EM algorithm, described, for example, by Cowell et al. [CDLS99]. The EM algorithm tries to maximize the likelihood for a given structure, and according to Proposition 1 from Chapter 2, it tries to maximize the conditional likelihood at the same time. Since running the EM algorithm is time consuming, we learn the parameters only after the final structure is learned.

To learn the structure of a HBN classifier, we perform *feature clustering*. For any non-leaf node of the tree, different clusters correspond to different subtrees of that node. We present three different feature clustering algorithms that more or less try to group similar features into the same clusters. These algorithms are no guaranteed to find optimal solutions, and just use

different heuristics for clustering features. The general algorithm for the construction of the HBN classifier does not depend on the particular feature clustering algorithm used.

For the comparison, we also try an algorithm for the construction of a *hierarchical naive Bayes* (HNB) classifier. HNB has the same structure as the HBN classifier with the exception that all arcs are going not towards but from the root. The HNB classifier uses the same feature clustering algorithms as the HBN classifier.

3.2 Construction of a Hierarchical Bayesian Network Classifier

In this section we describe an algorithm for the construction of the HBN classifier. The algorithm is given in Figures 3.3 and 3.4. Function **ConstructHBNClassifier** takes two arguments. The first argument is the training data set $\mathcal{D}$ of binary feature vectors, with one of the classes $\{c_1, \ldots, c_M\}$ assigned to each feature vector. The second argument is the branching factor B. The output of the function is the HBN classifier. If the number of features is not higher than B then all the feature variables are simply made the parents of the class variable. Otherwise, the feature clustering algorithm described in Section 3.4 is used. Features are divided into B clusters, and B variables (one corresponding to each cluster) are made the parents of the class variable. For dealing with each of those B clusters, function **ConstructSubtree** is used. It takes the set of all the features in a cluster as an argument, and returns a variable that should be at the bottom of the subtree for the given cluster. If the cluster contains only one feature then the subtree for the cluster consists only of that feature variable. Otherwise, there is a hidden variable at the bottom of the subtree. If the cluster contains no more than B features then the parents of the hidden variable are only those feature variables. Otherwise, feature clustering algorithm is used to make further partitioning of the current cluster. And for each of the new partitions function **ConstructSubtree** is called recursively. Symbols X_i in the pseudocode denote the clusters of variables.

After learning the structure of the HBN classifier, the EM algorithm for learning the parameters of the Bayesian network is used. We used a multiple-restart approach, as described by Chickering and Heckerman [CH97]. The number of starting configurations of the parameters is 64. For each starting configuration, random conditional probabilities for the hidden variables are generated. For the class variable, the initial probabilities are the same for all the classes and all the parent configurations. Since feature variables have no parents, the EM algorithm sets the probabilities for the feature variables to be simply the frequencies of their corresponding states in the training

data. The threshold for terminating the EM algorithm[1] was set to 0.0001.

Function **ConstructHBNClassifier**$(\mathcal{D}, B)$:

1. Let C be the class variable from $\mathcal{D}$.
2. Let $\mathcal{F} = \{F_1, \ldots, F_N\}$ be the set of all the feature variables from $\mathcal{D}$.
3. If $|\mathcal{F}| \leq B$, make $F_1, \ldots, F_N$ the parents of C.
4. Else,
   (a) Let $\{X_1, \ldots, X_B\} =$ **ClusterFeatures**$(\mathcal{D}, \mathcal{F}, B)$.
   (b) For each $i = 1, \ldots, B$ make the variable returned by **ConstructSubtree**(X_i) the parent of C.
5. Using EM algorithm, learn the probabilities for the Bayesian network from $\mathcal{D}$.
6. Return the constructed Bayesian network.

Figure 3.3: Function ConstructHBNClassifier

Sub-function **ConstructSubtree**$(\mathcal{F})$:

1. If $|\mathcal{F}| = 1$, return the feature variable in $\mathcal{F}$.
2. Else,
   (a) Let H be a new hidden variable with two states.
   (b) If $|\mathcal{F}| \leq B$, make feature variables from $\mathcal{F}$ the parents of H.
   (c) Else,
      i. Let $\{X_1, \ldots, X_B\} =$ **ClusterFeatures**$(\mathcal{D}, \mathcal{F}, B)$.
      ii. For each $i = 1, \ldots, B$ make the variable returned by **ConstructSubtree**(X_i) the parent of H.
   (d) Return H.

Figure 3.4: Function ConstructSubtree

[1]The EM algorithm terminates when the difference between the log likelihood for two successful iterations becomes less than the specified threshold.

3.3 Construction of a Hierarchical Naive Bayes Classifier

In this section we describe an algorithm for the construction of the HNB classifier. Function **ConstructHNBClassifier** takes the same arguments as function **ConstructHBNClassifier**. The output of the function is the HNB classifier where the class variable and each hidden variable has no more than B children. The pseudocode for the function **ConstructHNBClassifier** is the same as for the function **ConstructHBNClassifier** with the exception that in steps 3, 4.b, 2.b, and 2.c.ii from Figures 3.3 and 3.4 the corresponding variables are made children rather than parents of the variables C or H. The difference in using the EM algorithm is that the random initial conditional probabilities are generated not only for the hidden but also for the feature variables.

