An information theoretic approach to finding word groups for text classification

Keywords: Text classification, MDL principle, Feature extraction

Institute for Language, Logic and Computation

J.J. Verbeek

September 11, 2000

This document is a masters thesis for the 'Graduate Program in Logic' at the Institute for Logic, Language and Computation (ILLC), an institute of the University of Amsterdam (UvA). The thesis will be defended September 29$^{\text{th}}$ 2000 at 16:00 in room P.327 at the ILLC.

The author was happy to receive supervision of Nikos Vlassis (postdoc at the Intelligent Autonomous Systems group at the Faculty of Science of UvA) and Michiel van Lambalgen (ILLC staff member).
I would also like to thank Peter Grünwald for providing many detailed comments on a draft version of this thesis and useful hints and discussions. Thanks are also due to Ben Kröse, since he was able to provide some equipment to conduct a part of the experiments. I thank Christof Monz and Jon Ragetli for inspiring conversations and useful hints in the literature. Finally, Kristel deserves many thanks for reducing my doubts and worries and keeping me to work.

The author can be contacted at: `jverbeek@science.uva.nl`.

Institute for Logic Language and Computation
Plantage Muidergracht 24
1018 TV Amsterdam
The Netherlands

Orchidee

Zij is niet alleen mooi en zuiver,
maar ze geurt in een kamer voor iedereen.

Een orchidee bloeit langzaam
en leeft van het zuiverste water en de puurste lucht.

Ze ontluikt teder, loom.

Een orchidee kan niet ten oorlog trekken.
Ze kan geen bruggen bouwen, noch door de lucht vliegen.

En toch kunnen mensen met al hun knapheid en ontwerpen haar schoonheid
niet namaken, noch haar geur vervaardigen.

Abstract

This thesis concerns finding the 'optimal' number of (non-overlapping) word groups for text classification. We present a method to select *which* words to cluster in word groups and *how many* such word groups to use on the basis of a set of pre-classified texts. The method involves a greedy search through the space of possible word groups. The criterion on which is navigated through this space is based on 'mutual information' and is known as 'Jensen Shannon divergence'. The criterion to decide *which number* of word groups to use is based on Rissanen's MDL Principle. We present empirical results that indicate that the proposed method performs well at its task. The prediction model used is based on the Naive Bayes model and the data set used for the experiments is a subset of the 20 Newsgroup Dataset.

Contents

List of Figures

Chapter 1

Introduction

In this first chapter we discuss the topic of this thesis without going into technicalities. Furthermore, we present the research questions and motivation of this project and finally we outline the organization of the rest of this thesis.

1.1 Sketch of the landscape

This thesis is dealing with 'feature extraction' for 'classification' of texts. Moreover, the classification is set in the 'learning from examples' paradigm. These terms form research areas in themselves. They are part of the field of 'Artificial Intelligence'. *Artificial Intelligence* is a term capturing many fields in and between computer science, psychology and philosophy. Common in most topics in Artificial Intelligence are problems in which knowledge and information play an important role. Either information is vague (uncertain or imprecise) or there might be overwhelming amounts of information. In both cases it can be problematic to extract knowledge, represent knowledge and reason with knowledge. Often these problems are interconnected.

Classification and learning By *classification* we mean the following: we know an object belongs to one of several classes. The task is to decide to which of these it belongs. The decision is made using a mapping M, such mappings are also refered to as 'models' or 'hypotheses' or 'classifiers' in the Machine Learning literature. The mapping M assigns to an object a probability distribution on the classes. An important class of such mappings are those which concentrate all probability mass on just one class, i.e. the mapping just gives a 'guess' about the class. The term *Learning from examples* (also known as 'supervised learning') is used for situations in which a *learner* is presented with a set of objects (called the *examples*, *sample* or *training data*) for which the classes they belong to are given. The task is to choose a 'good' or 'the best' model from a set of models. The set of models is usually called the *model class*. The model that is picked by the learner is then used to make predictions for new data. We depicted the situation in Figure 1.1, the dashed box can be thought of as the model that is selected by the learner and used for predictions.

Text classification In this thesis we are dealing with learning from examples applied to text classification. The goal is the following: given a set of m texts $d_1, \dots, d_m$ together with their class $c_1, \dots, c_m$, predict the label c_{new} of a new, i.e. for which the label is unknown, text d_{new}. In our case, the prediction is made on the basis of a probabilistic model. In this thesis, the documents are represented as 'a bag of words', i.e. the only information we use is how often each word occurs

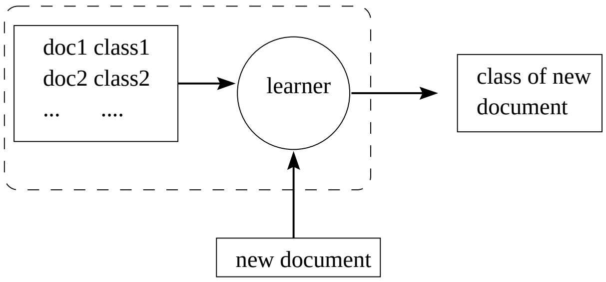

Figure 1.1: Learning from examples: the learner uses the pre-classified examples to obtain a model, which is in turn used to predict the class of new examples.

in a text. The prediction of the class label c_{new} is based on how often each word occurs in the text d_{new}. Sometimes, other representations are used. For example, one could count how often certain phrases (sequence of words) occur in a text. If we know that the documents always conform to a certain structure (like research papers: first there is a title, followed by the author, an abstract, etc.), we could use this structure in the classification. The quality of a model is evaluated by testing the predictive performance of the model on a set of test texts, which we call the *test set.* We use the so called 'zero-one' loss in the evaluation: if a text is correctly classified then it contributes zero to the total loss, other wise it contributes one. Then, optimal performance is defined in terms of minimal total loss.

Feature extraction Sometimes in classification tasks, the number of properties (or *features*) of the examples is so great that the computation needed for prediction becomes too intensive. Certainly when high classification speed is demanded by the application, we want to reduce the computational cost of prediction by reducing the number of features. More important, modeling data that is described with many features forces us to estimate many parameters (usually the number of parameters is related linearly to the number of features). However, the more parameters we have to estimate from a given body of data, the less robust these estimates become. Or: the more parameters we have to estimate, the more data is needed to provide good estimates. The fact that using more properties of the data in our model may not just improve our model, but may actually force us to obtain more data is known as "the curse of dimensionality". It would be a good idea to reduce the number of features so we have to estimate fewer parameters and we are able to estimate them more robustly.

Feature extraction is the process of extracting features (fewer than the original number) from a set of original features. An intuitive idea is to look which features are redundant, in the sense that they are highly inter-related with other features, and remove them. In fact, in this case we are merely *selecting* features, therefore the term *feature selection* is more appropriate here.

Feature extraction can also be seen as projecting the space spanned by the original features to a subspace spanned by new features. Much work has been done on finding good projections for data. Maybe the most widely known method is Principal Component Analysis (PCA) [15], which finds the projection that maximizes the statistical variance in the projected data. In this thesis, we use a method for feature extraction that is aimed at finding projections that are optimal for the classification task following the feature extraction. In the next chapter we will come back to feature extraction.

Information theory In this thesis we use an approach for feature extraction for text classification that is motivated by information theory. Information theory is a field on the border of statistics and computer science and physics, that describes fundamental properties of information based on classical probability theory. Topics include: transmission and compression of data, measures for uncertainty etc. We use some basic notions from this field, for which we provide definitions in Appendix A.

The model selection method (the method by which the learner picks a model) used in this thesis is the Minimum Description Length (MDL) principle. The MDL principle is founded on information theory and can be succinctly described as: "picking the model that compresses the pre-classified examples the most (and hence yields the minimum description length)". We provide a brief introduction to the MDL principle in Section 2.2.

Now that we have discussed the setting, we are ready to present in some more detail exactly what we will do in this thesis.

1.2 Goals and motivation

In this thesis we extend a method for feature extraction for text classification. Our extended method extracts the 'optimal' number of features (dimensionality of the feature space) for classification on the basis of a set of pre-classified documents. The original method proposed features *given* the desired number of features.

Aim of the project The question of *how many* features to use is often not treated in feature extraction. Sometimes a simple method like Cross Validation [4] is used to answer it. Other ways to determine the number of features to extract include: setting a limit on the decrease in prediction performance of the resulting models or setting a limit on the decrease in mutual information. However, how these limits should be set is again a difficult question and there is no general answer.

Our method to determine the number of features provides an alternative for cross validation. It has the appealing property that while proposing far less features than cross validation the predictive performance of our method was just slightly worse in our experiments. As original features individual words occurring in the texts are used. New features are formed by non-overlapping 'groups' or 'clusters' of several words. As prediction model we used the Naive Bayes[1] model, that assumes all words in a document to be independent given the class the document belongs to. We present a method to determine:

1. How many word groups should be used.
2. How the words should be grouped into these groups.

The approach Our approach to tackle these problems is to search for the word clusters step by step using a greedy search algorithm through the space of possible clusterings. To use a greedy algorithm we need a criterion to be optimized. We use a quantity related to the decrease in quality of prediction on the training data due to a combination of two words. Since we are not really interested in predictions on the *training* data but rather in predictions on *new* data, we run the risk of 'overfitting'. By *overfitting* we mean that a model is not only capturing the regularities in the data resulting from the phenomenon that is modeled, but it is also describing the (uninteresting) aspects of the data that are due to (measuring) noise. 'Simple' models are less 'risky' for prediction because they are less likely to 'overfit' the

[1] This method is also known as *simple Bayesian classifier* or *idiot Bayes*.

data. Simple models are less likely to overfit the data than complex models because simple models have, by definition, less descriptive power. The idea is that simple models will 'spend' all their descriptive power on the regularities in the data that originated from the phenomenon that is modeled. Very complex models would have descriptive power 'left' to describe the noise. We introduce a *penalty term* in the evaluation function that is high for complex models and low for simple models. Thereby, we introduce a preference for models making predictions based on only a few word groups.

The 'penalty term' may seem very 'ad hoc' in the above. However, the theory of Rissanen's Minimum Description Length principle [29] provides a well founded theoretical basis to compute such a penalty term. Using this penalty term, we have a criterion to compare models that base their predictions on different numbers of word groups.

Our algorithm performs a combination of two word groups at each step and then compares the model on the basis of performance and the penalty term to the other models encountered by the algorithm. Finally, it picks the model that optimized the used criterion.

Questions Besides proposing a framework for computing the optimal number of word groups and how words should be assigned to these groups, in this thesis we also try answer the following questions:

1. How does the performance of a model change as a function of the number of word groups used?

2. How does the performance of the model proposed by the algorithm compare to the performance achieved by other numbers of word groups?

3. How does the number of word groups change as a function of the number of training texts?

We answered these questions by conducting several experiments with an implementation of the proposed algorithm.

1.3 Overview

The rest of this document is organized as follows: We give a short introduction into Feature Extraction and the Minimum Description Length Principle in Chapter 2. Then, in Chapter 3 we discuss text classification and the Naive Bayes model used to do the text classification here. Chapter 4 deals with applying the MDL principle to this problem domain. Then, in Chapter 5 we present the implemented algorithm. Finally, in Chapter 6, we present our experimental results and the conclusions we draw from them. Appendix A provides definitions of some information theoretic concepts and in Appendix B we discuss an alternative method for the encoding of clusterings.

Chapter 2

Feature extraction and the MDL Principle

In this chapter we give brief introductions to feature extraction and the Minimum Description Length principle. We will also briefly discuss codes, which are needed to apply the Minimum Description Length Principle.

2.1 Feature extraction

Feature extraction is the process of extracting 'good' features from data, where *good* has to be specified depending on the task at hand. We treat data as individual observations consisting of the values of one or more *attributes*. For example, we measure the weight and length of fish. Then one particular fish is one observation with attributes 'weight' and 'length'. Sometimes, the data is said to be located in the *feature space*, this is the space spanned by the attributes of the data. In our case it is the two dimensional plane in which each point is a combination of a certain weight and length. The problem of feature extraction can be seen as the problem of finding the 'best' (or a 'good') projection of the data onto some subspace of the feature space. For example, a line in the weight-length plane, see Figure 2.1.

