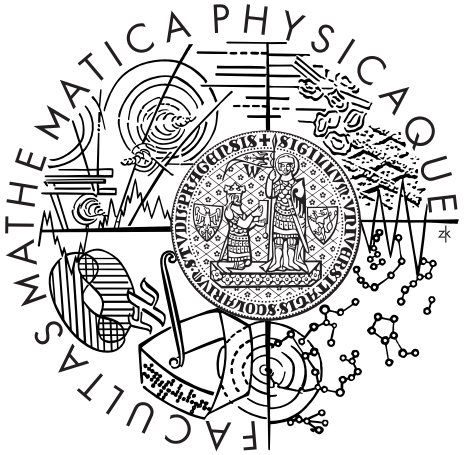
Charles University in Prague
Faculty of Mathematics and Physics

BACHELOR THESIS



Erik Lux

Feature selection for text classification with Naive Bayes

Department of Theoretical Computer Science and Mathematical Logic

Supervisor of the bachelor thesis:	Mgr. Zuzana Reitermanová
Study programme:	Computer Science
Specialization:	General Computer Science

Prague 2012

Thanks to my advisor Mgr. Zuzana Reitermanová, for her enormous support of this project and also for her work that made the project possible in a form in which it is presented in this thesis. Also thanks to all members of my family for their patience and for their useful suggestions.

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

In Prague, May 22, 2012 Erik Lux

Název práce: Výběr příznaků pro klasifikaci textu pomocí Naivního Bayesovského klasifikátoru

Autor: Erik Lux

Katedra: Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: Mgr. Zuzana Reitermanová, Katedra teoretické informatiky a matematické logiky

Abstrakt: Tato práce se zabývá výzkumem v oblasti klasifikace dokumentů. Popisuje již existujici techniky s důrazem na Naivní Bayesův klasifikátor. Zmíněny jsou i některé z metod pro výběr příznaků. Teoretické pozadí je základem pro implementaci klasifikační knihovny založené na metodě Naivního Bayesovského klasifikátoru. Knihovna poskytuje kromě samotného klasifikátoru i paletu nástrojů pro předzpracování textu. Tyto nástroje umožňují práci s rozličným typem dokumentů, ale především značně snižují nadbytečné dimenze vstupních dat. Knihovna je testována na dvou různych referenčních datových sadách na kterých jsou diskutovány rozdíly chování jednotlivých metod pro výběr příznaků. Funkčnost celé knihovny je prakticky ověřena jejím začleněním do open-source emailového klienta Mailpuccino.

Klíčová slova: Naivní bayesovský klasifikátor, Klasifikace textu, Výběr příznaků

Title: Feature selection for text classification with Naive Bayes

Author: Erik Lux

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: Mgr. Zuzana Reitermanová, Department of Theoretical Computer Science and Mathematical Logic

Abstract: The work presents the field of document classification. It describes existing techniques with emphasis on the Naive Bayes' classifier. Several existing feature selection methods suitable for the Naive Bayes' classifier are discussed. This theoretical background is the basis for the implementation of a classification library based on the Naive Bayes' method. Besides the classification program, the library provides a range of document preprocessing tools. They allow to work with different types of documents and, more importantly, they significantly reduce redundant document dimensions. Eventually, we tested the library on two different datasets and compared implemented feature selection methods. The functionality of the whole library is practically verified by including it into the open-source email client Mailpuccino.

Keywords: Naive Bayes, Feature Selection, Text classification

Contents

Introduction **3**

1 Classification **5**
1.1 Information preprocessing 5
1.2 Feature selection 6
1.2.1 Best Individual Features (BIF) 6
1.2.2 Subset Feature Selection (SFS) 9
1.2.3 Feature Transformation (FT) 9
1.2.4 Time complexity 9
1.3 Training 10
1.4 Cross-validation 10
1.5 Classification models 11

2 Classifiers **12**
2.1 Types of documents 12
2.1.1 Decision trees 12
2.1.2 Artificial neural networks 12
2.1.3 K-nearest neighbor algorithm 13
2.2 The Naive Bayes' classifier (NB') 13
2.2.1 Definitions 13
2.2.2 Notation 13
2.2.3 Bayes' theorem 14
2.2.4 Theoretical background 14
2.2.5 Classification risk 15
2.2.6 The NB' 15
2.2.7 Optimality 15
2.3 Similar projects 15
2.3.1 Data mining software in Java named Weka, using the Naive' Bayes Classifier 16
2.3.2 Classifier4J 16
2.3.3 The Naive Bayes' Classifier in C# - NClassifier 16
2.3.4 The RDP Classifier – the Naive Bayes' classifier 16
2.3.5 Mallet 17
2.3.6 jBNC 17

3 Technical documentation (Data classifier CX6) **18**
3.1 Basic information 18
3.2 General description 18
3.3 Project design 18
3.4 Data preprocessing 19
3.5 Data storage 19
3.6 Document classification 19
3.7 Project API 20
3.7.1 General 20
3.7.2 Main package 21

3.7.3 Data types package . . . 21
3.7.4 Plain text package . . . 22
3.7.5 Vector conversion package . . . 22
3.7.6 Serialize package . . . 23
3.7.7 Feature vector package . . . 23
3.7.8 Vector classifier . . . 24
3.8 Exceptions . . . 24
3.9 Libraries . . . 24

4 Application and user manual **25**
4.1 User manual . . . 25
4.2 Cooperation . . . 26
4.2.1 Mailpuccino – extension call . . . 28
4.2.2 Call of extension API . . . 28

5 Results **30**
5.1 Feature selection methods . . . 30
5.2 Overall score . . . 34
5.3 Summary . . . 35

Conclusion **37**

Appendix A **40**

Introduction

Classification or document classification is a well-known problem in Computer Science. The task is to assign a document to one or more categories, based on its content. There are two types of classification: supervised and unsupervised. Supervised classification is based on some external source, e.g. human feedback, which provides information on the correct classification. On the contrary, in unsupervised classification, the calculation must be performed entirely without any external information. This work concentrates only on supervised tasks.

The development of the internet and the growing number of documents available electronically complicate the work with large datasets. There is a need to manage various types of documents more effectively. In order to satisfy the need, documents can be divided according to user criteria. This is a reason why it is still essential to be concerned with the classification problem-solving.

Methods for document classification have been used intensively over the past two decades but their real significance was acquired just recently. In the past few years, there was a rapid progress in this area. It is now possible to choose from a various range of classifiers. Moreover, there are several methods to increase the accuracy of classification. These improvements have made it much easier to deal with classification tasks. However, the onerous task is still in finding a suitable approach for a given problem.