3.4 General Feature Clustering Algorithm

In this section we present a general function for feature clustering. Function **ClusterFeatures** takes the training data set $\mathcal{D}$, the set of features $\mathcal{F}$ to be clustered, and the number of clusters B as its arguments. It returns a partition $\{X_1, \ldots, X_B\}$ of $\mathcal{F}$ defined as:

- $X_i \subseteq \mathcal{F}, X_i \neq \emptyset, \forall i = 1, \ldots, B$,
- $X_i \cap X_j = \emptyset, \forall i \neq j$,
- $\cup_{i=1}^{B} X_i = \mathcal{F}$.

As seen from Figure 3.5, function **ClusterFeatures** is just a wrapper for the particular feature clustering functions **ClusterFeaturesAvg**, **ClusterFeaturesOr**, and **ClusterFeaturesDep**.

Function **ClusterFeatures**($\mathcal{D}, \mathcal{F}, B$):
Call one of the functions **ClusterFeaturesAvg**, **ClusterFeaturesOr**, **ClusterFeaturesDep**.

Figure 3.5: Function ClusterFeatures

3.5 Feature Clustering Algorithm using Probability Average

In this section we describe a feature clustering algorithm that, similarly to the algorithm of Slonim and Tishby [ST01] presented in Section 1.2, merges

smaller clusters into larger ones by using an information loss criteria. In our classifiers, both the presence and the absence of a feature in a text is used as an evidence, while in both [BM98] and [ST01] only the presence of a feature is used as an evidence. That is why instead of Equation 1.2 from Section 1.2 we must have two equations – one for computing $P(C|w_t \vee w_s = 1)$ (feature $w_t \vee w_s$ is present in the text), and one for computing $P(C|w_t \vee w_s = 0)$ (feature $w_t \vee w_s$ is absent in the text). Also, we redefine the weight of feature w_i to be the number of words that actually make up feature w_i (i.e., the size of cluster w_i) rather than $P(w_i)$. After the initial experiments we also made the adjustments to Equation 1.4 from Section 1.2 to penalize joining of the already large clusters.

Function **ClusterFeaturesAvg**, given in Figure 3.6, takes the same arguments as function **ClusterFeatures** and additionally the cluster size penalty parameter α. **ClusterFeaturesAvg** returns a partition $\{X_1, \ldots, X_B\}$ of $\mathcal{F}$. The number of features N that have to be clustered can differ for different calls of function **ClusterFeaturesAvg** depending on how deep in the HBN tree is a variable the parents of which have to be clustered. Each of the initial clusters contains one feature. Probabilities $P(X_n)$, $P(C|X_n)$, and $P(C)$ are calculated by taking frequencies from the training data. These probabilities are enough to compute the mutual information $I(X_i, C)$ between cluster and class variables. Step 4 of the function contains the main loop, where in each iteration two clusters that minimize the modified information loss criteria are merged into one. The loop continues as long as the number of clusters is higher than B.

Function **Merge**, given in Figure 3.7, returns a cluster that is obtained by merging clusters X_i and X_j. The information that the algorithm needs about any cluster X' is the mutual information $I(X', C)$ and the cluster size $|X'|$. To compute $I(X', C)$, probabilities $P(X')$ and $P(C|X')$ are needed. In this algorithm, these probabilities are computed by taking the weighted averages of the corresponding probabilities from the clusters X_i and X_j. The weights are the sizes of clusters X_i and X_j.

Function **InfoLoss**, given in Figure 3.8, computes the modified information loss when clusters X_i and X_j are merged. Without modifications, the information that is lost about C when merging X_i and X_j is equal to $I(X_i, C) + I(X_j, C) - I(\mathbf{Merge}(X_i, X_j), C)$. However, if such an information loss criteria is used, the algorithm most often would just merge the two largest clusters. This is because the larger the cluster is, the less information on average it provides about the class variable (with cluster variables having only two states, and function **Merge** defined as described above). And obviously the less information the cluster provides about the class variable, the less information can be lost when merging that cluster with other cluster. That is why we introduce the term $|X_i| + |X_j|$, which penalizes large clusters. Parameter α indicates how important cluster size penalty is compared to the standard information loss value. So,

Function **ClusterFeaturesAvg**$(\mathcal{D}, \mathcal{F}, B, \alpha)$:

1. Let $\mathcal{F} = \{F_1, \ldots, F_N\}$. Make initial clusters $X_i = \{F_i\}$, $i = 1, \ldots, N$. Let $X_i = 1$ iff $F_i = 1$. Let $\mathcal{A} = \{X_1, \ldots, X_N\}$.
2. From $\mathcal{D}$ calculate
   - $P(X_n = 1) = P(F_n = 1)$,
   - $P(C = c_m | X_n = 1) = P(C = c_m | F_n = 1)$,
   - $P(C = c_m | X_n = 0) = P(C = c_m | F_n = 0)$,
   - $P(C = c_m)$,

   $\forall n = 1, \ldots, N,\ m = 1, \ldots, M$, where C is the class variable.
3. Sort clusters $X_1, \ldots, X_N$ in descending order according to $I(X_i, C)$.
4. While $|\mathcal{A}| > B$, do
   (a) For each $\{X_i, X_j\} : X_i, X_j \in \mathcal{A}, X_i \neq X_j$ compute $loss_{ij} \leftarrow$ **InfoLoss**(X_i, X_j).
   (b) Select X_i, X_j that minimize $loss_{ij}$, and construct $X' =$ **Merge**(X_i, X_j). Remove from $\mathcal{A}$ clusters X_i, X_j, and add to $\mathcal{A}$ cluster X'.
5. Return $\mathcal{A}$.