Unsupervised methods There exist several methods to do feature extraction. We make a distinction here between 'supervised' and 'unsupervised' methods. *Supervised* methods make use of a 'labeling' (a 'label' is assigned to each observation) of the data, which is a property of the data of special interest: we want to predict the label for new data. In our example, the label could be the type of fish that was observed or the market value of the fish. In contrast, 'unsupervised' methods do not take into account such a special variable. For example: Principal Component Analysis (PCA) (the most common unsupervised method) projects the data on a (sub)space such that the variance in the projected data is maximized. Consider the following example: Figure 2.1 shows how PCA would project the two dimensional data (the dots) on a one dimensional space (the line). The left plot shows the case for unclassified data: each dot represents a measurement and PCA maps them to the depicted line which summarizes the data nicely. Now, suppose the data is 'labeled' and given by $\{(x_1, y_1), \ldots, (x_m, y_m)\}$ where $x_i \in \mathbb{R}^2$ and $y_i \in \{0, 1\}$. In the right plot of Figure 2.1 this case is depicted; a black dot corresponds to $y = 0$ and a white dot corresponds to $y = 1$. The plot shows the danger of using unsupervised methods when feature extraction is intended for classification: the projection maximizing the variance over all the data in the feature space does not have to be a projection that allows good discrimination between the classes.

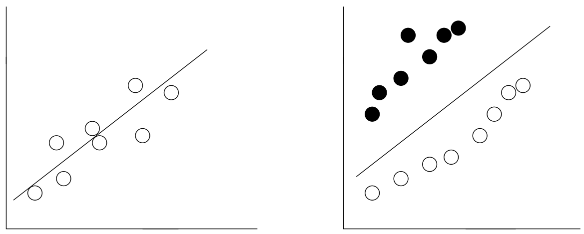

Figure 2.1: Two examples of PCA projection. In the right graph, data from two 'classes' is depicted: the 'white' class and the 'black' class.

Feature extraction for classification: supervised feature extraction If the task following the feature extraction is classification, we want a projection such that the quality of prediction is as good as possible. We could use the class information provided with the examples to guide our search for good projections. If a feature extraction method uses this information we call it *supervised*.

In Section 4.3 we show how we arrive at 'Mutual Information' (see appendix A for definition) as an interesting measure to guide the search for projections if we try to maximize predictive power. *Mutual Information* is a symmetric measure between variables. It expresses the amount of information the variables bear on each other. That is: the higher the mutual information, the more it helps us to know the value of one variable if we have to predict the value of the other. In fact, mutual information measures how many bits we save on average if we encode the value of one variable using the knowledge of the value of the other variable, assuming we use the (idealized) Shannon-Fano code ([5], Chapter 5) to encode the values of the variables.

Determining the optimal dimensionality of the subspace to which we are going to project (or: the number of features to extract) is one of our goals. Therefore, we need a criterion that lets us compare projections of different dimensionality. Mutual information alone does not provide us with such a criterion. We use the Minimum Description Length principle for this purpose. It will be discussed in the next section. There are situations, however, in which we *do* have an idea of the intrinsic dimensionality of the phenomena (maybe by experience in the domain of application). In such cases we can use that dimensionality in the feature extraction process.

Text categorization feature extraction methods Before we turn to the Minimum Description Length principle, we list a few feature selection methods commonly used in text categorization problems.

- The boldest among the methods is *document frequency thresholding*: For each word the fraction of documents in which it occurs is computed. Very rare words are removed since they are assumed to be either non-informative or not influential in global performance. Also, very frequent words are removed as they may not be informative, since they do not convey enough semantic content (in the sense of being indicative for the topic of the document), think of words like "a", "the", etc. This method is computationally very efficient and can be easily used for large corpora. If we want to reduce the feature, we exclude (very) frequent words and (very) rare words. However, if we want to reduce the number of words really drastically (say from several thousands to less than 100), we can only allow words with a frequency of occurrence from a very small 'band' (range of frequencies). How to choose a good band is not clear, and it will probably result in removing informative words, since it will probably remove 'not so frequent but highly informative words'. It is clear that this is an unsupervised method that should be used with care! We feel that this method should only be used to remove words with extreme document frequencies (near 100% or near 0%) to avoid discarding usefull information.

- *Information gain* is a term used for an evaluation function of words that could (in our view) better have been named *mutual information*, as will be made clear below. Following [37], let C denote the set of classes and w the event that the word under consideration occurs in a document and $\bar{w}$ the event that it does not. We assign to w the value $G(w)$, as defined below. If we want to select k words, we pick the k words with the highest $G(w)$.

$$G(w) = -\sum_{c \in C} p(c) \log p(c)$$
$$+p(w) \sum_{c \in C} p(c|w) \log p(c|w) + p(\bar{w}) \sum_{c \in C} p(c|\bar{w}) \log p(c|\bar{w}) \tag{2.1}$$

  The probability distribution p is the empirical distribution induced by the pre-classified documents. The reader with knowledge of information theory will immediately recognize equation (2.1) as the mutual information $\mathcal{H}(C) - \mathcal{H}(C|W) = I(C;\ W)$ [1] between the variables C' and W, where C' can take the classes as values and W can take the values w and $\bar{w}$. Therefore, we think it is more appropriate to call this criterion "mutual information". Note that this criterion (i.e. the function $G(w)$) does not take into account *how often* words occur in the training documents, the criterion discard this information and just uses *whether* a word occurs in a training document.

- In [37] there is a criterion called *mutual information*. However, we feel it really deserves another name, because it is based on values that are not the mutual information we know from information theory. Let

$$I(w, c) = \log \frac{p(c|w)}{p(c)}$$

  Two alternative functions of w are mentioned:

$$I_{avg}(w) = \sum_{c \in C} p(c) I(w, c) = -D(p \parallel p_w)$$
$$I_{max}(w) = \max_{c \in C} \{I(w, c)\}$$

  With $p_w(\cdot)$ we denote the conditional distribution $p(\cdot|w)$ over the classes and $D(\cdot \parallel \cdot)$ stands for the relative entropy (see appendix A for a definition). Again, the probability distribution p is taken to be the empirical distribution corresponding to the sample and the highest 'scoring' words are selected. Note that these functions do not take account of how often a word occurs in the training sample. Therefore, a very rare word w_1 will be preferred over a word w_2 that occurs many more times but has a just slightly lower I value. This is not advisable since we will 'benefit' more from retaining w_2 since it is probable that we will encounter it more often in the future.

In [37] the above mentioned methods are compared, together with a few other methods. It turns out that the 'information gain' method outperforms the other methods when the number of features is reduced 'aggressively' i.e. reduced to 300 features or less. The method called 'mutual information' is performing uncomparably more poorly in the survey, this is in line with our doubts about this measure.

[1] $\mathcal{H}(\cdot)$ denotes the entropy function, see appendix A.

In practice In practical applications one typically the following three steps. Document frequency thresholding is used to quickly remove spelling mistakes and other very rare words. Also, *stemming* is an important feature extraction method generally used. It makes all words stemming from the same verb identical. This reduces the number of word considerably while retaining most semantic (in the same sense as before) information. Finally, 'stoplists' are used. With the term *stoplists* we refer to lists of words like "a", "the" etc. that convey very little semantic information. It is common practice to use a stoplist, i.e. disregard all words in the texts that occur on the stoplist. Note the relation between stoplists and document frequency thresholding where all words with frequency *above* the threshold are removed.

2.2 The Minimum Description Length principle

In this section we give an informal introduction to the Minimum Description Length (MDL) principle and discuss some technical details we will need later. For a more formal and extensive treatment of the MDL principle we refer to [14, 29, 34].

2.2.1 An introduction

Model selection methods Rissanen's Minimum Description Length [29] principle is one of the many known 'model selection methods'. A *model selection method* is a method to choose among different possible models to explain some given data. Note that often we can divide a set of models into several classes of different complexity. For example, if $\mathcal{M}$ is the set of all polynomials, we can represent $\mathcal{M} = \bigcup_i \mathcal{M}_i$ as the union of a set of disjunct model classes $\mathcal{M}_i$. In general it is not simple to determine from which class $\mathcal{M}_i$ we should pick a model, recall the phenomenon of 'overfitting' discussed in Section 1.2 on page 3. Model selection methods provide an answer to this question, i.e. they make it possible to compare models of different complexities.

Now, we discuss two 'generic' model selection methods as examples. *Maximum Likelihood* (ML) is a model selection method that tells us to pick $M \in \mathcal{M}$ such that M mimics the training data the best. So, if there is noise in the data M will in general also mimic the noise. Note that the 'complexity' of the models is not taken into account and $\mathcal{M}$ is regarded as one set rather than the union of several sets $\mathcal{M}_i$. *Cross Validation* divides the training data into two sets: a 'training set' and a 'test set'. First, we find the ML model $\hat{M}_i \in \mathcal{M}_i$ over the training set for all i. Then, we measure the performance (according to some 'loss function') of each model on the test set. Cross validation tells us to pick the ML model $M_{CV} \in \mathcal{M}_i$ over *all the data* for the i that performed best on the *test* data. The idea is that too complex models will perform poorly on the test set, since they mimic the noise in the training data. In [34] several model selection methods are discussed in detail: Cross Validation, Guaranteed Risk Minimization, Predictive MDL and Two-part MDL.

We could pose the problem of model selection as follows: "Which model receives the highest probability, based just on observing this data, to have generated the data?" The MDL principle uses notions from information theory to restate this question as "Which model allows the greatest compression of the data?" Rissanen answered this question in two ways in [29] resulting in two model selection methods: Two-Part MDL and Predictive MDL. We discuss both methods below.

Two-Part MDL If we want to 'model' or 'describe' a given body of data[2], we may do so by first encoding (describing) a model, and then encode the data *using* this model. This results in a code length (measured in numbers if bits) given by equation (2.2).

$$L_H(D) = L(H) + L(D|H) \tag{2.2}$$

The first term in the sum is called the *complexity term* or *penalty term* (since it penalizes complex models), the second the *error term*. Simple models can be encoded using few bits and for complex models we need many bits. Furthermore, if a model fits the data very well, we need few bits to encode the data using the model (since the data deviates little from the 'predictions' made by the model) and vice versa. For example, for the data in the left plot of figure 2.1 on page 6 the PCA line seems a fine model: the line (given by just two parameters) summarizes the general structure of the data while some aspects of the data are treated as noise that was due to the specific data set and not due to the underlying phenomenon that is modeled.

The sum in (2.2) is called the total code length. Given a set of candidate models $\mathcal{M}$, MDL selects the model, H, with the *minimal total code length* or equivalently the model $H \in \mathcal{M}$ minimizing equation (2.2). We can also say: it selects the model yielding *maximal data compression*. The theory behind the MDL principle (depending largely on Kolmogorov Complexity [22]) provides us with a theory that shows that the model that yields maximal data compression will, under quite general circumstances, eventually (if enough examples are presented) be the best predicting model. Model selection methods, like Two-Part MDL, evaluating models on the basis of a sum of an error term and a penalty term as in equation (2.2) are said to be *penalty based*.

Summarizing: Two-Part MDL compares models on the basis of a sum of two terms. One term represents how well the model fits the data, the other term represents the complexity of the model. Two-Part MDL tries to find a good trade-off between between complexity and fit.

Predictive MDL The Predictive MDL (PMDL) technique was introduced in [29]. The same technique was also discovered by Dawid in 1984, who called it *prequential forecasting* [6]. Dawid, however, did not link the idea to code lengths.

To motivate PMDL we note that the codes used with Two-Part MDL are inherently inefficient if we want to encode a given body of data with the help of a model *class*. To see this, note that for each possible data there are as many code words as there are models in the model class $\mathcal{M}$ (since we can encode data D by first encoding model H and next encode encode D with respect to H). If we use a coding scheme that assigns just one code word to each data, we can assign short code words to more data sequences.

To obtain such a code we use a simple coding scheme, for clarity we present it in the program in Figure 2.2. The reader unfamiliar with PMDL is advised to study the figure before reading the rest of this paragraph.