Motivation

The Naive Bayes' classifier is one of computational models suitable for document classification. It is a simple probabilistic method based on Naive Bayes' theorem. It is also considered to be a core technique in information retrieval as it has been used for almost 40 years. Until now, a high number of books and manuscripts in the field of information retrieval have been written on the Naive Bayes' and the classifier has never lost its popularity. On the contrary, only recently it has emerged again as a focus of research in the area of machine learning. It is currently experiencing a renaissance in the field.

Implementation of the Naive Bayes' classifier is a simple task. The fact that makes it more challenging is that the accuracy of classification can be increased considerably by document preprocessing techniques. Most of the research in the field of document preprocessing concentrates on filtering important words out of a document. After a word filter (sometimes referred to as dimension reduction) is employed, the document is represented by a set of characteristic words (features).

Preprocessing helps the classifier to make more accurate decisions. The problem is that it directly depends on a variety of factors (e.g. the length of a document or the type of words). Therefore, it is very difficult to find a single approach which would provide efficient preprocessing in all cases. There have been developed many approaches to the problem that tried to improve the Naive Bayes'

classifier in their own way. Some of them succeeded, some not.

Most of the existing implementations of the Naive Bayes' classifier employ only one of the preprocessing methods. Therefore, we decided to implement the Naive Bayes' classifier together with various preprocessing methods in order to determine a stable and accurate classification approach. Moreover, we intend to write the classifier as a classification library. This makes the classifier more flexible and allows be used in more programs. In addition, we attempt to test implemented preprocessing methods on two different datasets: 20 Newsgroups and Reuters-21578.

Structure of the thesis

In the following chapter, we provide a brief introduction to the document classification, presenting a variety of different methods and classifiers. Chapter 2 explains a theoretical background of the Naive Bayes' classifier and it discusses several existing implementations of the Naive Bayes'. Chapter 3 and 4 describe a technical and user documentation of the project respectively. Chapter 5 shows the test results and some statistic information. Finally, the whole thesis is summarized in chapter Conclusion and the cd with the source code of the library and the executable application Mailpuccino is attached.

1. Classification

In the beginning there is a given text-classification problem. In order to describe and solve it properly, it is useful to divide the problem into smaller subproblems, including search for significant information, information preprocessing, training and document classification.

Information significance depends on the problem definition. Text classifiers usually require plain text documents. This causes problems and inaccuracies during classification. The solution is to construct a parser to retrieve the most relevant tokens out of documents. The parser analyzes given data and extract only textual information.

1.1 Information preprocessing

Information preprocessing starts with document indexing. Every term in the document takes an index or a position mark. The idea is to consider a document as a vector of terms. The approach is known as Vector space representation.

There arises a problem, the representation does not solve by itself. It is the problem of high dimensional data. In order to deal with it, the term-elimination techniques are often applied.

Firstly, some parts of English vocabulary are redundant. They do not carry semantic meaning. Therefore, they are not necessarily essential. These parts are known as stopwords, e.g., conjunctions, prepositions and interjections. They are simply eliminated from documents. On the other hand, some parts of English vocabulary carry meaning, they even have the same base word, but they take different word forms. These are called stemming words, e.g., train, trained and training. They are reduced to their bases [11].

Secondly, it is quite common to employ techniques that minimize the number of terms in a vector. Such techniques are applied not only because of their easy implementation but merely because of their small computational time. They select a subset of initial terms and create a new vector [6]. The selected terms are referred to as features, the vector as a feature vector and the representation as Feature vector representation.

The indisputable advantage of Feature vector representation is that it could be used by both, instance-based and model-based classifiers (Section 2.1). However, the representation does not capture all important structural information and therefore, it is not convenient enough for representation of web documents.

Nowadays, there exists one other way to represent a document that statistically outperforms the others and also satisfies the classification of web pages. It is a recently developed graph based document representation, using the k-nearest

neighbor classification algorithm.

Although it presents much better performance on web pages, the problem is that, the eager, model based classifiers, cannot use the representation directly because they are restricted to work with Feature vector representation. The problem is solved by new hybrid representations, combining previous two representations [7].

Feature vector representation is considered to be a method that aims at making text classifiers more efficient and accurate. Various different methods have been developed to optimize the quality of classification, but without obtaining any general results. It is still a difficult task to select a suitable method for a given problem. Before the description of concrete selection methods is discussed, a framework for feature selection is proposed along with some theoretical underpinnings.

1.2 Feature selection

1.2.1 Best Individual Features (BIF)

The BIF technique uses an evaluation function that is applied to a single term. Scoring individual terms can be performed by defined measures, e.g., term frequency, document frequency and mutual information. These measures are often defined by some probabilities which are estimated by some statistical information found across the training data (Section 1.3). The notation for these probabilities is given below.

- P(c): the probability that a document belongs to category c.
- $P(\neg c)$: the probability that a document is independent of category c.
- P(t): the probability that a document contains term t.
- $P(\neg t)$: the probability that the document does not contain term t.
- P(t',c'): the joint probability of variables t' $\in \{t, \neg t\}$ and c' $\in \{c, \neg c\}$. For example, P(t,$\neg c$) denotes the probability that a document contains term t and is independent of category c.

In order to work with some statistical information, we use the following notation for the training data.

- $\{c_i\}_{i=1}^{m}$: the set of categories.
- D: the set of the training data.
- T: the set of all terms in D.
- N: the size of D.
- N_c^t: the number of documents that contain term t and belong to category c.

- $N_c^{\neg t}$: the number of documents that do not contain term t and belong to category c.
- $N_{\neg c}^{t}$: the number of documents that contain term t and are independent of category c.
- $N_{\neg c}^{\neg t}$: the number of documents that do not contain term t and are independent of category c.
- N^{t}: the number of documents that contain term t.
- $N^{\neg t}$: the number of documents that do not contain term t.
- N_c: the number of documents that belong to category c.
- $N_{\neg c}$: the number of documents that are independent of category c.
- v(d,t): the value of term t $\in$ T for specific document d.

Term frequency (TF)

Term frequency evaluates terms in the vector according to their frequency in the training data [7].