Figure 3.6: Function ClusterFeaturesAvg

after introducing the penalty for large clusters we get a modified information loss $(I(X_i, C) + I(X_j, C) - I(\mathbf{Merge}(X_i, X_j), C)) + \alpha(|X_i| + |X_j|)$. The best value for the parameter α has to be determined experimentally. We would of course like α to be independent of the cluster size. However, as the clusters are being merged in the loop of step 4 of function **ClusterFeaturesAvg**, an average cluster size increases, and an average value for $I(X_i, C) + I(X_j, C) - I(\mathbf{Merge}(X_i, X_j), C)$ decreases while an average value for $\alpha(|X_i| + |X_j|)$ increases. That is why we multiply the first and divide the second term by an average cluster size k. There are no guarantees that these modifications are the best possible, but at least when using them better classifiers are produced than in the case of using standard information loss criteria.

Sub-function **Merge**(X_i, X_j):

1. Let $s_i = |X_i|, \ s_j = |X_j|$.
2. Set
   - $P(X' = 1) = \frac{s_i P(X_i=1) + s_j P(X_j=1)}{s_i + s_j}$,
   - $P(C = c_m | X' = 1) = \frac{s_i P(C=c_m|X_i=1) + s_j P(C=c_m|X_j=1)}{s_i + s_j}$,
   - $P(C = c_m | X' = 0) = \frac{s_i P(C=c_m|X_i=0) + s_j P(C=c_m|X_j=0)}{s_i + s_j}$,

   $\forall m = 1, \ldots, M$.
3. Set $|X'| = |X_i| + |X_j|$.
4. Return X'.

Figure 3.7: Function Merge

Sub-function **InfoLoss**(X_i, X_j):

1. Let $k = \frac{N}{|\mathcal{A}|}$.
2. Return $(I(X_i, C) + I(X_j, C) - I(\mathbf{Merge}(X_i, X_j), C))k + \alpha(|X_i| + |X_j|)\frac{1}{k}$.

Figure 3.8: Function InfoLoss

3.6 Feature Clustering Algorithm using OR

In this section we describe a feature clustering algorithm that uses the same criteria for cluster merging as **ClusterFeaturesAvg**, but the probabilities for the new clusters are defined by OR function. In algorithm **ClusterFeaturesAvg**, we defined just the probabilities $P(X')$ and $P(C|X')$ without giving semantics for the states of X'. In this algorithm we define that the cluster variable is in state 1 if and only if at least one of the feature variables that the cluster contains is in state 1. That is, we use OR function. Based on this, the probabilities $P(X')$ and $P(C|X')$ are calculated from the training data. The algorithm is given Figures 3.9 and 3.10.

Function **ClusterFeaturesOr**$(\mathcal{D}, \mathcal{F}, B, \alpha)$:
Same as function **ClusterFeaturesAvg**, but with different sub-function **Merge**.

Figure 3.9: Function ClusterFeaturesOr

Sub-function **Merge**(X_i, X_j):

1. Let $X' = 1$ iff $X_i = 1$ or $X_j = 1$.
2. From $\mathcal{D}$ calculate
   - $P(X' = 1) = P(F_{k_1} = 1 \vee \ldots \vee F_{k_l} = 1)$,
   - $P(C = c_m | X' = 1) = P(C = c_m | F_{k_1} = 1 \vee \ldots \vee F_{k_l} = 1)$,
   - $P(C = c_m | X' = 0) = P(C = c_m | F_{k_1} = 0 \wedge \ldots \wedge F_{k_l} = 0)$,

   $\forall m = 1, \ldots, M$, where $\{F_{k_1}, \ldots, F_{k_l}\}$, is the set of all the features contained by clusters X_i and X_j.
3. Set $|X'| = |X_i| + |X_j|$.
4. Return X'.

Figure 3.10: Function Merge (for ClusterFeaturesOr)

3.7 Feature Clustering Algorithm using Independence Tests

In this section we describe a feature clustering algorithm that uses an information about feature dependencies. Function **ClusterFeaturesDep**, given in Figure 3.11, takes the same arguments as function **ClusterFeatures** and additionally the mutual information importance parameter α.
ClusterFeaturesDep returns a partition $\{X_1, \ldots, X_B\}$ of $\mathcal{F}$. For each pair of features the probability of feature independence given the class variable is estimated. The initial experiments showed that measuring feature independence given the class variable instead of an unconditional feature independence gives slightly better classification.