The idea is the following: First, encode the first example. Then, for all $i \in I$ (with I the index set of all model*classes*), and $j = 2, \ldots, m$ (m is the number of examples) encode y_j on the basis of the ML model $\hat{M}_{i,j-1} \in \mathcal{M}_i$ over the previous $j-1$ examples. The encoding is done on the basis of the Shannon-Fano code (see next subsection) corresponding to the model; where, for simplicity, we assume that there is a probability distribution corresponding to the model. Then, we pick model class $\mathcal{M}_i$ that minimizes the code length.

[2]The data is assumed to be discrete, which makes sense at least as long as we are using discrete computers

INPUT:
A sequence $S = \{(x_1, y_1), \ldots, (x_m, y_m)\}$
and a collection of model classes $\mathcal{M}_i, i \in I$
OUTPUT:
The index i of the model class allowing the shortest encoding of the data.

```
begin
  - set total_i to zero for all i ∈ I.
  for j = 2 to m do
     for i ∈ I do
        - compute ML model p̂_i over {(x_1, y_1), ..., (x_{j-1}, y_{j-1})}
        - compute number of bits l_{i,j} needed to encode y_j using p_i
        - add l(i, j) to total_i.
     end
  end
  - return i such that total_i = min({total_k | k ∈ I}
```

Figure 2.2: Pseudo code for applying Predictive MDL.

Note, that PMDL prevents us from having to find a 'good' coding scheme to encode the models: the used models follow from the modelclass and the observed data sequence. Using Two-Part MDL we do have to find an efficient coding scheme to encode the models. There are not always clear measures that indicate whether one uses the 'right' coding scheme. This can be problematic in practice.

However, the estimation of the ML models and the prediction we make with the ML model to apply PMDL can be computationally very costly and therefore not desired. This is also the case in our application due to the large samples (up to 7200 documents) and many features (up to 1000). Therefore, we decided to use Two-Part MDL.

2.2.2 Some technical details

Throughout this thesis we need some technical details concerning the MDL principle, which we discuss here.

Different types of codes An important issue when applying Two-Part MDL is that the codes used to encode the data should be self delimiting, i.e. a decoder should be able to recognize the end of a code word without looking past the end of the code word. We follow the definitions of [5]:

Definition 2.1 *A* **code** *C for a random variable X is a mapping from $\mathcal{X}$, the range of X, to $\mathcal{D}^*$, the set of finite length strings of symbols from a D-ary alphabet.*

Throughout this thesis we will use a binary alphabet i.e. $D = 2$. Self delimiting codes are known as 'prefix codes':

Definition 2.2 *A code is called a* **prefix code** *or* **instantaneous code** *if no codeword is a prefix of any other codeword.*

Kraft Inequality An important result in information theory, linking probability distributions and prefix codes, is the Extended Kraft Inequality:

Theorem 2.1 (Extended Kraft Inequality) *For any countably infinite set of codewords that form a prefix code, the codeword lengths, l_i, satisfy:*

$$\sum_{i=1}^{\infty} D^{-l_i} \leq 1$$

Conversely, given any $l_1, l_2, \ldots$ satisfying the former inequality, we can construct a prefix code with these codeword lengths.

For the proof we refer to ([5], page 84). Often, it is useful to consider 'idealized' code word lengths $l_1, l_2, \ldots$ which satisfy the Extended Kraft Inequality, but are *not* integer valued. We call them *idealized* because obviously a code word can only consist of an integer number of symbols. Given a probability distribution $p(\cdot)$ over set of objects $\mathcal{X}$, we call a code $\hat{C}$ optimal if:

$$\hat{C} = \arg\min_{C} \sum_{x \in \mathcal{X}} p(x) l_C(x),$$

where C is a code over $\mathcal{X}$ and $l_C(x)$ is the length of the code word that C assigns to $x \in \mathcal{X}$. Hence, the optimal code needs the minimal *expected* (with respect to p) code length. A question one could ask is: "Given a probability distribution p over $\mathcal{X}$, what are the code word lengths that an optimal code assigns to the elements of $\mathcal{X}$?". We call those lengths the *optimal code word lengths.* It turns out ([5], Section 5.3) that the optimal codeword length for $x \in \mathcal{X}$ is given by[3]: $l_x = -\log p(x)$. We call a code assigning such code word lengths a *Shannon-Fano code.* Note that these codeword lengths satisfy the Extended Kraft Inequality. This means that there is a prefix code with these optimal, in the sense of on average shortest, codeword lengths. Also, note that in practice code words consist of an integral number of symbols. So, code lengths are to be taken $\lceil -\log p(i) \rceil$, so they differ less than 1 bit from the idealized code length. It is clear that these code words still satisfy the Kraft Inequality. See ([5], Chapter 5) for a discussion on how the ideal code length can be approximated arbitrarily close by encoding blocks of objects rather than individual objects.

Encoding integers Often, we have to encode integers when we are encoding models. If the code does not have to be self delimiting we can code the integers just by assigning to n the n^{th} string of a lexicographical ordering of all binary strings of finite length. This takes about $\log n$ bits. To make this code self delimiting, we can just insert a '1' after each symbol and insert a '0' after the last symbol. Hence, this takes about $2 \log n$ bits. However, we can make more efficient self-delimiting codes for the integers by applying the following 'trick' one or more times. A self delimiting code for the set of integers can be obtained as follows:

1. Encode the *length* of the code word in a self delimiting way.

2. Encode the integer in a standard (not self-delimiting) way.

Now, the end of the code word can be determined from its already encoded length. Hence, the concatenation of the codewords forms a self-delimiting code for the integers. With $L^*(n)$ we denote the codeword length assigned to integer n by this coding scheme, note that $L^*(n) \leq 2 \log \log n + \log n$.

[3]Throughout this thesis we will use logarithms with base 2.

Encoding models In an application of Two-Part MDL we need to have a code for encoding the models under consideration. Suppose that we can represent our models with a parameter vector $\theta^{(k)}$ where k is the number of parameters in the model. We can code the model by first encoding the number k and d with a prefix code, where d stands for how many bits we will use to encode each parameter. Next, we encode the parameters. However, how do we determine how many bits we use to encode the parameters? It turns out that this question can be partially answered: if the models satisfy certain regularity conditions, then asymptotically in m, the number of examples available, the optimal precision to encode the parameters is: $\frac{1}{2}\log m + c$ where c is some computable constant that may depend on k ([29], page 57).

However, in our case this way to encode the models rendered the code needed to encode the models too long as compared to the code needed to encode the class labels. That is: the code length of the models dominated the code length of the class labels, hence models with just one word group were selected. We suspect that the regularity conditions needed for Rissanen's result do not hold in our case. Therefore, we used a tailor made coding scheme to encode the models. We discuss it in Chapter 4.

Chapter 3

Text classification

This chapter provides an introduction to text classification and describes in detail the Naive Bayes classification model.

3.1 Introduction to text classification

In recent history the quantity of 'online' textual information has been growing explosively. In order to use these huge amounts of available information there is a growing need for efficient methods to help people accessing the information. *Text classification*, also known as *text categorization*, aims to provide solutions here.

Given a set of classes, the goal of text classification is to decide to which (unique) class a text belongs. Much of the research on text classification is done on the 'automatic generation of text classifiers'. The idea is that a program is presented with a set $S = \{(d_1, c_1), \ldots, (d_m, c_m)\}$, where d_i is a document and c_i is the class that belongs to the document (typically assigned by an 'expert' who knows to what class a document belongs). The program generates, on the basis of S, a classifier that can be used to assign a class to new (unseen) documents. As can be expected, text classification has also been a quickly evolving field of research in the last years.[1] Many methods have been applied to the problem of text classification: k-nearest neighbor, support vector machines, decision trees, naive Bayes, neural networks [36] and symbolic rule learning [25], to name just a few. See [1] for a survey on text classification methods. The conclusion of [1] is that Naive Bayes, k-nearest neighbor, decision trees and support vector machines all perform reasonably well and neither approach appears to be superior over the others.

One of the problems encountered when designing text classification method is the large 'dimensionality' of the data. It is common practice to represent the texts in a space with one dimension for each word encountered in the collection of documents under consideration. Typically, there are in the order of tens of thousands of different words in such a collection. Both time- and space complexity of classification algorithms are dependent on the dimensionality of the data. Modern computer equipment is not able to implement efficient classification methods which operate on data in such high dimensional spaces. Therefore, in order to produce fast classification methods we have to reduce the dimensionality of the data. In the next chapter we will start discussing the problem of reducing the enormous dimensionality.

This chapter discusses in the next section the Naive Bayes model we used for text classification in this thesis. We used Naive Bayes in our research (also used

[1] In [30], a standard work on Artificial Intelligence, a special section is devoted to text classification. This indicates that text classification is becoming a field of research in itself rather than 'just a application domain of standard classification and machine learning techniques'.

in [3]), because of its ability to provide simple and robust classifiers in situations where relatively little data is provided.

3.2 A probabilistic model for text classification

In classification (and also in regression tasks, which form a generalization of classification tasks) we are interested in predicting a (set of) unknown variable(s) on the basis of a (set of) known variable(s). In the case of text classification we want to predict the class-label on the basis of a text.

A bag of words It is a common practice in text classification to represent texts as a *bag of words*. The text is treated as a list of word frequencies, therefore the information of the position of the words in the text is discarded. We adopt this practice here too. Quoting ([20], Section 3):

> "An ongoing surprise and disappointment is that structurally simple representations produced without linguistic or domain knowledge have been as effective as any others [19, 21]. We therefore make the common assumption that the preprocessing of the document produces a bag (multiset) of *index terms* which do not themselves have internal structure."

The task is to predict the class label for a new, not yet classified, text. We do this by computing for this text a distribution over the classes generated by some probabilistic model and then simply pick the class with the highest probability. So, if d is a text and $c \in C$ is a class from the set C of classes, we need to compute $p(c|d)$. We do so by computing $p(d|c)$ and using Bayes' Rule to link them together:

$$p(c|d) = \frac{p(c)p(d|c)}{p(d)} \tag{3.1}$$

We write $p(d|c) = p(|d|)p(d|c, |d|)$, with $|d|$ the length of document d and let $p(d) = \sum_{c \in C} p(c)p(d|c)$. Since $p(|d|)$ gets divided away in equation (3.1) there is no need here to define it explicitly. The dependence of $p(d|c)$ on $|d|$ will be throughout assumed and dropped in the rest of this thesis.

A bag of *independent* words Furthermore, we make an independence assumption between the words in a text. Let w_k be the k^{th} word in the text d, then by the independence assumption we write:

$$p(d|c) = \prod_{k=1}^{|d|} p(w_k|c) \tag{3.2}$$

This is a quite unrealistic assumption. However, we make the assumption to make the number of probabilities we have to estimate manageable. It turns out ([2], page 25) that none of the known text classification approaches shows this assumption to be degrading performance. The name of this type of model, *Naive* Bayes (NB), reflects the naiveness of this independence assumption.

Estimating the probabilities The $p(w|c)$ can be estimated directly from the set of training documvocabularyents. Treat all documents in the class as one big document, then take $p(w|c)$ as the fraction of the words occurring in all documents in the class being w. Say we have 10 documents of 100 words in class c, in each of which the word w occurs 5 times. Furthermore, we have 9 documents of 1000 words in class c in each of which w occurs 100 times. Then we estimate $p(w|c)$ as

$$p(w|c) = \frac{10 \cdot 5 + 9 \cdot 100}{10 \cdot 100 + 9 \cdot 1000} = 0.095.$$

We smooth this distribution to avoid probabilities with value zero or one. We use the *Laplace Smoothing* ([5], Section 7.10) method: just add '1' to each entry in the word frequency lists and renormalize. So then, assuming a V of size $|V|$, we arrive at:

$$p(w|c) = \frac{950 + 1}{10000 + |V|}.$$

The $p(c)$ are directly estimated from the training data, $p(c)$ is just the proportion of the training texts having label c.

Why being naive might be ok under zero-one loss Despite the fact that in many cases the NB assumptions are violated, the NB classifier appears to perform good also in those cases. In [9, 11] an effort is made to explain this phenomenon in a rigorous manner. Here we give an informal summary of the arguments provided there. The surprisingly good performance of NB is explained by first dividing the risk (average loss) into three additive parts:

1. *Intrinsic error* due to noise in the sample
2. *Bias*, systematic component of the error due to the model class used. It reflects how good the models are on average.
3. *Estimation variance*, due to the finiteness of the sample and the resulting imperfect estimates. This reflects how the models vary when other samples are used.