Document frequency (DF)

According to document frequency, [7]

$$DF(t) = \sum_{i=1}^{m} N_{c_i}^{t}. \tag{1.1}$$

Mutual Information (MI)

Mutual information measures how much information the presence or the absence of term t contributes to making the correct classification decision on category c. Formally:

$$MI(t,c) = \sum_{t' \in \{t,\neg t\}} \sum_{c' \in \{c,\neg c\}} P(t',c') \log \frac{P(t',c')}{P(t')P(c')}. \tag{1.2}$$

MI is estimated as follows [3]:

$$\begin{aligned} MI(t,c) = & \frac{N_c^t}{N} \log \frac{NN_c^t}{N_c N^t} + \frac{N_{\neg c}^t}{N} \log \frac{NN_{\neg c}^t}{N_{\neg c} N^t} \\ & + \frac{N_c^{\neg t}}{N} \log \frac{NN_c^{\neg t}}{N_c N^{\neg t}} + \frac{N_{\neg c}^{\neg t}}{N} \log \frac{NN_{\neg c}^{\neg t}}{N_{\neg c} N^{\neg t}}. \end{aligned} \tag{1.3}$$

Information gain (IG) and Information gain ratio (IGR)

To understand information gain, the concept of Entropy must be understood first. Entropy is a measure of how pure or impure a category is. It is evaluated for the whole set of documents D as below:

$$H(D) = \sum_{i=1}^{c} -P(c_i) \log P(c_i), \tag{1.4}$$

IG is frequently employed as a criterion of term quality. The information gain for document $d \in D$ is defined as follows (note the logarithm is still base 2 in the following samples):

$$IG(D,t) = H(D) - \sum_{w \in v(t)} \frac{|\{d \in D | v(d,t) = w\}|}{|D|}$$
$$H(\{d \in D | v(d,t) = w\}). \tag{1.5}$$

To make the value of Information gain for each term more accurate, IG(D,t) is divided by the Intrinsic value of term t. It is actually the entropy of D with respect to the values of term t. Intrinsic value for term t is estimated as follows:

$$IV(D,t) = - \sum_{w \in v(t)} \frac{|\{d \in D | v(d,t) = w\}|}{|D|}$$
$$\log \frac{|\{d \in D | v(d,t) = w\}|}{|D|}. \tag{1.6}$$

The ratio between Information gain and Intrinsic value is called Information Gain Ratio. It penalizes terms by incorporating a term, called Intrinsic value, that is sensitive to how broadly and uniformly the attribute splits the data [18] [12]:

$$IGR(D,t) = \frac{IG(D,t)}{IV(D,t)}. \tag{1.7}$$

χ^2 statistics (X2)

This technique evaluates terms in a vector by the measure of the independence between term t and category $c_i, t \in T, 1 \leq i \leq m$. The formula below demonstrates the calculation:

$$\chi^2_{avg}(t) = \sum_{i=1}^{m} P(c_i) \chi^2(t, c_i), \tag{1.8}$$

where $\chi^2(t, c_i)$ is computed as follows [7] [3]:

$$\chi^2(t, c_i) = \frac{N * (N_c^t N_{\neg c}^{\neg t} - N_c^{\neg t} N_{\neg c}^t)^2}{(N_c^t + N_c^{\neg t}) * (N_{\neg c}^t + N_{\neg c}^{\neg t}) * (N_c^t + N_{\neg c}^t) * (N_c^{\neg t} + N_{\neg c}^{\neg t})}. \tag{1.9}$$

1.2.2 Subset Feature Selection (SFS)

There are two traditional approaches such as a sequential forward selection and a sequential backward selection. They both represent this type of feature selection. Sequential forward selection starts initially with an empty set of features. In each iteration, exactly one term is added to the set. To determine which term to add, the selection tentatively chooses one term that is not already in the set and tests the accuracy of classification based on the current features. The term with the highest score is eventually added.
The algorithm usually ends when there is no other term that would result in an increase in accuracy. Alternatively, it may end the moment it reaches the desired number of features.
In contrary, the sequential backward selection starts with all terms in the set. In each iteration, it eliminates the term that achieves the highest accuracy gain.
Concrete SFS methods differ in a way, they measure the accuracy of classification.

Sequential feature selection (SFSMI)

SFSMI works with the pre-calculated value of MI. Let V be the initial vector and t the term selected by MI . The method finds term t' maximizing MI(t). Let U be $V\backslash\{t'\}$ and S be {t'}. The next selection step is a loop that runs until it reaches the desired number of features:

- For term t $\in U$: compute MI(t,S).
- Find the term t" that maximizes MI(t,S), set $U = V\backslash\{$t"$\}$ and $S \cup \{t''\}$.

The calculation of MI(t,S) or finding the direct relation between t and the whole set of terms is quite difficult. Therefore, several approaches were presented so far to approximate the value of MI. A possible solution is suggested by maxMI algorithm. It substitutes the problematic assignment with the formula:

$$maxMI(t, S) = MI(t) - \max_{s \in S} I(t, s), \tag{1.10}$$

which provides an easier way to deal with the calculation of MI for a term and a term set [7] [13].

1.2.3 Feature Transformation (FT)

The approach does not measure the weights of terms but compacts the vocabulary based on feature occurrences. It learns a discriminative transformation matrix in order to reduce initial vector space. The thesis does not discuss this method at all.

1.2.4 Time complexity

Classified vector is a vector of terms being classified. Let n be the length of the vocabulary and l be the length of the classified vector.
The stemming and stopword process takes O(l) time multiplied by a constant that represents the length of calculation process for one term. This means O(l)

asymptotically.
Let us have a closer look at feature selection methods. Each method searches the whole system of the vocabulary. It iterates over each term in every single document. This means the time of $|categories| * |documents| * |terms|$, approximately equaling to O(n). Then, it iterates over classified vector, which takes O(l). All together it takes O(ln). Assuming that the classified vector is probably much smaller than the vocabulary, this takes linear time O(n). Let vocabulary search be the name of all these iterations.

- DF – Vocabulary search is performed once. This means linear time O(n).
- TF – Vocabulary search takes O(n). This takes O(n) as well.
- MI – To provide the number of documents that contain term t and belong to category c, the vocabulary is searched during the vocabulary search. The information if term t is present or absent is searched for in each iteration of the vocabulary search. The same search is applied for the information if document d belongs to category c or does not. This takes O(l)$O(n^2)$. Assuming that l $\ll$ n, it is quadratic time $O(n^2)$.
- IG – The vocabulary is searched in every iteration of the vocabulary search. This means quadratic time $O(n^2)$.
- $X2$ – The method applies the same principle of the vocabulary search in the vocabulary search as Information gain. This takes quadratic time. $O(n^2)$.
- SFSMI – Let k be the final number of features in a feature vector. It is guaranteed that the calculation complexity would be at least quadratic because of MI. The vocabulary search in the vocabulary search is performed for each of k features. It is quadratic in the worst case scenario. This takes O(k)$O(n^2)$ time. Assuming that k $\ll$n, it is quadratic time $O(n^2)$ asymptotically [18].

1.3 Training

From now on the term document is used for a feature vector. Before any classifiers are applied to document recognition, they need to gain certain knowledge of a given problem. The knowledge is provided by the documents, called training documents. Classifiers store the knowledge into their data structures.