For estimating $P(F_i \perp F_j \mid C)$ (the probability that F_i and F_j are independent given C) we use a χ^2 test for independence as described, for example, by Spirtes et al. [SGS93]. Let x_{abc} denote the number of cases in the training data where $F_i = a$, $F_j = b$, and $C = c$. Let $x_{+bc} = \sum_a x_{abc}$, $x_{a+c} = \sum_b x_{abc}$, and $x_{++c} = \sum_{a,b} x_{abc}$. Then using the hypothesis that F_i

and F_j are independent given C, we can compute the expected values of x_{abc} as $E(x_{abc}) = \frac{x_{a+c}x_{+bc}}{x_{++c}}$. Let $X^2 = \sum_{abc} \frac{(x_{abc}-E(x_{abc}))^2}{E(x_{abc})}$. If the independence hypothesis is true, the probability density function of X^2 converges to the probability density function of χ^2 distribution with $(|F_i|-1)(|F_j|-1)|C|$ degrees of freedom as the number of cases in the training data approaches infinity. Using this, we estimate $P(F_i \perp F_j \mid C)$. In our classifiers, the class variable C has always two states, so the χ^2 distribution with two degrees of freedom is used.

As in the two previous algorithms, each of the initial clusters contains one feature, and the new clusters are obtained by merging smaller clusters into larger ones. Clusters are merged by trying to put mutually dependent features into the same clusters. p_{ij} can be considered as a measure of distance between features F_i and F_j: the smaller p_{ij} is, the more dependent F_i and F_j are expected to be. That is why for any cluster X_k we want to minimize $\sum_{F_i,F_j\in X_k} p_{ij}$. We sum this over all the clusters. The total number of distances p_{ij} summed depends on the sizes of clusters. To get an average distance p_{ij} we divide the whole sum by the total number of pairs $\{F_i, F_j\}$ inside all the clusters. That is how we get the term $\frac{\sum_{X_k\in\mathcal{A}'}\sum_{F_i,F_j\in X_k} p_{ij}}{\sum_{X_k\in\mathcal{A}'}\binom{|X_k|}{2}}$ in Expression 3.1.

If feature clustering is performed by trying to minimize this term only, no distinction is made between features that provide a lot and features that provide little information about the class. As the initial experiments showed, the classifier performance suffered because features that had a high mutual information with the class variable were separated from the class variable by too many hidden variables. To overcome this problem, in Expression 3.1 we introduce the second term, where we penalize large clusters that contain informative features. For each feature its mutual information value is divided by the size of the cluster that the feature belongs to. On average, the sum $\sum_{i=1}^{N} \frac{I(F_i,C)}{|X_{k_i}|}$ is higher when features with high mutual information values belong to smaller clusters. And the smaller the cluster is, the less hidden variables on average separate its features from the class variable. Parameter α specifies how important this mutual information factor is compared to the goal of having dependent features in the same clusters. As in the two previous feature clustering algorithms, the best value for the parameter α has to be determined experimentally, and we would like α to be independent of the number of clusters. However, when the number of clusters decreases the sum $\sum_{F_i\in X_k} \frac{I(F_i,C)}{|X_k|}$ for any cluster X_k stays on average the same, but the number of such sums decreases. That is why we normalize by dividing the whole sum by the number of clusters $|\mathcal{A}'|$.

After step 3, where we obtain B clusters by merging clusters in a greedy way, we try to improve clustering by further minimizing Expression 3.1. Function **ImproveClustering** takes the current set of clusters and the

Function **ClusterFeaturesDep**$(\mathcal{D}, \mathcal{F}, B, \alpha)$:

1. Let C be the class variable. $\forall \{F_i, F_j\} \subset \mathcal{F}$ compute $p_{ij} \leftarrow P(F_i \perp F_j \mid C)$ from $\mathcal{D}$ by using χ^2 test for independence.
2. Make initial clusters $X_i = \{F_i\}$, $i = 1, \ldots, N$. Let $\mathcal{A} = \{X_1, \ldots, X_N\}$.
3. While $|\mathcal{A}| > B$, do
   (a) For each $\{X_a, X_b\} : X_a, X_b \in \mathcal{A}, X_a \neq X_b$ let $X' = X_a \cup X_b$, $\mathcal{A}' = \mathcal{A} \setminus \{X_a, X_b\} \cup X'$, and compute
   $$\frac{\sum_{X_k \in \mathcal{A}'} \sum_{\{F_i, F_j\} \subset X_k} p_{ij}}{\sum_{X_k \in \mathcal{A}'} \binom{|X_k|}{2}} - \alpha \frac{1}{|\mathcal{A}'|} \sum_{i=1}^{N} \frac{I(F_i, C)}{|X_{k_i}|}, \qquad (3.1)$$
   where X_{k_i} is the cluster that F_i belongs to.
   (b) Select X_a, X_b that minimize Expression 3.1. Remove from $\mathcal{A}$ clusters X_a, X_b, and add to $\mathcal{A}$ cluster $X_a \cup X_b$.
4. Let $\mathcal{A} =$ **ImproveClustering**$(\mathcal{A}, E_1)$, where E_1 is the current value of Expression 3.1.
5. Return $\mathcal{A}$.

Figure 3.11: Function ClusterFeaturesDep

current value of Expression 1 as the arguments. It tries to minimize Expression 3.1 by repeatedly moving one feature from its current to another cluster.