There is a natural trade-off between bias and variance. Highly expressive model classes are likely to have small bias while being highly sensitive on the actual sample (high variance). When using very simple model classes the bias is high due to the limited modeling capabilities and variance is low.

The functions that measure prediction performance are usually called *loss functions*. Common choices for loss functions are 'log likelihood' (take $-log$ of the probability the model assigned to the correct value), 'squared error' (taking the loss as the square of the difference between correct and predicted value) and 'zero-one error' (which equals zero for correct prediction and one for incorrect prediction). Using the zero-one loss function in classification, the loss becomes relatively insensitive to the 'bias' while it is still sensitive to the 'variance', as compared to squared error loss. This is due to the 'winner takes all' nature of zero-one loss, i.e. it suffices that the actual class get highest probability to ensure zero loss. In other words: the conditional (on the examples) class probabilities can be coarsly estimated without affecting the loss. Due to the limited number of parameters used by the NB model and the resulting smoothness of the models, NB has relatively low 'Estimation variance'. Therefore, NB may perform very well under zero-one loss, even if the independence assumptions are violated.

3.3 Clustering words

In the introduction we have talked about 'clustering' or 'merging' words already, but in a rather vague way. The clustering of words into groups is implemented by

Before:

	c_1	c_2	$\dots$	c_n
w_1	$o_{1,1}$	$o_{1,2}$	$\dots$	$o_{1,n}$
w_2	$o_{2,1}$	$o_{2,2}$	$\dots$	$o_{2,n}$
$\dots$	$\dots$	$\dots$	$\dots$	$\dots$
w_m	$\dots$	$\dots$	$\dots$	$\dots$

After:

	c_1	c_2	$\dots$	c_n
$w_{1,2}$	$o_{1,1} + o_{2,1}$	$o_{1,2} + o_{2,2}$	$\dots$	$o_{1,n} + o_{2,n}$
w_3	$o_{3,1}$	$o_{3,2}$	$\dots$	$o_{3,n}$
$\dots$	$\dots$	$\dots$	$\dots$	$\dots$
w_{m-1}	$\dots$	$\dots$	$\dots$	$\dots$

Figure 3.1: Merging words in the word-class frequency matrix

mixing the corresponding distributions over the classes. We follow the method of merging the conditional distribution of classes given a certain word, as proposed in [3, 32]. When we merge two distributions, we use the weighted mean of the original distributions. Suppose we merge the words w_s and w_t into w_{st}, the resulting distribution is computed as follows:

$$p(c|w_{st}) = \frac{p(c, w_t) + p(c, w_s)}{p(w_t) + p(w_s)} = \tag{3.3}$$

$$\frac{p(w_t)}{p(w_t) + p(w_s)} p(c|w_t) + \frac{p(w_s)}{p(w_t) + p(w_s)} p(c|w_s).$$

Furthermore, we set $p(w_{st}) = p(w_s) + p(w_t)$. Since we *identify* w_s and w_t as being really the same word, it is clear that its probability of occurrence is the sum of the probabilities of occurrence for the old words. This fully determines the new distributions. Figure 3.1 shows why this is a natural way to merge the word distributions. The tables in the figure are called *word-class frequency table.* The i^{th} row represents the i^{th} word group or *cluster* in the lexicon V. The j^{th} column represents the j^{th} class. The value of position (i, j) in the table is the number of times we saw the word i in the class j. In the rest of this thesis we identify such tables with the probabilistic model that corresponds to it. The lower table shows how the new word-class table is derived from the original (upper) one: just sum the rows of the two words in the word-class frequency list, as if the two words were really the same. After summing the rows we can remove the old rows since the information in the old rows is not needed anymore. This is due to the fact that if we want to group a number of words together, we can do so by a sequence of pairwise groupings. This is easily seen from Figure 3.1.

What is the model? Note that if someone wants to use a Naive Bayes model that does not use all the $|V|$ original words, but say $n < |V|$ word groups he should know which words correspond to a certain word group. Therefore, we argue that a model is not just the probability distribution described in the previous section. A function mapping the original words to their word groups is needed in interpreting the model. This function 'implements' the clustering.

In the next chapter we are concerned with encoding the models and the data, in order to obtain code lengths needed to implement the MDL principle.

Chapter 4

Applying two-part MDL

To apply the MDL principle to a particular problem, we have to determine the model that will optimize the sum in equation (2.2). Since in our case (and in many other cases) there is no method available to determine the optimizing model directly from the training data, we have to find it using some local search algorithm. In this chapter we discuss in detail the coding schemes used to apply Two-Part MDL. First, we discuss how we can code the models. Second, the computation of the code length of the data using the model is discussed.

4.1 Encoding the models

Recall from Section 3.3 that the model consists of a certain clustering of words into word groups, together with a distribution over the classes for each word group. Therefore, we split the code in two:

1. we code which clustering is used.

2. we code the corresponding class-distributions.

Encoding clusterings To encode which clustering is used, we code the index in some agreed enumeration of all possible clusterings. To compute the number of bits needed to code the index of the used clustering we need to know how many possible clusterings there are. Recall that the words are clustered in non-overlapping groups. This leads to the following combinatorial question:

> "We have n marbles (words) and k boxes (clusters). In how many possible ways can we distribute the marbles over the boxes, such that no box is empty and we do not distinguish between different orderings of the boxes?"

This is exactly the definition of Stirling numbers of the second kind ([12], Section 6.1) denoted $\left\{ {n \atop k} \right\} (n \geq k)$. Note that $\left\{ {n \atop 1} \right\} = 1 = \left\{ {n \atop n} \right\}$. The rest of these numbers can be computed using the recurrence relation:

$$\left\{ {n \atop k} \right\} = \left\{ {n-1 \atop k-1} \right\} + k \left\{ {n-1 \atop k} \right\} \tag{4.1}$$

The index can now be encoded by coding the number k in a prefix free fashion, because then we can compute how many bits will follow for the index, as n is given. To encode k in a prefix free fashion we use a standard coding scheme to encode integers. We denote the the number of bits needed to do so by $L^*(k)$, see Chapter 2. We then use $\log \left\{ {n \atop k} \right\}$ bits to encode the index itself. In appendix B we discuss another more intuitive way to encode which words are clustered together. It turns out that this alternative method approximates the code lengths obtained with the Stirling encoding reasonably well.

Encoding the distributions Next, we discuss how to code the distributions over the classes for all word groups. Remember that the probabilistic model can be represented by a table as depicted in Figure 3.1 on page 16. We called such tables *word-class frequency tables.* Therefore, we may encode the frequency table to encode the probabilistic model. Now, we need to code in a prefix free way a sequence, of known length, of integers. This can again be done using the same standard way to encode integers as above. However, this coding scheme is sub-optimal here.

It turns out that a few words occur very often and most other words quite irregularly. This is a known property of natural language called Zipf's Law [38], which we observed to hold for the documents used in our experiments. Furthermore, some simple rewriting shows that to use the model for prediction, we do not need to know how often each word occurred in the sample.[1] It suffices to have a distribution over the classes given each word and a marginal distribution over the classes. This implies that we may use the distributions $p(c|w)$ for prediction and do not need the joint distribution $p(c, w)$. Also, note that we do not *have* to use the exact distribution estimated from the sample, we may also use a (close) approximation to it, see paragraph *'Why being naive might be ok under zero-one loss'* on page 15. If we scale all rows in the frequency table to sum up to the same number, then we *know*, when decoding, what the sum of the row will be. Finally, note that the marginal distribution $p(c)$ over the classes is not changed when we merge word groups. Hence, the code length needed to encode the marginal distribution remains constant if we merge two word groups. Therefore, we neglect the encoding of the marginal distribution $p(c)$ here.

We use all these observations to find an efficient coding scheme for the models. Below we describe our coding scheme in four steps:

Scaling Scale all rows in the original word-class table (see Section 3.3) to sum up to some fixed integer $\rho \geq 1$. So, ρ determines the precision with which we will approximate the original distributions over the classes for each word group. We will still use only integers to represent the distribution.

Pruning We can use the fact that all rows in the scaled table sum up to ρ. We know that all other entries on a certain row are zero if ρ is the sum of all entries up that point and we know ρ. In the same manner, we always know the value of the last entry on a row if we have seen the other entries. We do not have to encode the values of these entries that can be derived from previous entries. We can 'prune' the table to exclude those entries, say that the pruned table has n entries.

Histogram We make a histogram of how often each value in $\mathcal{P} = \{0, 1, \ldots, \rho\}$ occurs in the pruned table. This histogram can be seen as an empirical distribution p on $\mathcal{P}$. We encode this distribution p over $\mathcal{P}$ using the L^* encoding for the values in the histogram.

Simple encoding Now, we approximate the code length needed to encode the n symbols in the pruned table by $n\mathcal{H}(p)$. We encode n objects, with an expected code length of $\mathcal{H}(p)$ bits. $\mathcal{H}(\cdot)$ denotes the entropy function, see Appendix A for a definition. If we encode a block of n objects, it is provable that there is

[1]The quantity $L(c)$ as defined on page 19 can be expressed in terms of $p(c|w)$ and $p(c)$ up to an additive constant. The constant can be disregarded in the prediction proces.

a code which yields an expected code length of at most $\mathcal{H}(p) + 1/n$ bits. See [5] Section 5.4 for a detailed exposition on this approximation.

It turns out that this coding scheme produces code words much shorter than those from the standard coding scheme or a coding scheme that does not use the distribution over $\mathcal{P}$.

Empirically we found that a value of $\rho = 25$ is (close to) optimal for the experiments we conducted here. In our experimentation, the prediction performance (of about 90% correct) was reduced by less than one percent if we used $\rho = 25$. For values smaller than 25 performance started to drop significantly, however using $\rho = 3$ still enables us to predict up to 70% correctly for a training set of 400 documents per class! (See paragraph 'Why being naive might be ok under zero-one loss' on page 15.)

4.2 Encoding the labels

Recall that the code for the data consists of two parts. We just described how we can code the models, now we will discuss how to encode the data *using* a model. We discuss two methods to encode the class labels.

Using code lengths prescribed by the probabilistic model To encode the text labels based on the texts and a model, we can use a standard encoding that uses directly the probability that the model assigns to a class based on a text. We use the Kraft Inequality (see Section 2.2.2) to obtain code lengths corresponding to a certain probabilistic model. Remember that the code lengths are given by $-\log p(c|d)$ to encode class c based on text d and the probabilistic model p.

To compute the code length we use the definitions from page 14. However, for a normal text the probabilities $p(d|c)$ can get so small that they cannot be represented in standard programming tools like MATLAB. However, we can easily compute the *logarithm* of $p(d|c)p(c)$ (by using that 'the log of a product' is 'the sum of the logs'). Therefore, we may hope to compute $p(c|d)$ using the following method:

1. Compute for all $c' \in C$: $L(c') = \log \big(p(d|c')p(c')\big)$
2. If the $L(c')$ differ not too much, we can also compute
$$E(c') = 2^{L(c')-L(c)} = \frac{p(d|c')p(c')}{p(d|c)p(c)} = \frac{p(c'|d)}{p(c|d)}$$
3. Compute
$$S = \sum_{c' \in C} E(c') = \frac{\sum_{c' \in C} p(c'|d)}{p(c|d)} = 1/p(c|d)$$
4. It follows that $-\log p(c|d) = \log S$.

This way, we can compute the code length exactly without needing extremely great precision. We use the fact that while the $p(d|c)$ for a document d are very small, the ratios $E(c')$ are likely to be less small.[2]

[2] In our experiments the $E(c')$ were representable in MATLAB. However, one could also set the $E(c')$ to a very small value in cases where the difference $L(c') - L(c)$ would still be too large.

Alternatively: encoding exceptions The probabilities the model assigns to the classes may differ greatly. This is due to the structure of our model; the probability on a class given a document is the product of as many terms as there are words in the document. Therefore, if the model makes a wrong prediction, the minus log of the probability of the correct class can be a rather large number. As a result, the curve of the code length needed to encode the class labels can behave rather wildly i.e. not smooth at all. The greedy nature of the search algorithm might be a second cause for the large fluctuations in code length over the classes. However, the strong fluctuations in code length may result in bad performance of the algorithm because the number of word groups attaining the minimum description length may vary greatly from sample to sample due to the fluctuations. Below, we discuss two alternative coding schemes that result in a smoother curve for the code length.