Classifiers usually do not store all training documents. Documents are divided into two partitions instead. The first partition is stored (known as training data). The second partition is intended to become testing data. Division ratio depends on user preferences (typically around 50 % for training and 50 % for testing).

1.4 Cross-validation

The technique is sometimes known as rotation estimation. It assesses how the results of a statistical analysis on training data generalizes to independent testing data. It is used to estimate how accurately a classifier, working with the

vocabulary, performs in practice. This is achieved by selecting different subsets of training documents to form the training data set and the testing data set in multiple rounds.

1.5 Classification models

Classification model defines the internal representation of feature vectors. Two basic models are proposed below [6]:

- Multivariate model is an older model. It handles a vector as a binary vector that carries only the information whether a current term is present or absent in the vocabulary. Recently, it is not used very often because it lacks the ability to utilize term frequencies.
- Thus a multinomial model is introduced as an alternative model. It treats a document as an ordered vector of term occurrences. Two serious problems are encountered while working with the model:
  - Rough parameter estimation – the training data contain a variety of documents. Some of them could be much longer than the others. Although their length is rarely related to their importance, these documents have a bigger influence on the calculation.
  - Insufficient number of the training data – this is a general problem. Some of the categories are poorly equipped with data. They need more training.

2. Classifiers

2.1 Types of documents

Approaches to document classification can be divided into two groups. One is known under the name model-based classifiers and the other is called instance-based classifiers.

Model-based classifiers are built upon a mathematical model. Therefore, their calculation steps use the basic principle of the current model. They generalize the problem and apply the principles. Examples of model-based classifiers are methods such as Naive Bayes', expectation maximization, latent semantic indexing, artificial neural network or decision trees.

On the other hand, instance-based learning or memory-based learning is a family of learning algorithms that, instead of performing explicit generalization, compare new problem instances with instances included in the training data, which have been saved into the memory. Instance-based learning is a kind of lazy learning. It is known under the name of instance-based because it makes hypotheses directly from the training set of instances. As an example of instance-based classifiers is considered k-Nearest neighbor algorithm.

The notation for the evaluation of classifiers is provided before some concrete methods are discussed. Target function is a function that assigns each document into a category. This function models reality. Concept is a function that is made and improved by a classifier to approximate the target function.

2.1.1 Decision trees

Decision tree learning ([12]) is a method for approximating a discrete-valued target function, where the concept is represented as a tree. Such tree could be transformed into a set of if-then rules to improve human readability.

The approach classifies documents by sorting them down the tree from the root to some leaf. Leaves provide the classification of the document. Each node in the tree represents a test on some feature of the document. Each branch that descends from the node is assigned with one of the possible values of the feature.

A document is classified, starting at the root node, testing the feature specified in the node and moving down the corresponding tree branch. The process is performed recursively for the subtree rooted by a new node.

2.1.2 Artificial neural networks

Artificial neural networks ([12]) is a robust way to approximate discrete-valued, real-valued and vector-valued target functions. It is one of the most effective models in the field of learning to interpret complex real-world sensor data. It has

been successful in recognition of hand-written characters, spoken words or faces. A survey of practical applications is provided by Rumelhart et al. (1994).

The approach has been inspired by the study of biological learning systems, which are built of complex networks of interconnected neurons. Analogically, Artificial neural networks is built of highly interconnected simple units. Each unit takes a number of real-valued input (it could be the output of another unit) and returns a single real-valued number of output (it could be the input for another units).

2.1.3 K-nearest neighbor algorithm

The approach of K-nearest neighbor algorithm ([12]) assumes that all documents correspond to points in the n-dimensional space R^n. The nearest neighbor of a document is defined in terms of Euclidean distance.

Learning in the algorithm simply consists of storing the training data. When a new document is encountered, a set of similar related documents is retrieved from the memory and used to classify the new document.

The algorithm can construct a different approximation of the target function for each classified document. In fact, it is sufficient enough to construct a local approximation of the target that performs well in the neighborhood of the new document.

All of this could be seen as a significant advantage if the target is a very complex function. On the other hand, the disadvantage is that the cost of document classification could be high. This is because of the fact that nearly all computation takes place at the time of classification.

2.2 The Naive Bayes' classifier (NB')

The NB' ([12] [15]) provides a simple probabilistic approach. It is based on the assumption that the entities of classification are managed by their probability distribution. Optimal performance can be achieved by working with these probabilities in combination with training data.

2.2.1 Definitions

The task is to determine the best category from some space of categories S, given the training data D plus any initial information about prior probabilities of various documents in D. Bayes' theorem provides a direct way to calculate such probabilities.

2.2.2 Notation

P(c) is the initial probability that category c holds before the observation of training data. It is often called the prior probability of c. It may reflect to

any background knowledge about c being the right category or it simply holds a value that is same for each category. P(D) denotes the prior probability that training data D will be observed. $P(D|C)$ is the probability of observing data D, given some documents which already belong to category c. $P(c|D)$ denotes the probability that c is the best category, given the observed training data D. $P(c|D)$ is called the posterior probability of c because it corresponds to the confidence that c is the right category after the training data have been seen.

2.2.3 Bayes' theorem

The theorem is the base of Bayes' learning. It provides a way to calculate the posterior probability $P(c|D)$ from the prior probability P(c), together with P(D) and $P(D|c)$:

$$P(c|D) = \frac{P(D|c)P(c)}{P(D).} \tag{2.1}$$

In many learning cases, some set of candidate categories C is considered. The solution is to find the most probable category c∈C, given the observed training data:

$$Category = argmax_{c \in C} P(c|D), \tag{2.2}$$

Category is an unobserved random variable that denotes the category to a document.

2.2.4 Theoretical background

Discriminative function is a function of a set of variables used to classify a document. Let $D = (F_1, \ldots, F_n)$ be a document – a feature vector. Each feature F_i takes value from its domain Dom_i. Then, the document takes value f. Let $\Omega = Dom_1 \times \ldots \times Dom_n$ be the set of all feature vectors.

A function $g : \Omega \to c, c \in C$. g(f) = Category is a target function to be learned. Deterministic g(f) represents a noise-free target. Such target always assigns the same category to a given document. In general, a target could be noisy which corresponds to a random function g.