First, to speed up the calculations, we precompute d_{ik} – the sum of distances from feature F_i to all the features in cluster X_k. Then the loop is executed, where in each iteration we try to move one feature from its current to another cluster. We select those feature and the cluster to move to that minimize the value of the corresponding Expression 3.1. For each feature F_i that belongs to a cluster X_l with more than one feature in it and for each cluster X_m that is different from X_l we calculate E_{2m} – the value of Expression 3.1 if F_i were moved from X_l to X_m. In term $\frac{\sum_{X_k \in \mathcal{A}} \sum_{\{F_i, F_j\} \subset X_k} p_{ij} - d_{il} + d_{im}}{\sum_{X_k \in \mathcal{A} \setminus \{X_l, X_m\}} \binom{|X_k|}{2} + \binom{|X_l|-1}{2} + \binom{|X_m|+1}{2}}$ we compute the new average distance p_{ij}. Since we try to move feature F_i from X_l to X_m, in the numerator of this term we compute a new sum of distances by subtracting d_{il} from the current sum and adding d_{im} to it. When computing the new number of distances in

the denominator of the term, we decrease the size of X_l by one, and increase the size of X_m by one. In the rest part of E_{2m} we compute the new penalty term. $\sum_{\substack{j\in\{1,\ldots,N\} \\ F_j\notin X_l, F_j\notin X_m}} \frac{I(F_j,C)}{|X_{k_j}|}$ computes the penalty for all the clusters except X_l and X_m, $\sum_{F_j\in X_l} \frac{I(F_j,C)}{|X_l|-1} - \frac{I(F_i,C)}{|X_l|-1}$ computes the new penalty for X_l, and $\sum_{F_j\in X_m} \frac{I(F_j,C)}{|X_m|+1} + \frac{I(F_i,C)}{|X_m|+1}$ computes the new penalty for X_m.

The value of $gain_{im}$ indicates the gain in Expression 3.1 if F_i is moved to X_m. If the maximal $gain_{im}$ is positive, we move the feature F_i and accordingly update the values of d_{jl}, d_{jm} $(j = 1,\ldots,N)$, and E_1. We have set the maximum number of iterations for the "repeat" loop to be equal to the number of features N. In our experiments we have observed that $gain^*$ becomes less or equal to zero before N iterations are performed. In some cases strange $gain^*$ values were reported, which may be explained by a possible error in the program code for **ImproveClustering**.

Sub-function **ImproveClustering**$(\mathcal{A}, E_1)$:

1. $\forall F_i \in \mathcal{F}, \forall X_k \in \mathcal{A}$ compute $d_{ik} \leftarrow \sum_{F_j \in X_k} p_{ij}$.
2. Repeat
   (a) $\forall X_l \in \mathcal{A}, |X_l| > 1, \forall F_i \in X_l$
      i. $\forall X_m \in \mathcal{A} \setminus \{X_l\}$ compute

$$E_{2m} = \frac{\sum_{X_k \in \mathcal{A}} \sum_{\{F_i,F_j\} \subset X_k} p_{ij} - d_{il} + d_{im}}{\sum_{X_k \in \mathcal{A} \setminus \{X_l, X_m\}} \binom{|X_k|}{2} + \binom{|X_l|-1}{2} + \binom{|X_m|+1}{2}}$$
$$-\alpha \frac{1}{|\mathcal{A}|} \left(\sum_{\substack{j \in \{1,\ldots,N\} \\ F_j \notin X_l, F_j \notin X_m}} \frac{I(F_j, C)}{|X_{k_j}|} + \sum_{F_j \in X_l} \frac{I(F_j, C)}{|X_l| - 1} - \frac{I(F_i, C)}{|X_l| - 1} \right.$$
$$\left. + \sum_{F_j \in X_m} \frac{I(F_j, C)}{|X_m| + 1} + \frac{I(F_i, C)}{|X_m| + 1} \right).$$

      ii. Let $gain_{im} = E_1 - E_{2m}$.

   (b) Select F_i, X_m that maximize $gain_{im}$. Let $gain^*$ be the corresponding $gain_{im}$. If $gain^* > 0$,
      i. Move F_i from its current cluster X_l to cluster X_m.
      ii. $\forall F_j \in \mathcal{F}$ set $d_{jl} \leftarrow d_{jl} - p_{ji},\ d_{jm} \leftarrow d_{jm} + p_{ji}$.
      iii. Set $E_1 \leftarrow E_{2m}$.

   Until $gain^* \leq 0$ or the maximum number of iterations has been reached.

Figure 3.12: Function ImproveClustering

Chapter 4

Performance Experiments

In this chapter we describe the experiments performed. First we describe the test setup. After that we present the tests results.

4.1 Test Setup

In this section first we describe the text data that we use in our experiments. Then we describe how we index the documents in that data. Finally we mention the classifiers that we test in the experiments.

4.1.1 Data Used

In the experiments we test the classifiers on the Reuters-21578 text categorization test collection (Distribution 1.0), available at

`http://www.daviddlewis.com/resources/testcollections/reuters21578`.

It is currently the most widely used test collection for text categorization research. It contains about 20000 documents (the Reuters news stories), each of them assigned to zero or more classes. We use the most popular "ModApte" split of this collection into training and test sets. This split has been made according to the time of documents: stories in the documents from the training set appeared earlier than stories in the documents from the test set. Following the test setup of Yang and Liu [YL99], we select the classes that have at least one document both in the training set and the test set. It results in selecting 90 classes. After eliminating documents that do not belong to any of these 90 classes, we get a training set of 7769 documents and a test set of 3018 documents. We further split the training set into what we call a training-0 set with 5827 documents and a validation set with 1942 documents. This split is again made according to the time of documents. First we train the classifiers on a training-0 set and tune their

parameters based on the performance on a validation set. Then we learn the classifiers with the best parameters on the whole training set and run the final experiments on a test set. Because of too much time required to process data for a single class, we run the tests only for the 10 most frequent classes. These are earn, acq, money-fx, grain, crude, trade, interest, ship, wheat, and corn classes (given in the decreasing class frequency order).