We can transform our model to reduce these large differences in probabilities. Note that we can measure prediction accuracy on the training data, say that a fraction ϵ of the examples is misclassified. Let $|C|$ be the number of classes. Now, an intuitive method suggests itself to transform the model: Put probability $1-\epsilon$ on the class proposed by the model (i.e. the class to which the highest probability was assigned by the model). Put probability $\epsilon/(|C|-1)$ on the other $|C|-1$ classes. Now, we can use this new distribution to encode the class labels as before. Note that in order to use this code, the decoder needs to know the number of misclassifications $m\epsilon$ (since this number determines the distribution and hence the code we use to encode the class labels). Therefore, we need to encode the number $m\epsilon$, we use the coding scheme discussed in paragraph 'Encoding integers' in Section 2.2.2 to do so.

Another idea to encode the class labels relative to a model would be the following:

1. Encode *how many* examples were misclassified by the model. We use the L^* coding scheme to do so.

2. Encode *which* examples were misclassified by the model. Let m be the number of examples in the sample and let $\mathcal{M}$ be the set of misclassified examples. There are $\binom{m}{m\epsilon}$ possible subsets of misclassified examples. We encode the index of the actual subset $\mathcal{M}$ in an agreed enumeration of all such subsets using $\log\binom{m}{m\epsilon} \approx m\mathcal{H}(\epsilon)$ bits.[3]

3. Encode for the examples incorrectly predicted the correct class. We need $\log(|C|-1)$ to correct each of the $m\epsilon$ mistakes.

This costs $L^*(m\epsilon) + m\mathcal{H}(\epsilon) + m\epsilon\log(|C|-1)$. It is easy to show (we do so below) that this code length is equal to the one resulting from the method of transforming the model described above. First of all, we encode the number $m\epsilon$, this takes $L^*(m\epsilon)$ bits. Note that we use the Shannon-Fano code in the first method described. So, we encode correctly predicted classes using $-\log(1-\epsilon)$ bits, there are $m(1-\epsilon)$ correctly classified examples, so this takes $-(1-\epsilon)m\log(1-\epsilon)$ bits. Next, the $m\epsilon$ incorrectly predicted labels get a code length of $-\log\frac{\epsilon}{|C|-1}$ bits, hence here we need $-m\epsilon\log\frac{\epsilon}{|C|-1}$ bits. So in total this becomes:

$$
\begin{aligned}
L^*(m\epsilon) - (1-\epsilon)m\log(1-\epsilon) - m\epsilon\log\frac{\epsilon}{|C|-1} &= \\
L^*(m\epsilon) + m\big(-(1-\epsilon)\log(1-\epsilon) - \epsilon\log\epsilon\big) + m\epsilon\log(|C|-1) &= \\
L^*(m\epsilon) + m\mathcal{H}(\epsilon) + m\epsilon\log(|C|-1)
\end{aligned}
$$

More on the topic of interpreting apparently 'non-probabilistic' models as probabilistic is written in Section 2.2 and Chapter 5 of [14] and [27, 29]. In fact, the method mentioned above is based on the method mentioned in Example 2.6, page

[3] For the approximation see ([34], page 24) and ([5], Example 12.1.3).

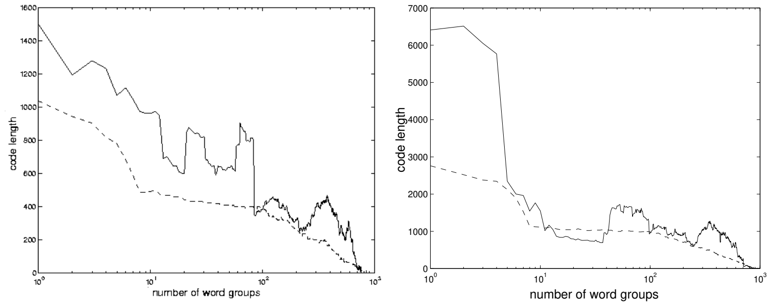

Figure 4.1: A plot comparing the code lengths yielded by the two methods to code the labels. The solid line represents directly using probabilistic model. The dashed line represents coding exceptions. Note that the horizontal axis, containing the number of word groups, is logarithmic! We used 50 (left) and 150 (right) texts per class.

33 of [14]. Note that this coding scheme implements a zero-one loss function, in the sense that the code length depends only (as a concave function) on the *fraction of misclassification* ϵ. Recall that in the third paragraph of Section 3.2 on page 14 we argued that using the Naive Bayes model is more likely to be ok under zero-one loss (as compared to log-likelihood).

A plot comparing the code length yielded by the original probabilistic model and by the method described above is provided in Figure 4.1. This plot represents the typical situation: the curve for the exceptions code is smoother than, and on most places below, the curve using the probabilistic model directly. Specifically for small samples coding exceptions yields shorter code lengths. This might be due to the rough estimates for the parameters we obtain for small samples: Using the probabilities directly to encode the classes could render a class label as 'almost impossible' and hence give it a *very* long code. code lengths. Coding exceptions assigns code words of reasonable size as long as the ϵ is of reasonable size. For large samples, coding exceptions may yield longer code lengths. In this case the estimates are quite good, therefore is likely in case of misclassification that the correct class gets a high probability that is just slightly lower than the class receiving the highest probability. Hence, the probability that the probabilistic model assigns to the correct class might be (considerably) higher than $\epsilon/(|C|-1)$. However, the code length when coding exceptions is always reasonably smooth as a function of the number of word groups. Furthermore, it is shorter than the code using the probabilistic model directly when using just a few word groups (ten or fewer).

4.3 A concession to reduce computational cost

Because we use MDL as model selection method, ideally we would use the change in code length (due to merging two word groups) as a criterion to decide which word groups to merge. We call this quantity the *merge cost*. At each step of the algorithm we need to know the merge cost of each pair of word groups. Therefore, we need to be able to compute the merge cost efficiently. If we do not use the exact change in code length for the coding schemes discussed above, but settle for the indicative measure discussed below instead, computational costs can be reduced considerably. The reason is that using this measure the merge cost of two groups does not change if we merge two other groups. This will be shown below.

Using conditional entropy as expected code length Instead of computing the change in exact code length, we could use the change in *expected code length.* The expected code length can be written in terms of 'conditional entropy'. The *conditional entropy* $\mathcal{H}(X|Y)$ is the expected amount of bits needed to encode the value of a variable X given the value of a variable Y.[4] In Appendix A definitions of entropy and conditional entropy are given.

Let C be a variable taking as values the different classes. We assume that all texts in the sample are of equal length, say n. Let $\{W_i|1 \leq i \leq n\}$ be variables taking as values words from the vocabulary V. The expected amount of bits needed to encode a text label based on the text and the probabilistic model is given by:

$$\mathcal{H}(C|W_1, \ldots, W_n) = \mathcal{H}(C) - I(C;\ W_1, \ldots, W_n), \tag{4.2}$$

where $I(X;\ Y)$ is the *mutual information*, see Appendix A. The following proposition shows that $I(C;\ W_1)$ provides upper and lower bounds on $I(C;\ W_1, \ldots, W_n)$, the proof is given in appendix C. Note that we need the i.i.d. property that is also made by the naive Bayes model[5]:

Proposition 4.1 *Let $C, W_1, \ldots, W_n (n \geq 1)$ be discrete random variables such that $W_1, \ldots, W_n$ are identically distributed given and independent given C, then*

$$I(C;\ W_1) \leq I(C;\ W1, \ldots, W_n) \leq nI(C;\ W_1)$$

Note that $\mathcal{H}(C)$ is constant if we merge word groups, see Section 3.3 on how we merge word groups. It follows that we can find an upperbound for the change in $\mathcal{H}(C|W_1, \ldots, W_n)$ due to a combination of word groups in terms of the decrease in $I(C;\ W_1)$. In the next paragraph we discuss how this quantity decreases due to a merge of word groups. This, forms one motivation[6] to use $I(C;\ W_1)$ as a measure to decide which word groups to merge. Another, more intuitive, motivation is that the mutual information expresses how mutually predictable the variables C and W_1 are (how much information the convey on each other): we want to merge those word groups such that the resulting word groups convey as much information on the class variable as possible.

The change in mutual information Suppose we merge word groups w_s and w_t into w. Let p_s, p_t and p_w be the corresponding distributions over the classes, see Section 3.3. Furthermore, let π_s, π_t and π_w denote the marginal probabilities on these word groups. As noted in [32] $I(C;\ W_1)$ changes by the Jensen-Shannon divergence, $JS_{\pi_s,\pi_t}[p_s, p_t]$, between p_s and p_t multiplied by the probability of either word: $p(w_s) + p(w_t)$. See Appendix A for a definition of the Jensen-Shannon divergence. The cost of merging two words is given by:

$$\sum_{c \in C} \sum_{i=s,t} p(w_i) p(c|w_i) \log \frac{p(c|w_i)}{p(c|w)} \tag{4.3}$$

[4]It is interesting to note that in [23] it is shown that the conditional entropy provides upper and lower bounds on the 'Bayes classification error'. The *Bayes classification error* for a classification problem is defined as: $P_{\mathrm{BE}} = \sum_x p(x)(1 - \max_i\{p(y_i|x)\})$, where x stands for observations and $y_i (1 \leq i \leq M)$ for class i. We predict the class y_i for x that gets highest probability $p(y_i|x)$ over i. Now, P_{BE} measures, if the distributions $p(x)$ and $p(y_i|x)$ are the 'true' distributions according to which the data are drawn, how likely it is to make an incorrect prediction. Or, equivalently, the expected zero-one loss if we predict in the way described above and the probabilistic model is the 'true' distribution, see [26] for a full treatment of the Bayes Error in decision theory. The bounds on P_{BE} are:

$$\frac{(\mathcal{H}(Y|X))^2}{4(M-1)} \leq P_{\mathrm{BE}} \leq \frac{\mathcal{H}(Y|X)}{2}$$

[5]See Section 3.2.

[6]Note that the upper bound can be rather loose for large n, since it depends linearly on n.

It is easy to see from (4.3) that the merge cost of two words remains equal if we merge two other words. Note that the distributions $p(c|w_i), i = s, t$ are not influenced when we merge two other words. Also, the distribution $p(\cdot)$ over the words is not changed by the merge except for the probability on the new word group. Therefore, if we merge two words, we only have to compute merge costs corresponding to merging the new word group with already existing word groups.

Chapter 5

The algorithm

In this chapter we discuss the algorithm we propose to find good word groups for text classification.

5.1 Implementation

Here, we describe the algorithm on a 'meta-level', all details have been discussed in the previous chapters already. The data set and preprocessing needed to use the algorithm are discussed in the first section of the next chapter. For our experiments we implemented the ideas discussed above in MATLAB.

Our program takes as input a set of documents that are represented as word-text frequency lists. As output it gives a clustering of the words together with a naive Bayes model that uses the word groups as features. First, the algorithm computes empirical distributions

1. $p(c)$ over the classes.
2. $p(w|c)$ over the words given the classes.

How to compute these distributions is explained in paragraph 'Estimating the probabilities' in Section 3.2 on page 14. Next, for all word pairs under consideration the merge cost, i.e. the JS divergence between the class distributions of two words, is computed. This is just a matter of filling in the definition and straightforward computation.

Now, the actual clustering of words begins. In general there are many ways[1], to clusters the words. Therefore, we need a heuristic to search the *cluster-space*, the space filled with all possible ways to cluster the words. To do so, we do the following:

1. Start out with each word in its own cluster.
2. Merge two clusters that are the 'closest' to each other in terms of JS divergence, see Section 4.3.
3. Repeat this until just one cluster is left.

In this way we only have to examine as many clusterings as we have words in the corpus. In general there are many methods to find clusters on the basis of proximity information. However, because we have to investigate clusterings using one up to some (large) number of clusters, the *agglomerative clustering* method described above is appropriate.

[1] See Section 4.1. For example, $\left\{ {9 \atop 3} \right\} = 118124$.