A classifier is defined by a (deterministic) function $h : \Omega \to c, c \in C$. It assigns a category to any given document. The approach of Bayes' classifiers is to associate each category c with a discriminative function $k_c(f)$ and then select the category with a maximum discriminant function on a given document:

$$Category = h(f) = argmax_{c \in C} k_c(f). \tag{2.3}$$

The optimal Bayes' classifier $h_{opt}(f)$ uses as discriminant functions the category posterior probabilities given a document (Equation 2.1):

$$k_c^*(f) = P(c|D = f). \tag{2.4}$$

Applying the Bayes' theorem the discririminat function changes to:

$$k_c^*(f) = \frac{P(D = f|c)P(c)}{P(D = f).} \tag{2.5}$$

The P(D=f) is identical for all categories. Therefore, it can be ignored:

$$k_c^*(f) = P(D = f|c)P(c). \tag{2.6}$$

Thus, the optimal classifier looks like:

$$h^*(f) = argmax P(D = f|c)P(c). \tag{2.7}$$

2.2.5 Classification risk

The risk of classifier h is defined as:

$$R(h) = P(h(D) \neq P(g(D)) = \sum_{f \in \Omega} P(h(f) \neq g(f))P(D = f). \tag{2.8}$$

$R^* = R(h^*)$ denotes the Bayes' error. It is said that classifier h is optimal on a given problem if R(h) is approximately equal to R^*. Assuming there is no noise, there is zero Bayes' risk.

2.2.6 The NB'

Direct estimation of $P(D = f|C = c)$ from a given set of training data is hard when the feature space is high dimensional. The NB' classifier greatly simplifies the estimation. It assumes that features are independent given a category. The discriminant function of the NB' is described as follows:

$$k_c^{NB'}(f) = \prod_{1 \leq i \leq n} P(F_i = f_i|c)P(c). \tag{2.9}$$

Surprisingly, although the independence is generally a poor characteristic, the NB' is fiercely competitive with other learning techniques. In many cases it even outperforms these other methods. A detailed comparison is provided in the book by D. Mitchie at el.

2.2.7 Optimality

NB' is optimal for the following cases. Firstly, features are completely independent (this is expected). Secondly, features are functionally dependent (surprisingly good performance).
The dependence distribution is a key aspect which plays a crucial role in the classification. For example, how the local dependence of a feature distributes in each category, how the local dependencies of all features work together. Either they consistently support a certain category or inconsistently cancel each other out and there is no influence left to affect the classification. The complete study on the optimality of the NB' is presented in the paper by Harry Zhang [19]. This paper also provides a sufficient condition for the optimality with the corresponding proof.

2.3 Similar projects

Thanks to the long history of Bayes' classifiers, there is a lot of applications using the techniques. Some of them are well-known projects, described below:

2.3.1 Data mining software in Java named Weka, using the Naive' Bayes Classifier

Weka is a collection of machine learning algorithms for data mining tasks. The algorithms can either be applied directly to a dataset or called from your own Java code. Weka contains tools for data pre-processing, classification, regression, clustering, association rules, and visualization. It is also well-suited for developing new machine learning schemes. Moreover, Weka provides access to SQL databases with java connectivity. It is not directly matched with the input tables for categorization process but can be simply transformed into these tables. The area which is still not included in weka algorithms is sequence modeling.

This cross-platform software provides user except the Naive' Bayes and other basic algorithms even the implementation of Expectation Maximalization algorithm or K-means algorithm. It could be easily manipulated by a graphical interface. Simply, this software is a present of text categorization [16].

2.3.2 Classifier4J

Classifier4J is a Java library designed to do text classification. It comes with an implementation of a Bayes' classifier, and now has some other features, including a text summary facility. It makes the use of vector space representation. Its API works as a spam filter or blog classifier [10].

2.3.3 The Naive Bayes' Classifier in C# - NClassifier

NClassifier is an open source product. It is a .NET library that supports text classification and text summarization. It is a very extensible library consisting largely of interfaces. It includes, out of the box, an implementation of the Bayes' classification algorithm. The classifier is synched closely with Classifier4J project. For example, the database handling in java is replaced with ADO.Net solution. The future perspective of the project is undetermined. It could at least stay with Classifier4J or, at the most, cut into its own direction [17].

2.3.4 The RDP Classifier – the Naive Bayes' classifier

The Ribosomal Database Project (RDP) Classifier, a Naive Bayes' classifier, can rapidly and accurately classify bacterial 16S rRNA sequences into the new higher-order taxonomy proposed in Bergey's Taxonomic Outline of the Prokaryotes (2nd ed., release 5.0, Springer-Verlag, New York, NY, 2004). It provides taxonomic assignments from domain to genus, with confidence estimates for each assignment [2].

2.3.5 Mallet

Mallet is a java-based software for natural language text processing. It provides efficient routines for feature vector conversion. Mallet includes a wide variety of classification tools (the Naive Bayes' and decision trees techniques for instance). Furthermore, Mallet has got code for evaluating its classification algorithms and its efficiency [5].

2.3.6 jBNC

It is a java toolkit, providing methods for training, testing and applying Bayes' Network Classifiers. This software was manly tested in artificial intelligence and machine learning tasks [1].

3. Technical documentation (Data classifier CX6)

3.1 Basic information

Data classifier CX6 is a java toolkit for classification of text documents of different file format. It implements several feature selection methods to be applied with the Naive Bayes' classification algorithm.
CX6 stands for Classifier X of six selection methods. CX6 classifies documents into user-defined categories. Category is a set of documents, containing some metadata information. Document examples, falling within the same category, share a common theme. Consequently, it is user competence to provide such examples.

3.2 General description

CX6 is based on two component model. The components are a classification library and an application. The library is the main component of the project, applicable to a variety of different texts (e.g. newspaper articles, email documents and scientific papers). The library should recognize a given text and categorize it correctly due to its knowledge. The process includes information retrieval from texts, learning from collected samples and testing of the acquired knowledge.
The component is built upon the Naive Bayes' classifier, a simple and quite accurate probabilistic classifier applying Bayes' theorem [6].

The other component of the project is an application built on the library. It provides an extra graphical interface which allows users to perform classification tasks without any direct knowledge of the library interface. The application is not limited to any specific software. On the contrary, it is designed to extend already existing software. In this case, the email client, Mailpuccino, is considered to be the one. The client is extended in a way to categorize incoming mail.

3.3 Project design

As mentioned above, CX6 is based on two component model. In order to perform classification tasks, the application component needs to call the library component. In other words, the application is dependent on the library. Therefore, it is important to establish the communication between them. This is ensured by the public library interface. It provides the following methods: to create a new category, to train a category and to classify an input document into a category.

Before discussing the concrete implementation of CX6, a theoretical basis on document classification related to the library is introduced. It mainly describes some suitable theorems and techniques mentioned in (Chapters 1 and 2).