4.1.2 Text Indexing

When indexing documents, for simplicity we do not distinguish between the text that appears in a title and the text that appears in a body of a document. First for the extraction of the features from the document we convert the text to lowercase and take from it words (i.e., sequences of alpha symbols delimited by any other symbols). Then, as described in Section 1.3, we remove function words. We also tried to perform stemming by using Porter [Por80] algorithm, but on the initial tests with a naive Bayes classifier this gave a slightly worse performance. So, no stemming is used. After this preprocessing each document has a number of features – the words that appear in the document and that are not function words. The feature set is then built by taking from the documents in the training data all the features except those that appear only in one document (because these are very unlikely to provide any information about the class). The final feature set consists of 15715 words. As in many other work on text categorization, we use binary features.

Since for many classifiers it is computationally impossible to use 15715 features, we perform local dimensionality reduction by feature selection, as described in Section 1.3. For feature selection we use an information gain criteria, because it has been reported as one of the most effective by Yang and Pedersen [YP97]. Unless mentioned otherwise, the default number of features used is 30. But for most of the classifiers we also perform the experiments with higher number of features. Both for the experiments on the validation and on the test data we perform feature selection based on the cases in the whole training data.

4.1.3 Classifiers Tested

Totally we test 8 types of classifiers. Namely, we test HBN classifiers that use that use probability average, OR, and (in)dependence test feature clustering algorithms. We call these classifiers correspondingly *HBN-AVG*, *HBN-OR*, and *HBN-DEP*. Also, we test HNB classifiers that use the same feature clustering algorithms. We call these classifiers *HNB-AVG*, *HNB-OR*, and *HNB-DEP*. We also test a naive Bayes classifier with the smoothing parameter N_0, described in Section 1.1.3, set to 0.1. We call this classifier *NB*. And we test the Support Vector Machines (SVM), because they have been re-

ported by Dumais et al. [DPHS98] to perform better than other methods for text categorization. We test linear SVM (i.e., SVM with polynomial kernels of degree 1, as described by Christianini and Shawe-Taylor [CST00]). For training SVM, the sequential minimal optimization algorithm is used, and the output of SVM is transformed into probabilities by applying a standard sigmoid function, as described in WEKA API documentation [WEK].

For implementing all the HBN and HNB classifiers we use HUGIN API V5.0 [HUG]. For implementing *NB* and *SVM* we use WEKA 3.2.1 [WEK]. We also use WEKA 3.2.1 for running tests.

4.2 Test Results

In this section we present the results of our tests. First we present the results of the experiments on the validation data and then on the test data. In most of the work in the area of text classification the micro-averaged breakeven point, as described in Section 1.3, is used to measure the classifier performance. We also use this measure to compare the classifier performance on the validation data. But in the experiments on the test data for comparison we also report the macro-averaged breakeven point for each classifier. In all the tables, we report the breakeven point as a percentage. In the end of the section we give the examples of the HBN classifier structures learned by our feature clustering algorithms.

4.2.1 Experiments on the Validation Data

HBN Classifiers

We use the validation data to tune the branching factor B and the parameter α (which is the cluster size penalty parameter or the mutual information importance parameter) for each of *HBN-AVG*, *HBN-OR*, and *HBN-DEP* classifiers separately. First we take $B = 7$ and test the classifier with the different values of α. Then we take α that gave the best performance and test the classifier with values 3, 5, 7, and 9 for B. Then we take B that gave the best performance as the value to be used in the experiments on the test data. For this B we again test the classifier with the different values of α. This time we take values of α that are from shorter interval and closer to each other than in the first phase of tuning α. We take α that gave the best performance as the value to be used in the experiments on the test data. In Table 4.1 we present the final B and α values for the HBN classifiers.

We have also tried the HBN classifiers with 60 features. In Tables 4.2, 4.3, and 4.4 we present the performance of the HBN classifiers with 30 and 60 features with $B = 7$ and different values of α (the first phase of the parameter tuning). The performance of *HBN-AVG* and *HBN-OR* decreased when more features were added, and the performance of *HBN-DEP* stayed

Classifier	B	α
HBN-AVG	7	0.05
HBN-OR	9	0.09
HBN-DEP	7	170

Table 4.1: The Final Parameter Values For the HBN Classifiers

similar. That is why we do not test the HBN classifiers with 60 features any further.