INPUT: A set of pre-classified texts
OUTPUT: A clustering of words
<u>**begin**</u>
  - compute empirical distribution over classes
  - compute empirical distributions over classes given words
  <u>**for**</u> each pair of words <u>**do**</u>
    - compute JS divergence of pair
  <u>**end**</u>
  <u>**while**</u> there is more than 1 word group <u>**do**</u>
    - determine pair with minimum JS divergence
    - compute change in code length and store it
    - record word clustering if shortest code length up to now is reached
    - adapt probability distributions to implement the word merge
  <u>**end**</u>
  - output word clustering yielding minimum code length
<u>**end**</u>

Figure 5.1: The algorithm in pseudo code

During the clustering, after each merge the change in code length (as defined by the coding schemes discussed in the previous chapter) is calculated and stored. When just one cluster is left, the number of clusters yielding minimum code length is determined. A pseudo code version of the algorithm is provided in Figure 5.1.

5.2 Complexity of the algorithm

In this section we discuss the time and space complexity of the algorithm. Let $|V|$ denote the number of words in the lexicon V, i.e. the number of words occurring in the sample. Let $|C|$ denote the number of classes and let $|D|$ denote the number of documents in the sample D. The $f \in \Theta(g)$ notation is used to denote that the function f can be upper and lower bounded by a multiple of g from some point on.

Space complexity To store the word-class data we need $|V||C|$ integer positions. To compare all the possible merges of clusters at each step, we need to compute for each pair of clusters the merge cost. There are originally $|V|(|V|-1)/2 \in \Theta(|V|^2)$ word pairs.

For both coding schemes to code the class labels we need the training documents in order to determine the code length. Therefore, we need to store all the word frequency lists corresponding to the training texts, this costs $|V||D|$ integer positions. Note that $|D| \gg |C|$, therefore, the space complexity is $\Theta(|V||D| + |V|^2)$.

Time complexity At the start of the algorithm, we compute merge costs for all pairs. If we merge two clusters, we have to update the merge costs only for at most $|V|$ pairs. Since there are originally $|V|(|V|-1)/2 \in \Theta(|V|^2)$ pairs of words and we perform $|V|-1$ merges, we conclude that we have to compute $\Theta(|V|^2)$ merge costs. Looking at equation (4.3) on page 22 we see that each merge cost can be computed in $\Theta(|C|)$ operations. Yielding a total $\Theta(|C||V|^2)$ operations to compute the JS divergences.

The change in code length needed to encode all the classlabels can be computed in $\Theta(|C||D||V|)$ operations. This is easily seen if we note that equation (3.2) can also be written as a product over the words in the vocabulary. Let $o_{i,j}$ denote how often word i occurs in document j, then we can compute $\log p(d_j|c)$ in $\Theta(|V|)$ operations using:

$$\log p(d_j|c) = \sum_{w \in V} o_{w,j} \log p(w|c).$$

To obtain the code length for class labels, we need to compute $p(d_j|c)$ for all $d_j \in D$ and $c \in C$.[2] Note that we compute the change in code length after each of the $|V| - 1$ merges. Hence the algorithm has time complexity $\Theta(|C||D||V|^2)$.

Reducing complexity The mentioned time complexity can be a problem in applications where fast performance is a must. However, we can reduce the time complexity of the first phase of the algorithm from $\Theta(|C||V|^2)$ to $\Theta(|C|)$ by using a maximal number of wordgroups we consider for merging[3]. Each time when a merge is performed a new word can enter the 'pool' of merge candidates. Of course, if we bound the time complexity in this way, space complexity becomes $\Theta(|V||D|)$. In our experiments we used a 'pool size' of 100 word groups; this influenced performance hardly.

[2]This holds for both coding schemes, although the code length for the 'coding exceptions' coding scheme can be computed considerably faster. This is because we can just take the maximum of the $p(d|c_i)$ to determine whether the class was predicted correctly. For the other code scheme we also have to determine the value $p(c|d)$ exactly using the method described on page 19.

[3]a trick already mentioned in [3]

Chapter 6

Results and conclusions

In the first section of this chapter we discuss how our approach relates to other work reported in the literature. Second, we present our experimental results and provide some plots to gain insight in performance of the algorithm. In the last section of this chapter we present our conclusions.

6.1 Relation to other work

In the literature several publications [7, 8, 28] report on feature *selection* using the MDL principle to determine the number of features. Note that these papers concern feature *selection*, so they discuss methods to select a *subset* of useful features. This contrasts with the method proposed here: we use *combinations* of features instead. We found no publication considering the *clustering* or *combining* of original features into new ones using MDL in a 'supervised learning' setting.

Also, in text classification and *information retrieval*[1] literature, we find most feature extraction methods to be feature *selection* methods. In [37] a survey considering several feature *selection* methods is provided. Totally disregarding redundant words seems to be suboptimal. It would be better, if two words are highly related, to use some combination of them to make estimates more robust, instead of throwing away all information contained in one of the words. Latent semantic indexing (LSI) [10] is a method which does find linear combinations of words as new features. The method is based on the singular value decomposition of the word-text frequency matrix[2] and belongs in the 'unsupervised' class of feature extraction methods. The method proposed here is just like LSI finding combinations of words. We extend the method of finding word groups presented in [3] (and justified in [32]) to determine also the 'optimal' number of word groups. So, two advantages over LSI can be noted: First, the method is 'supervised'. Second, the method determines the number of new features itself.

Finally, it is interesting to mention the work published in [13] (and an accompanying book chapter by the same author [35]). There, words are clustered in non-overlapping word groups. These word groups are used together with a 2-gram Markov model to form a simple grammar over a (large) set of sentences given to the algorithm. Note that this setting differs from ours in that it is *unsupervised* and a more complex model class is used.

[1] Information Retrieval is a field concerned with developing methods to find for example documents that match a user query or are 'like' another document etc.

[2] In the *word-text matrix* each entry (i, j) denotes how often word i occurred in text j.

6.2 Experimental results

In this section we present some of our experimental results. We used the 20 NEWSGROUP DATASET [16] for text classification. To extract the necessary statistics from the data set we made gratefully use of the RAINBOW program, part of the BOW TOOLKIT by McCallum [24].[3] To measure predictive performance we used a test set of fifty documents per class. It turned out that the variance in predictive performance on different smaller test sets was negligible, therefore we decided to use just one test set.

pre-processing Using the RAINBOW program we were able to do stemming (convert all verbs to their stem, to reduce the number of words in the lexicon), use a stop list (ignore very frequent words like 'a' and 'the' etc.) and remove words that are very rare (they are likely to be spelling mistakes and/or not to influence global performance). To make the experiments manageable on the available equipment we decided to prune the lexicon to the 1000 words having the highest mutual information with the text labels.[4] Our algorithm was implemented in MATLAB. Using some simple UNIX 'shell-scripts' we transform the output of the RAINBOW program into a form (word-document matrix) readable for MATLAB.

The dataset The '20 NEWSGROUP DATASET' contains texts from newsgroups that are very much like each other, for example `comp.os.ms-windows.misc` and `comp.windows.x`. We wanted to test our algorithm in a setting where high prediction accuracy is possible. This, because we felt that finding a simple classifier that retains good classification accuracy is more interesting when the best prediction accuracy attainable is high. Therefore, we used a subset of the data set containing the newsgroups: `misc.forsale`, `rec.autos`, `rec.motorcycles`, `rec.sport.baseball`, `rec.sport.hockey`, `talk.politics.guns`, `talk.politics.mideast`, `talk.religion.misc`. The test set to measure learning performance (remember that no test set is used by the algorithm!) consisted of 50 documents from each class. The task was to predict the newsgroup from which an article originated.

Discussion of results In the figures on the next pages we illustrate the performance of the algorithm. The figures provide insight to the questions posed in Section 1.2. We will discuss them here one by one, starting with Figure 6.1.

In Figure 6.1 prediction accuracy on the test set is plotted against the number of word groups used in the model. For each number of word groups performance on the test set was tested and plotted. The shape of this curve is determined by the criterion we use to merge the word groups. The plot shows the validity of the method we used, i.e. the method proposed in [32]. For this experiment we used a sample of 450 documents per class. Note the logarithmic scale on the horizontal axis! We see that prediction performance is hardly affected until a *very* small number of word groups is used (performance starts to drop significantly only when we use order of ten wordgroups, where there were originally 1000 words!). This is the typical situation, we call the point in the curve where the performance starts to drop drastically the *critical point*. This phenomenon of initial great increase in quality and subsequent marginal increase in quality is also known in pattern recognition literature as the *elbow phenomenon*.[5]

[3] This software package (running under LINUX) implementing several useful techniques for text classification, is available for free over the Internet as well as the source code.

[4] This, together with the simple naive Bayes model, might be the reason that no overfitting is observed when using many word groups, see Figure 6.1.

[5] It is interesting to note that in [18] approximately the same number of features (10-15) is found to be optimal. Experiments are reported on both *selecting words* and *clustering* words. For

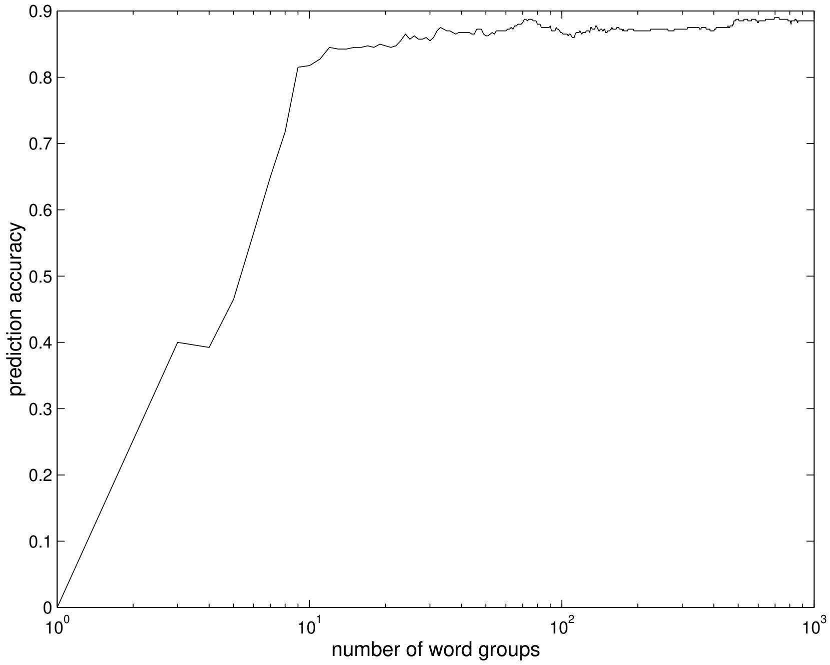

Figure 6.1: Prediction accuracy against number of word groups for sample size of 450 documents per class.

Figure 6.2 shows the number of word groups proposed by the algorithm as a function of the sample size, i.e. the number of documents per class in the sample. For this experiment as well as the one depicted in Figure 6.3 on page 31, sample sizes of ten up to 900 (with an interval of ten) documents per class were used. Note the logarithmic scale on the vertical axis!

Figure 6.2 contains four curves: three solid curves and one dotted. The highest (smooth) curve depicts the number of different words present in the sample. The heavily oscillating curve below the latter curve shows us which number of word groups yielded best performance on the test set. Hence, this is the number of word groups used by the model picked by cross validation. The heavy oscillation can be explained by looking at figure 6.1. One sees that little prediction performance is lost, and some is gained, during the first ±900 merges of word groups. Therefore, where exactly the maximal prediction accuracy is obtained differs heavily.

The remaining two curves show how many word groups were proposed by the algorithm using the two different coding schemes discussed in Section 4.2. The dotted line corresponds to using the probabilistic model directly to code the labels, where the solid line corresponds to coding exceptions. Note that using both coding schemes the algorithm suffers from a *start-up period*, in which the algorithm proposes using just one word group (containing all words). Such a start-up period was also encountered in the experiments reported in [34]. It is caused by the domination of the error term in equation (2.2) by the complexity term for small samples, this is due to the fixed precision with which the models are encoded. To interpret this

the clustering the 'cosine correlation' ([2], page 27) corresponding to the classical *Vector Model* [31] for text representation between words was used as a similarity measure. The Reuters Dataset [17] was used for most experimentation. There, a more general (but still supervised) framework is used: a varying number of 'class labels' can be assigned to a document. In experiments where the number of 'class labels' assigned is forced to be equal to the *correct number* of class documents, the 'elbow phenomenon' just mentioned is also observed.