3.4 Data preprocessing

The input document is a vector of terms. The length of the vector is not known in advance. It is possible that high dimensional data flows through the program. However, not all dimensions are useful. To reduce dimensions, the program proceeds in 6 steps:

1. create an ordered vector of word occurrences
2. leave out the words shorter than two characters
3. eliminate the punctuation and the characters that are not letters
4. leave out stopwords (e.g. modal words and conjunctions)
5. stemm words in the vector (training -> train, train -> train, trained -> train)
6. select top features – the process is completely described in Section 1.2

3.5 Data storage

A newly created feature vector either becomes a part of the training data or the testing data. Firstly suppose, it is used for training. The vector is either added into an existing category or a completely new category is initialized. The vector then becomes a part of the newly created category.
The term vocabulary denotes the system of all documents within their categories. This system is stored and managed by the library itself to provide some statistical information for classification purposes.
Assume the feature vector is intended to be classified. On the basis of the statistical information from the vocabulary, a decision about the vector is made and it is labeled with the name of a particular category. If demanded, the vector could be still trained into the category.

3.6 Document classification

CX6 implements the classifier that uses the multinomial Naive Bayes' model (Section 1.5). The calculation of the classifier is based on the following algorithm:

1. compute a conditional probability that category c is the searched category given the probability that document d belongs to c.
2. determine the most probable category applying the Naive Bayes' function.

Conditional probabilities are estimated using the discriminant function of the Naive Bayes' classifier as described in Section 2.2.6. Then, the Naive Bayes' classifier is applied directly as follows:

$$Category = argmax_{c \in C} \bar{P}(c|d) = argmax_{c \in C} \bar{P}(c) \prod_{1 \leq k \leq n_d} \bar{P}(t_k|c), \qquad (3.1)$$

where t_k denotes to the k^{th} term in document d. The formula contains the probability of $\bar{P}$(c|d) instead of P(c|d) and $\bar{P}$(c) instead of P(c) to emphasize the fact that the probabilities are not real but estimated from the training data. Formula 3.1 contains a high number of multiplications which can cause a problem, especially in the floating point underflow. Therefore, the logarithm function is applied to the formula using the following rule:

$$log(xy) = log(x) + log(y). \tag{3.2}$$

Thanks to the logarithmic monotony, the formula can be rewritten as follows:

$$Category = argmax_{c \in C}[\log \bar{P}(c) + \sum_{1 \leq k \leq n_d} \log \bar{P}(t_k|c)]. \tag{3.3}$$

The parameters $\bar{P}(c)$ and $\bar{P}(t_k|c)$ are estimated by:

$$\bar{P}(c) = \frac{N_c}{N}, \tag{3.4}$$

where N_c denotes the number of documents in class c and N is the number of all documents.

$$\bar{P}(t|c) = \frac{O_{ct} + 1}{\sum_{\bar{t} \in V} O_{c\bar{t}} + 1}, \tag{3.5}$$

O_{ct} denotes the number of occurrences of term t in the training data of category c. The reason why the simpler term $\frac{O_{ct}}{\sum_{\bar{t} \in V} O_{c\bar{t}}}$ is not used is that it leads to an undefined expression if the denominator equals zero. Therefore, number 1 is added to both the numerator and the denominator [3] [4].

3.7 Project API

Project API describes the public interface of the library in combination with implementation details Appendix A on page 40. Most importantly, it presents all public methods to show how to use the library.

3.7.1 General

The library is implemented as a set of java packages working together. The packages are all exported as a jar file. To use the library from another project, the built path of the project has to be set to the external jar file and the main package *NaiveBayes* has to be imported.

The jar provides public methods that access the library. They are described as follows:

- The method *train* takes sample documents and expands the vocabulary. There are implemented four variants of the method *train* differing in their input parameters. All variants accept three parameters. The first two

parameters are the same for each of them: the category name and the document type. The document type flags the input format in order to preprocess the text. It is passed as an integer number (0 – plain text, 1 – html document, 2 – xml document, 3 – email document). The third parameter, the input text, is different for each variant.
The first two variants are: the method *trainFile* and the method *trainString*. Whereas the third parameter of *trainFile* is the file name of an existing input file, the third parameter of *trainString* is the input document represented as a string. In addition, there are two overloaded variants of the methods *trainFile* and *trainString*. Their third parameters are the vector of file names and the vector of strings respectively.

- The method *classify* classifies the input document into a category. Two variants of method *classify* are implemented: *classifyFile* and *classifyString*. Both variants accept three parameters. The first two parameters, the input text and the document type , have the same function as the third and the second parameter of the method *train*. The third one, the selection type, allows to choose one or more feature selection methods to select top features.

- The training data are stored into a file. The name of the file is *catfile*. By default, there is only one file. The property methods *setcategoriesFileName*, *addcategoriesFileName* and *removecategoriesFileName* provide an interface to control serialization, to store the training data into more than one file and to enable more training tasks at once, where the task could be described as a training system of documents related to a certain topic.

- The last method of the library available is *setnumFeat*. It allows to change the default number of selected features. It has an integer parameter.

3.7.2 Main package

This package provides a public interface using the functionality of all packages. Each method of the interface deals with one subproblem of document classification. This basically means that each package could be extended or changed independently of others but under two conditions. The package has to provide the required method and this method has to preserve required parameters and the return type.

3.7.3 Data types package

Vocabulary is internally represented as a vector of categories. Category is a new data type. Each category has member items such as a name, a length and a vector of documents. Additionally, a few setters and getters to add, rename or change the category content are included in a category.
In fact, Document is implemented as another data type. It contains a property vector, which serves as a metadata storage. A piece of metadata information is one property. It also contains a vocabulary vector, which is internally represented as a hashmap of terms and their occurrences.

3.7.4 Plain text package

Document conversion is provided by the package Plain text. Each class represents a specific type of conversion. The conversion tool recognizes the document type and converts its content into plain text. The html, email and xml conversion tools are available:

- The html converter is an extension of the Java HTMLEditorKit class Section 3.9. HTMLEditorKit is a built-in Swing parser to parse HTML and extract the links. The tool removes all html tags and their parameters. It parses plain text between two tags. Unknown tags are not removed. The parser simply ignores wrong written html code (e.g. missing brackets or incorrect tags). This could lead to more serious problem as the parts of the tags remain untouched and harm the training or the classification process. The main reason not to implement a control over wrong code is a variety of mistakes, it could contain. Trying to handle unpredictable mistakes could be less efficient than leaving them out.

- Xml parser just removes specific symbols of xml tagging(",&,',<,>,/) because it takes all tag names and attributes as a part of the document. However, some wrong written xml code remains untouched.

- The raw mail converter is written in a way to extract certain parts of an email. The parts include meta information (e.g. the subject, the sender, the receiver and the email body). It is assumed that an email is a standard mail document.