Value of α	30 features	60 features
0	78.8	78.9
0.03	80.8	75.9
0.06	80.7	79.6
0.09	80.6	76.3
0.12	80.2	77.7

Table 4.2: The Performance of *HBN-AVG* with Different Number of Features

Value of α	30 features	60 features
0	80.1	79.7
0.05	80.4	78.8
0.10	80.1	78.1
0.15	79.8	77.7
0.20	79.8	75.5

Table 4.3: The Performance of *HBN-OR* with Different Number of Features

Value of α	30 features	60 features
0	77.1	76.3
50	78.9	80.8
100	80.6	80.2
150	81.0	80.6
200	80.6	80.5
250	80.6	80.7
300	80.3	80.9

Table 4.4: The Performance of *HBN-DEP* with Different Number of Features

HNB Classifiers

In Tables 4.5, 4.6, and 4.7 we compare the performance of the HNB (hierarchical naive Bayes) classifiers and the corresponding HBN (hierarchical Bayesian network) classifiers when $B = 7$ and the value of α varies (the first phase of the parameter tuning). The HNB classifiers performed worse than the corresponding HBN classifiers. That is why we do not test the HNB classifiers any further.

Value of α	*HNB-AVG*	*HBN-AVG*
0	78.8	78.8
0.03	77.6	80.8
0.06	78.2	80.7
0.09	77.5	80.6
0.12	76.9	80.2

Table 4.5: The Performance of *HNB-AVG* Compared to *HBN-AVG*

Value of α	*HNB-OR*	*HBN-OR*
0	76.2	80.1
0.05	76.9	80.4
0.10	77.7	80.1
0.15	77.5	79.8
0.20	76.5	79.8

Table 4.6: The Performance of *HNB-OR* Compared to *HBN-OR*

Value of α	*HNB-DEP*	*HBN-DEP*
0	76.7	77.1
50	75.0	78.9
100	76.0	80.6
150	75.4	81.0
200	74.4	80.6
250	75.0	80.6
300	75.0	80.3

Table 4.7: The Performance of *HNB-DEP* Compared to *HBN-DEP*

A Naive Bayes Classifier

In Table 4.8 we present the performance of *NB* with the different number of features used. For the experiments on the test data, we select *NB* with 30 features (to make a comparison with other classifiers that use 30 features) and *NB* with 100 features (because it performed best on the validation data).

Number of features	Performance
30	78.3
60	79.1
100	79.5
200	78.7
300	78.8
400	78.3
500	77.9

Table 4.8: The Performance of *NB* with Different Number of Features

The Support Vector Machines

In Table 4.9 we present the performance of *SVM* with the different number of features used. For the experiments on the test data, we select *SVM* with 30 features (to make a comparison with other classifiers that use 30 features) and *SVM* with 200 features (because it performed best on the validation data). In the tests presented in Table 4.9, the complexity constant parameter of *SVM*, described in WEKA API documentation [WEK], is tuned for the classifier with 30 features. Before using *SVM* with 200 features in the experiments on the test data, we separately tune the complexity constant parameter.

Number of features	Performance
30	83.3
60	85.1
100	87.4
200	87.5
300	86.7
400	86.5
500	86.1

Table 4.9: The Performance of *SVM* with Different Number of Features

4.2.2 Experiments on the Test Data

In Table 4.10 we present the classifier performance on the test data. For each classifier we present the breakeven performance on each of 10 classes, and also the micro-averaged and macro-averaged breakeven. When all the classifiers use 30 features, the micro-averaged breakeven performance of the HBN classifiers is about 2% better than the performance of *NB* and about 2% worse than the performance of *SVM*. If *NB* and *SVM* use more features, the micro-averaged breakeven performance of the HBN classifiers is only slightly better than the performance of *NB* and about 4% worse than the performance of *SVM*. When the macro-averaged breakeven performance is measured, the HBN classifiers are about 6% better than *NB* and about 2% worse than *SVM*. This means that, compared to the other classifiers, *NB* performs much worse on the classes with few positive instances.

Among the HBN classifiers, *HBN-OR* is slightly better than *HBN-AVG* and *HBN-DEP*.