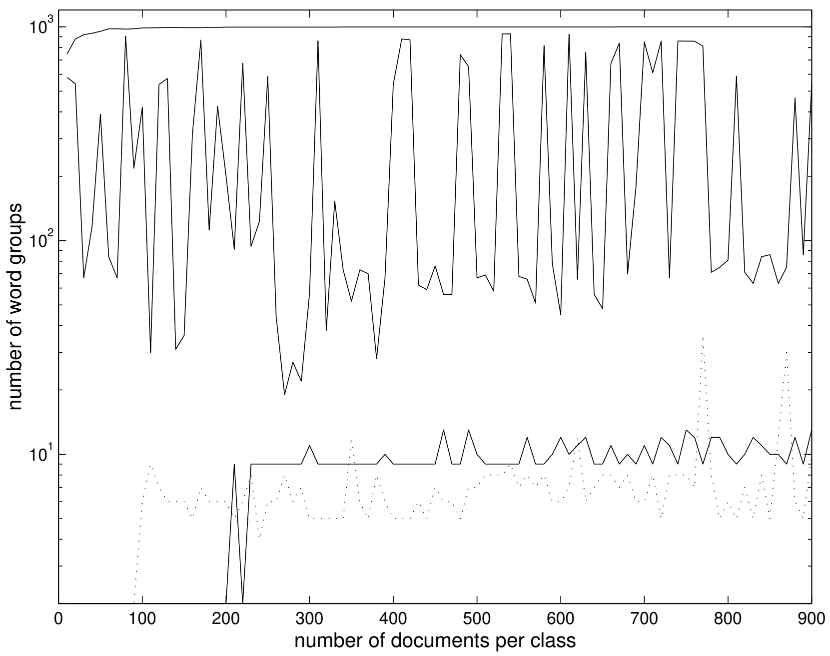

Figure 6.2: The number of selected word groups against sample size.

plot completely we need to look at the next plot, Figure 6.3 on page 31.

In Figure 6.3 we see three curves corresponding to the three lower curves in Figure 6.2. Here, the prediction accuracy of the proposed models on the test set is depicted. The highest curve represents the performance of the best performing model encountered by the algorithm. The dotted curve uses the probabilistic model directly, the lower solid curve corresponds to encoding the exceptions. Note the dramatic performance during the start-up period for both methods. Furthermore, note the sub-optimal performance when using the probabilistic model directly to encode the class labels. We see that the 'coding exceptions' coding scheme, by favoring just slightly more word groups than the direct coding scheme, yields far better results in terms of predictive performance on the test set. We can see that the difference between the optimal performance and the performance of the 'coding exceptions' coding scheme differ in general between five and ten percent. More precisely: outside the 'start-up' period, the mean difference is 7.16% with a standard deviation of 1.70% This conforms nicely to our observations of the difference in optimal performance and the performance at the 'critical point' (see above).

6.3 Conclusions and further thoughts

Conclusions Based on the experiments reported in this thesis and the matters encountered during this research, we are able to draw the following conclusions:

1. We implemented our algorithm using two coding schemes. It was clear that using the 'coding exceptions' coding scheme yields the best performance for our algorithm.

2. The experimental results suggest the proposed method as an adequate method to determine, in a supervised setting, the number of word groups for a classification task. We use the MDL principle to determine automatically the

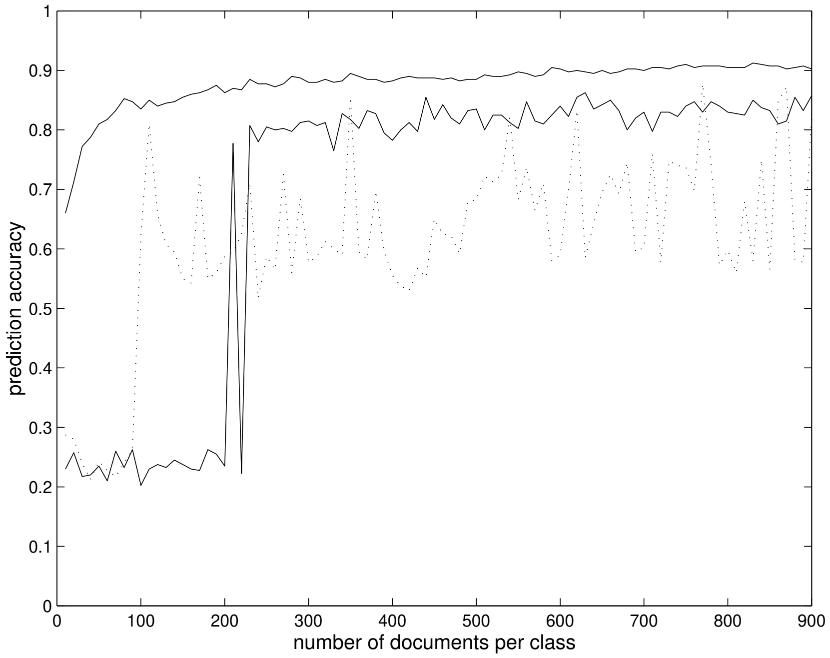

Figure 6.3: Prediction accuracy against sample size.

'optimal' number of word groups where usually this task is done by hand or by a method like cross validation. Our experiments show that the performance of our method is favorable as compared to cross validation, in the sense that our method reduces the number of features aggressively while retaining good performance. Cross Validation has higher predictive accuracy, however the number of features is reduced far less aggressively.

3. We had originally hoped to be able to use the conditional entropy as an useful estimate of the code length needed to encode the class labels. This would have been nice, since then we could use the JS-divergence (which we compute anyway to decide which word groups to merge) as the change in (expected) code length. To do so, we could use the upper bound given by proposition 4.1. However, using the change in mutual information $I(C;\, W)$ (multiplied by the a constant to compensate for the number of words in a text) as an estimate of the change in code length turned out to yield poor performance.

4. One of the main problems we encountered during this research was to find efficient coding schemes to encode the models; we also encountered this problem in [34]. We are led to conclude that this can form serious problems in applications of the MDL principle. In fact, finding a good coding scheme is what remains of *'the modelers art'* when applying Two-Part MDL, although in some situations asymptotic results can be used to find a good coding scheme, see the last paragraph of Section 2.2.2.[6]

Some further thoughts Below, we present some not (yet) worked out matters encountered during this research.

[6] In general we can say that in applications of Two-Part MDL, either 'nice' model classes are used that allow the use of the asymptotically optimal precision to encode the models. In other cases rather 'ad hoc' coding schemes are used.

Comparison to other methods Unfortunately, due to limited time, we were not able to compare our empirical results with results obtained when using other feature extraction methods, like Latent Semantic Indexing. Such a comparison would be very useful to evaluate our method better.

Smoothing of probabilistic models We are not aware of more applications of the 'smoothing method' for probabilistic models implemented by coding exceptions. However, we feel it may be fruitful in other applications where a (naive) probabilistic model is mapping from a high dimensional space to a low dimensional discrete space. It would be interesting to look whether such applications can be found and whether one could make any formal statements (theorems) about the use of this method.

An application A possible application of the algorithm is the following: A well known area of research is user modeling, in which we for example try to build a user profile of a user of a program like NETSCAPE. A very simple user profile would consist of a segmentation of the set of documents a person accesses: one part would be the 'interesting set' and the other part the 'uninteresting set'. We could use our algorithm to speed-up classification of new documents as being 'interesting' or not. Then, we could sort or rank the results of a search engine query on the basis of this classification.

Clustering documents As discussed in [33], we can use generally the same methods to cluster documents as we use to cluster words. We have a strong feeling that a similar method as presented here can be used to find the, in some sense, optimal number of document groups in a large set of documents. This would be very well applicable to indexing documents encountered on the Internet. We could use this method to iteratively cluster documents into categories. Each of the found categories could then (if the method suggests to do so) be split up in again smaller categories.

Bibliography

[1] K. Aas and L. Eikvil. Text categorization: A survey. Technical Report 941, Norwegian Computing Center, June 1999. fetched July 6 2000, `http://www.nr.no/research/samba/tm_survey.ps`.

[2] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval.* Addison-Wesley-Longman, 1999.

[3] D. Baker and A. McCallum. Distributional clustering of words for text classification. In *SIGIR 1998*, 1998.

[4] M. M. Browne. Cross-validation methods. *Journal of Mathematical Psychology*, 44(1):108–132, March 2000.

[5] T. Cover and J. Thomas. *Elements of Information Theory.* Wiley, 1991.

[6] A. Dawid. Present position and potential developments: Some personal views, statistical theory, the prequential approach. *Royal Statistical Society*, 147, 1984.

[7] M. Derthick. A minimal encoding approach to feature discovery. In *Proceedings of AAAI-91*, pages 565–571, Anaheim, CA, 1991.

[8] B. Dom, W. Niblack, and J.Sheinvald. Feature selection for classification using the MDL principle. Computer Science RJ-7117, IBM Almaden Research Center, March 1989.

[9] P. Domingos and M. Pazzani. On the optimality of the simple baysian classifier under zero-one loss. *Machine Learning*, 29(2/3):103–130, November 1997.

[10] S.T. Dumaisand, G.W. Furnas, T.K. Landauer, S. Deerwester, and R. Harshman. Using latent semantic analysis to improve access to textual information. In *Conference proceedings on Human factors in computing systems*, pages 281–285, 1998.

[11] J.H. Friedman. On bias, viariance 0/1 loss, and the curse-of-dimensionality. *Data Mining and Knowledge Discovery*, 1(1):55–77, April 1997.

[12] R. Graham, D. Knuth, and O. Patashnik. *Concrete Mathematics, a Foundation for Computer Science.* Addison Wesley, 1989.

[13] P. Grünwald. A minimum description length approach to grammar inference. Master's thesis, Free University of Amsterdam & Centrum voor Wiskunde en Informatica, 1996.

[14] P.D. Grünwald. *The Minimum Description Length Principle and Reasoning under Uncertainty.* PhD thesis, University of Amsterdam, ILLC, 1998.

[15] I.T. Jolliffe. *Principal Component Analysis.* Springer-Verlag, New York, 1986.

[16] K. Lang. 20 newsgroups data set. fetched May 24 2000, http://www.ai.mit.edu/~jrennie/20_newsgroups/.

[17] D.D. Lewis. The reuters-21578 text categorization test collection. fetched July 26 2000, http://www.research.att.com/~lewis/reuters21578.html.

[18] D.D. Lewis. Feature selection and feature extraction for text categorization. In *Proceedings of Speech and Natural Language Workshop*, pages 212–217. Defense Advanced Research Projects Agency, Morgan Kaufmann, February 1992.

[19] D.D. Lewis. Text representation for intelligent text retrieval: A classification oriented view. In P.S. Jacobs, editor, *Text-Based Intelligent Systems: Current Research and Practice in Information Extraction and Retrieval*, pages 179–197. Lawrence Erlbaum, Hillsdale, 1992.

[20] D.D. Lewis. Naive (Bayes) at Forty: The Independence Assumption in Information Retrieval. In *Conference proceedings of European Conference on Machine Learning*, 1998.

[21] D.D. Lewis and K. Sparck Jones. Natural language processing for information retrieval. *Communications of the ACM*, 39(1):92–101, 1996.

[22] M. Li and P. Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer Verlag, second edition, 1997.

[23] J. Lin. Divergence measures based on the shannon entropy. *IEEE Transactions on Information Theory*, 37(1):145–151, January 1991.

[24] A.K. McCallum. Bow: A toolkit for statistical language modeling, text retrieval, classification and clustering. fetched May 24 2000 http://www.cs.cmu.edu/~mccallum/bow, 1996.

[25] I. Moulinier, G. Raskinis, and J. Ganascia. Text categorization: a symbolic approach. In *Proceedings of the fifth annual symposium on document analysis and information retrieval*, 1996.

[26] B.D. Ripley. *Pattern Recognition and Neural Networks*. Cambridge University Press, October 1995.

[27] J. Rissanen. Stochastic complexity. *Journal of the Royal Statistical Society*, 49:223–239, 1987. Series: B.