The package also allows to extract and store some metadata information about documents for technical or statistical purposes. Metadata are represented as a dictionary or a hashmap of key-value pairs. The main key *Type* represents the document type. Other keys depend on the actual document type. For example, a html document contains the key *Title*. An email document contains the key *From* and the key *To* for instance. The tool for metadata extraction is called metadata parser. It searches for appropriate meta tags in documents. If they are found, the parser extracts information from the tags and creates a new key-value pair.

The package is freely extensible by other conversion tools or any conversion methods (these would allow to work with more types of documents and make the library more flexible).

3.7.5 Vector conversion package

The package provides two functions: removal of stopwords and word stemming.

The input document is parsed according to a default stopword file deleting all stopwords. The file can be easily changed or updated by an overloaded variant of the method *remStopWords*, which by default only removes stopwords. The

overloaded method allows to add a new file and use it instead.

Stemming a word means reducing a derived word into its stem or base, which can differ a lot in many situation. A freely distributed Porter stemming algorithm, slightly adjusted, deals with the stemming in the package [14].

3.7.6 Serialize package

In order to calculate statistical information when necessary, the vocabulary is serialized into a file, specified by the internal variable *catfile*. The serialization is performed using the methods: *serjavaObject* and *deserjavaObject*.
It is built in a way to be used as little as possible because this process is time consuming. By default, the changes are not serialized implicitly but this option can be reset, if required.

3.7.7 Feature vector package

The package provides the following BIF (Best Individual Feature) methods: term frequency (TF), document frequency (DF), mutual information (MI), information gain (IG), information gain ratio (IGR) and χ^2 statistics (X2). In addition, the SFS (Subset Feature Selection) method, using the precalculated value of mutual information, (SFSMI) is implemented as well.

The construction of feature vectors has a direct influence on the quality of classification. Therefore, it is necessary to design the package with a certain amount of abstraction.

In order to ensure portability and easy extensibility, we decided to put each selection method into a different class of the package. Such class implements some kind of a general scheme which specifies some parameters and a return type for each selection method. The scheme also contains a structure to capture calculation results. It is implemented as an abstract class.

The class *selector* is constructed. It contains a member field, that represents a final feature vector, a virtual function (that selects top features) and a key-value hashmap (that contains features and the statistical information about them). It also provides a comparator of features. The comparator implements the java comparator interface and compares features according to their values.

The paragraph introduces selection methods. They are represented as classes, extending class *selector*, and their names correspond to appropriate selections. They contain derived items from *selector*. Furthermore, they implement several string-double hashmaps, which store certain statistical information about documents. The key of each hashmap is a concatenated string value of the category name, the mark $ (a separator) and the appropriate vocabulary term. Some metadata information that belongs to a particular key, useful for the following calculation, is stored into the hashmap as a double number.
The following calculation of top features is based on details, precisely described

in Section **??**. Selected features are then put into another map. Afterwards, the map is sorted by the implemented comparator and the topmost features are provided as a final feature vector.

It is still necessary to provide an internal switch among several feature selection methods, more precisely to call a specific method. Class *Top features* implements the requirement in which the member method *selectTopFeatures* is used for the purpose. The method requires certain parameters: the feature vector, the vocabulary as well as the number of features. However, the most notable is the parameter *featSelect*. The parameter is a string built from concatenated integer numbers, which define the type of method selection ("1" – TF, "2" – DF, "3" – MI, "4" – IG, "5" – X2, "6" – SFSMI).

3.7.8 Vector classifier

The whole process of calculation is described in section *The Naive Bayes' classifier (NB')*. Vector classifier strictly separates the calculation of conditional probabilities and the calculation of probabilities for given categories. The reason for the separation is that both calculations are time consuming and it is not necessary to recalculate conditional probabilities each time a document is being classified. Therefore, *Vector classifier* provides two methods: *vectorClassify* and *applyNB*. The method textitvectorClassify is applied to the vocabulary to calculate conditional probabilities whereas *applyNB* is applied to the feature vector to calculate the probability for the given set of categories. The name of a chosen category is eventually provided as a result of classification.

3.8 Exceptions

Generally speaking, there are two types of exceptions: internal and external. Internal exceptions, occurring right in the packages, do not affect the program a lot. The calculation just cannot be processed exactly in the way it is expected to. For example, if the current file of stopwords cannot be found, an exception is thrown but it is almost immediately caught and the program is forced to use the default file instead. There is no need in propagating the exception up the methods' hierarchy.

On the other hand, if an exception is thrown during serialisation, it seriously affects the calculation. The approach is to propagate the exception, show the corresponding error description and eventually terminate the program.

3.9 Libraries

CX6 uses mainly some of the dynamically loadable libraries called in runtime from the Java Class library. These are java.io, java.swing.text.html.parser, java.utils, java.lang, java.swing.text.html, java.awt, java.swing, java.swing.text and java.math.

4. Application and user manual

4.1 User manual

Mailpuccino is a free, fully featured email client written as a java application (see Figure 4.1 on page 25). It supports POP3 and SMTP, has an address book and features to view, save and send attachments. Its source code is distributed under the terms of the GNU (General Public License).

It consists of three panels: left, top and bottom panel. Left panel provides a main menu and mail folders. Top panel displays the content of a currently chosen mail folder. Bottom panel shows the body of a concrete mail in the folder. Moreover, there are two menu items above top panel used to configure a user account and to check folders for a newly incoming mail (Figure 4.2 on page 26).

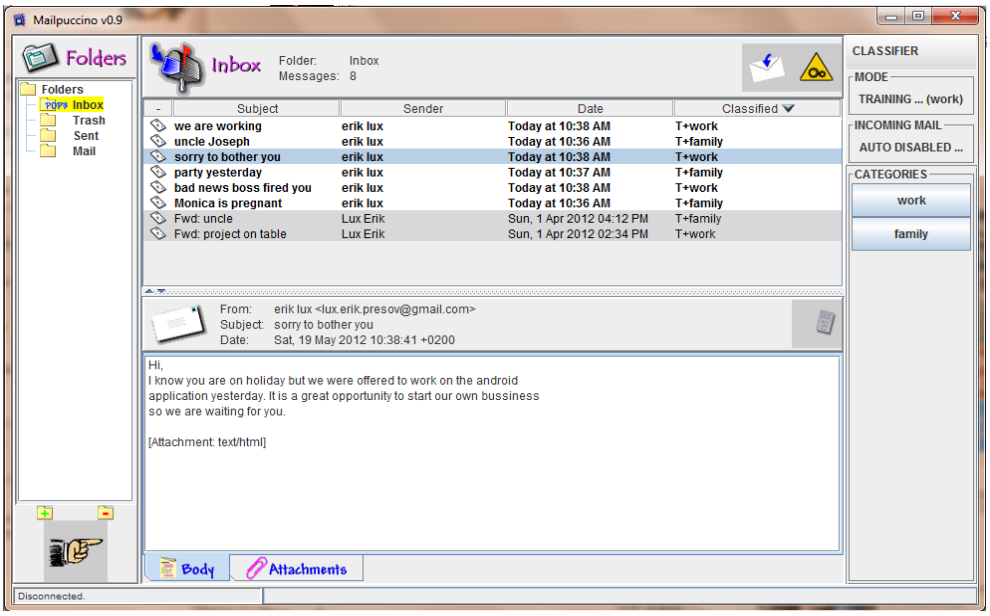


Figure 4.1: The email client Mailpuccino.