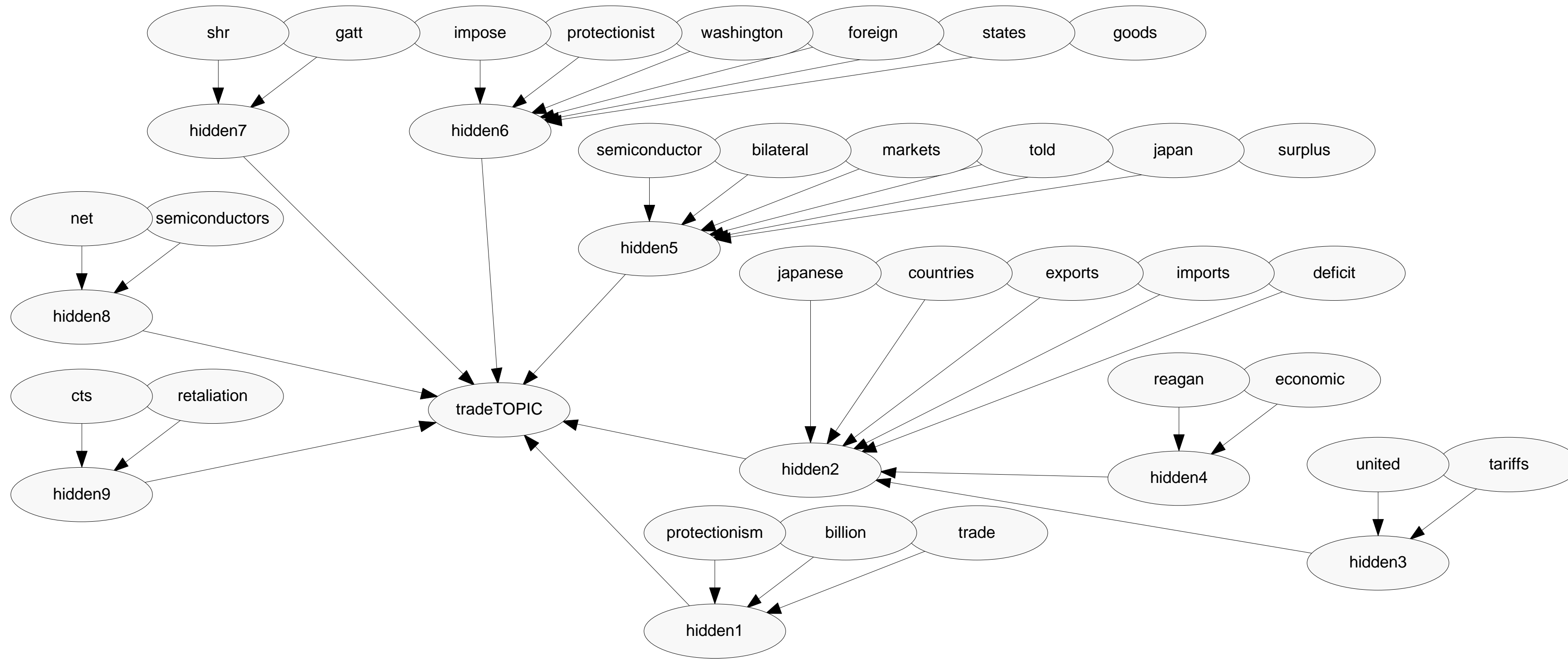
Running time for different algorithms is quite different. Learning and to testing of the classifier for one class takes about 1 minute for *NB*, about 10 minutes for *SVM*, and about 30 minutes for the HBN classifiers. For the HBN classifiers, most of the time is spent on learning the conditional probabilities by using the EM algorithm.

4.2.3 Feature Clustering Algorithms

In Figures 4.1, 4.2, and 4.3 we depict the *HBN-AVG*, *HBN-OR*, and *HBN-DEP* classifiers learned during the experiments on the test data for class trade. In *HBN-AVG*, all the parents of the class variable ("tradeTOPIC") are hidden variables. In *HBN-OR*, the variables "trade" and "tariffs", which have the highest information gain values, are made directly the parents of the class variable. The features inside the clusters seem to be more similar than in the case of *HBN-AVG* classifier. In *HBN-DEP*, the informative feature variables are put as close to the class variable as possible. The class variable has only one hidden variable as its parent, all the other its parents are feature variables. Similarly, hidden variables have mostly features as their parents, and only "hidden2" has two hidden variables as its parents. So, the second term from Equation 3.1 clearly dominates. For the comparison, in Figure 4.4 we depict the *HBN-DEP* classifier learned during the experiments on the validation data for class trade with the parameter α set to 0. That is, only the first term from Equation 3.1 is taken into account when constructing the classifier. This classifier performed worse than the one where the second term from Equation 3.1 has more impact. However, the way features were clustered seems to be very similar to how a human would do it if the criteria was the similarity of features.

Class	*HBN-AVG*	*HBN-OR*	*HBN-DEP*	*NB* (30 feat.)	*NB* (100 feat.)	*SVM* (30 feat.)	*SVM* (200 feat.)
earn	95.4	95.8	95.1	95.2	96.7	95.8	98.1
acq	86.7	85.2	84.6	86.6	89.4	87.6	93.7
money-fx	59.9	60.5	57.1	59.2	61.3	60.3	66.5
grain	83.2	84.2	86.9	73.3	75.6	92.0	85.9
crude	75.5	79.9	74.1	77.6	83.1	78.4	81.5
trade	64.1	65.8	68.4	57.3	54.7	67.5	70.1
interest	62.6	63.4	59.0	61.6	58.6	67.9	65.6
ship	79.8	80.9	79.2	80.9	80.9	83.0	69.7
wheat	85.9	87.3	87.3	63.4	71.5	90.7	85.9
corn	85.7	85.7	89.3	58.9	57.0	88.8	83.9
Micro-averaged	85.1	85.4	84.5	83.1	84.4	86.9	88.9
Macro-averaged	77.9	78.9	78.1	71.4	72.9	81.2	80.1

Table 4.10: The Performance on the Test Data



shr
gatt
impose
protectionist
washington
foreign
states
goods
hidden7
hidden6
semiconductor
bilateral
markets
told
japan
surplus
net
semiconductors
hidden5
japanese
countries
exports
imports
deficit
hidden8
reagan
economic
cts
retaliation
tradeTOPIC
hidden2
hidden4
united
tariffs
hidden9
protectionism
billion
trade
hidden3
hidden1

ERROR: rangecheck
OFFENDING COMMAND: xshow

STACK:

[84 46 46 93 93 84 93 139 93 93 46 84 46 93 93 93 46 84 93 46 46 37 93
93 84 46 93 83 46 37 84 46 93 93 84 84 46 93 93 46 93 83 93 46 93 93 0]
( )
-mark-
5
-savelevel-