[28] J. Rissanen. On optimal number of features in classification. Technical Report RJ-6471, IBM Thomas J. Watson Research Division, 1988.

[29] J. Rissanen. *Stochastic Complexity in Statistical Inquiry*. World Scientific, 1989.

[30] S.J. Russel and P. Norvig, editors. *Artificial Intelligence: A Modern Approach*. Prentice Hall series in Artificial Intelligence. Prentice-Hall International, 1995.

[31] G. Salton and M.E. Lesk. Computer evaluation of indexing and text processing. *Journal of the ACM*, 15(1):8–36, January 1968.

[32] N. Slonim and N. Tishby. Agglomerative information bottleneck. *Advances in Neural Information Processing Systems*, pages 617–623, 2000.

[33] N. Slonim and N. Tishby. Document clustering using word clusters via the information bottleneck method. In *SIGIR 2000*, 2000. To appear.

[34] J.J. Verbeek. Overfitting using the MDL Principle. Master's thesis, University of Amsterdam, 1998. fetched May 24 2000, `http://www.science.uva.nl/~jverbeek`.

[35] S. Wermter, E. Riloff, and G. Scheler, editors. *Symbolic, Connectionist and Statistical Approaches to Learning for Natural Language Processing*, volume 1040 of *Lecture Notes in Artificial Intelligence*, chapter A Minimum Description Length Approach to Grammar Inference, pages 203–216. Springer Verlag, 1996.

[36] E. Wiener, J.O. Pedersen, and A.S. Weigend. A neural network approach to topic spotting. In *Proceedings of the Fourth Annual Symposium on Document Analysis and Information Retrieval*, 1995.

[37] Y. Yang and J.O. Pedersen. A comparative study on feature selection in text categorization. In *Proc. 14th International Conference on Machine Learning*, pages 412–420. Morgan Kaufmann, 1997.

[38] G.K. Zipf. *Human Behavor and the Principle of Least Effort, an Introduction to Human Ecology.* Addison-Wesley, 1949.

Appendix A

Some information theoretic concepts

In this appendix we give definitions of some basic concepts in the field of Information Theory. We follow [5, 32] in our definitions.

Entropy The entropy $\mathcal{H}(X)$ of a random variable X is measure of the uncertainty of the variable. It measures the expected number of bits needed to encode the value of X when using the optimal codeword length of $-\log p(x)$.[1]

$$\mathcal{H}(X) = -\sum_x p(x) \log p(x)$$

Sometimes, we abuse notation and write $\mathcal{H}(p)$ to denote the entropy of a variable that is distributed according to probability distribution p. Also, we write $\mathcal{H}(\epsilon), (0 \leq \epsilon \leq 1)$ for the entropy of a binary variable taking value '0' with probability ϵ (and hence taking value '1' with probability $(1-\epsilon)$).

Conditional Entropy The conditional entropy $\mathcal{H}(X|Y)$ of a random variable X given another random variable Y is the expectation over Y of $\mathcal{H}(X)$.

$$\mathcal{H}(X|Y) = E_Y \mathcal{H}(X|Y=y) = -\sum_{y,x} p(x,y) \log p(x|y)$$

Relative Entropy The relative entropy or Kullback Leibler distance $D(p \parallel q)$ between distribution p and q is a measure of distance between the two distributions. It measures how many bits we would use extra if we would use the code for distribution q to code objects that are emitted according to distribution p:

$$D(p \parallel q) = \sum_x p(x) \log \frac{p(x)}{q(x)}$$

Mutual Information The mutual information $I(X;\ Y)$ is a measure of how statistically independent two variables X and Y are, i.e. how well can we predict the one if we know the other.

$$I(Y;\ X) = I(X;\ Y) = \sum_{x,y} p(x,y) \log \frac{p(x,y)}{p(x)p(y)} = \mathcal{H}(X) - \mathcal{H}(X|Y)$$

[1] See Section 2.2.2.

Jensen-Shannon Divergence The JS divergence $JS_{\pi_1,\ldots,\pi_n}[p_1,\ldots,p_n]$ of distributions $p_1,\ldots,p_n$, each with a probability of occurrence π_i, measures how 'far' these distributions are from their weighted mean.

$$JS_{\pi_1,\ldots,\pi_n}[p_1,\ldots,p_n] = \mathcal{H}(\sum_i \pi_i p_i) - \sum_i \pi_i \mathcal{H}(p_i) = \sum_i \pi_i D(p_i \parallel \sum_i \pi_i p_i)$$

Appendix B

An alternative encoding for the clustering

Remember that the models we encode are basically tables filled with integers. If two words are merged together, they get the same distribution over the classes. Since we only need these distributions the two words will get the same row in the table after scaling to some fixed precision. Now, two approaches suggest themselves:

1. **Two parts:** First code which words belong together using the Stirling numbers of the second kind. Next, encode the reduced table for the word-*groups*. This is the coding scheme we used in Section 4.1.

2. **One big table:** We divide the table in rows that are *originals* and rows that are *copies*. For each row we specify whether the row is a 'copy' or a 'original'. We encode the table row by row; for each there are two possibilities:

   (a) Either we encode the values occurring in the row, we do so for 'originals'.

   (b) Or we encode the number of the row containing the 'original' corresponding to this 'copy'

To make a simple comparison between the two coding schemes we use the following approximation of the number of bits needed when using option two.

- We use one bit in front of the encoding of each row to indicate whether it is a copy or a original. Suppose that there are $|V|$ rows in the table. Let R be the variable taking values 'original' and 'copy' according to the rows in the table. Then we can encode which rows are copies and which are originals in approximately $|V|\mathcal{H}(R)$ bits.

- To encode for all copies which are the corresponding originals. Let c denote the number of 'copy'-rows. Then simply listing all numbers of the corresponding 'original'-rows costs $c \log o = c \log (V - c)$ bits.

In Figure B.1 we present two plots indicating that both methods use a comparable number of bits, although the Stirling method is strictly optimal. This is easily understood, since in the coding scheme just mentioned we count certain clusterings double, that is: there are multiple code words for the same clustering. For example, suppose that row one and two are identical. Then we can encode it as:

1. row one is a original and row two is a copy of row one

2. row two is a original and row one is a copy of row one

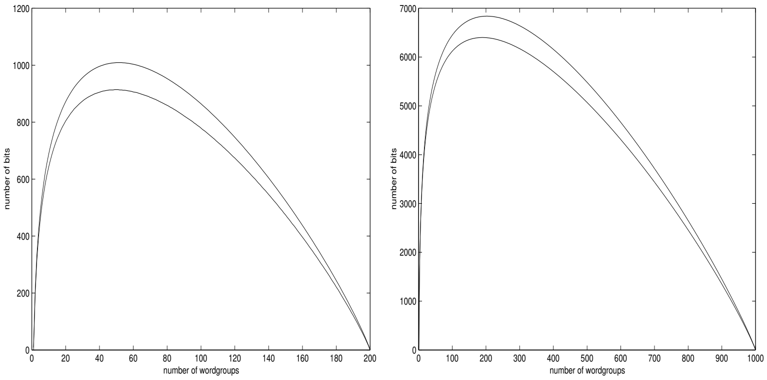

Figure B.1: Number of bits needed to encode the clustering for 200 items (left) and 1000 items (right). The lower lines correspond to method one, the upper lines correspond to method 2.

However, there is one advantage of the second coding scheme: the code lengths it produces can be very efficiently computed as compared to the second order Stirling numbers (for which we have to use the recurrence relation (4.1) on page 17 to compute them).

Appendix C

A proposition on mutual information

In the proof of the following proposition, we use two relations provided in [5]. We provide them here without proof, before we continue to the proposition. Theorem 2.6.6 in [5] states:

Theorem C.1 *Let $X_1, X_2, \ldots, X_n$ be drawn according to $p(x_1, x_2, \ldots, x_n)$. Then*

$$\mathcal{H}(X_1, X_2, \ldots, X_n) \leq \sum_{i=1}^{n} \mathcal{H}(X_i)$$

with equality if and only if the X_i are independent.

It is easy to obtain the following conditioned variant of Theorem C.1:

Theorem C.2 *Let $C, X_1, X_2, \ldots, X_n$ be drawn according to $p(c, x_1, x_2, \ldots, x_n)$. Then*

$$\mathcal{H}(X_1, X_2, \ldots, X_n|C) \leq \sum_{i=1}^{n} \mathcal{H}(X_i|C)$$

with equality if and only if the X_i are independent given C.

Proof We use the definitions of conditional entropy and theorem C.1 to obtain:

$$\begin{aligned} \mathcal{H}(X_1, X_2, \ldots, X_n|C) &= E_C \mathcal{H}(X_1, X_2, \ldots, X_n|C = c) \\ \leq E_C \sum_{i=1}^{n} \mathcal{H}(X_i|C = c) &= \sum_{i=1}^{n} E_C \mathcal{H}(X_i|C = c) = \sum_{i=1}^{n} \mathcal{H}(X_i|C) \end{aligned}$$

The first equality follows from the definition of conditional entropy. The inequality (equality when the X_i are independent given C) follows from Theorem C.1. The second equality follows from

$$\sum_{c} p(c) \sum_{i=1}^{n} \mathcal{H}(X_i|C = c) = \sum_{i=1}^{n} \sum_{c} p(c) H(X_i|C = c)$$

The last equality follows again from the definition of conditional entropy. □

As a corollary (equation 2.21) of the *Chain rule* provided by Theorem 2.2.1 in [5] we have:

$$\mathcal{H}(X, Y|Z) = \mathcal{H}(X|Z) + \mathcal{H}(Y|X, Z) \tag{C.1}$$

Now we have the tools to prove the following proposition:

Proposition C.1 *Let $C, W_1, \ldots, W_n (n \geq 1)$ be discrete random variables such that $W_1, \ldots, W_n$ are identically drawn and independent given C, then*

$$I(C;\ W_1) \leq I(C;\ W1, \ldots, W_n) \leq nI(C;\ W_1)$$

Proof First, we prove the right-hand side inequality. From the definition of mutual information we have:

$$I(C;\ W_1, \ldots, W_n) = \mathcal{H}(W_1, \ldots, W_n) - \mathcal{H}(W_1, \ldots, W_n|C) \tag{C.2}$$

Theorem C.1 together with the fact that all W_i are identically distributed tells us (i) $\mathcal{H}(W_1, \ldots, W_n) \leq n\mathcal{H}(W_1)$. Theorem C.2 together with the independence of the W_i given C and the fact that the W_i are identically distributed tells us (ii) $\mathcal{H}(W_1, \ldots, W_n|C) = n\mathcal{H}(W_1|C)$. Combining (i) and (ii) we can upperbound (C.2) with:

$$n\mathcal{H}(W_1) - n\mathcal{H}(W_1|C) = n(\mathcal{H}(W_1)\mathcal{H}(W_1|C)) = nI(C;\ W_1)$$

For the left-hand side inequality, note that $I(C;\ W) = \mathcal{H}(C) - \mathcal{H}(C|W)$. Therefore it suffices to show that $\mathcal{H}(C|W_1, W_2, \ldots, W_n) \leq \mathcal{H}(C|W_1)$. From equation (C.1) we have:

$$\mathcal{H}(C|W_1, \ldots, W_n) = \mathcal{H}(C, W_n|W_2, \ldots, W_{n-1}) - \mathcal{H}(W_n|W_1, \ldots, W_{n-1}) \tag{C.3}$$

Next, we use Theorem C.2 to obtain:

$$\mathcal{H}(C, W_n|W_1, \ldots, W_{n-1}) \leq \mathcal{H}(C|W_1, \ldots, W_{n-1}) + \mathcal{H}(W_n|W_2, \ldots, W_{n-1}) \tag{C.4}$$

Plugging (C.4) into (C.3) we obtain

$$\mathcal{H}(C|W_1, \ldots, W_n) \leq \mathcal{H}(C|W_1, \ldots, W_{n-1})$$

By multiple applications of the above steps, we obtain the desired result:

$$\mathcal{H}(C|W_1, \ldots, W_n) \leq \mathcal{H}(C|W_1, \ldots, W_{n-1}) \leq \cdots \leq \mathcal{H}(C|W_1)$$

□