The application of the project extends the functionality of Mailpuccino. It provides a graphical interface to classify mail into user-defined categories.

Physically, the extension consists of a right-side panel (Classification panel) that displays a menu and a list of categories (Figure 4.3 on page 27). The menu provides options to add or remove categories. It allows to set parameters of classification: the type of input (the file name or the string), the selection method, the form of classification (train or classify) and the actual number of features. Moreover, it is possible to train or classify a new document by selecting the option *add a document*.
In fact, the application works in two modes: training mode (Figure 4.4 on page 27) and classifying mode (Figure 4.5 on page 28). Therefore, it depends on the mode whether an email is trained or classified.

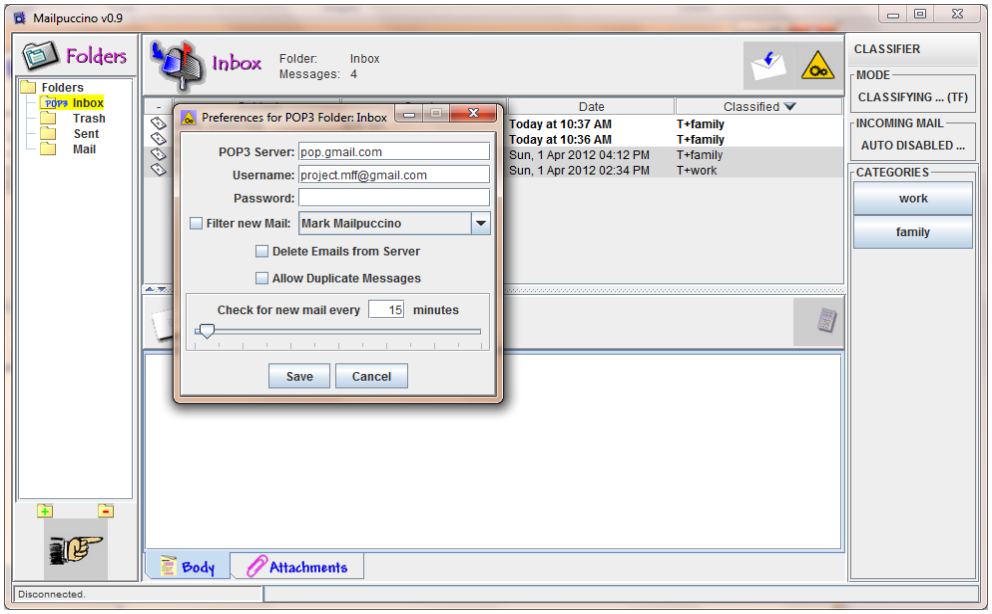


Figure 4.2: Setting user account in Maillpuccino.

The term Mail headers is used to describe column tags, displayed in mail folders, that provide information about the sender, the subject and the date for each received mail.
The extension marks all mail. An email document is marked with "U" (unclassified) if the document has not undergone the classification process yet. The document is marked "T+Category" if the document belongs to the training data of Category. The document is marked with the label "C+Category" if it was classified into Category. Marks are visible among other mail headers in folders of Mailpuccino such as Inbox or Trash. Marking could be also performed implicitly (Figure 4.6 on page 29) by choosing the option *Automatic classification* in Settings (Figure 4.3 on page 27). This option is independent of the selected mode. Therefore, it is possible to perform an automatic classification in the training mode.

The right-side panel displays the set of categories in the vocabulary as a panel of buttons. An action listener is registered on all buttons. The name of the selected button (the name of the category) is changed by the corresponding left-click if it does not exist yet or if it is not the same as the old one.

4.2 Cooperation

It is important to understand the communication among Mailpuccino, the library component and the application component. Mailpuccino calls public API of the extension (package Extension). The extension adds a visual interface to API of the library (package Classifier). In order to use API of the extension within Mailpuccino, there is a need to establish the communication with Mailpuccino. This is defined in next Subsections.

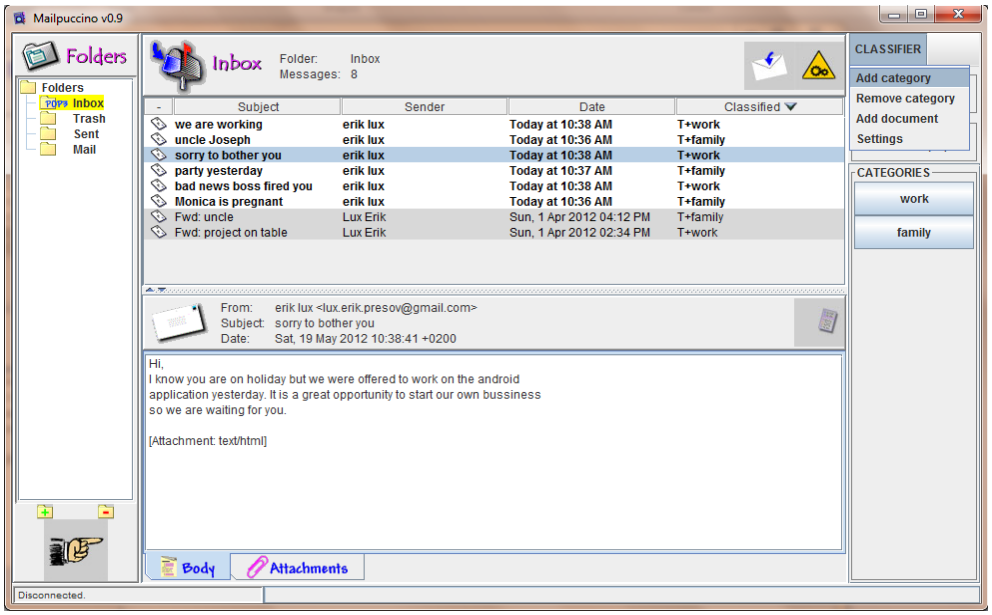


Figure 4.3: Extension menu.

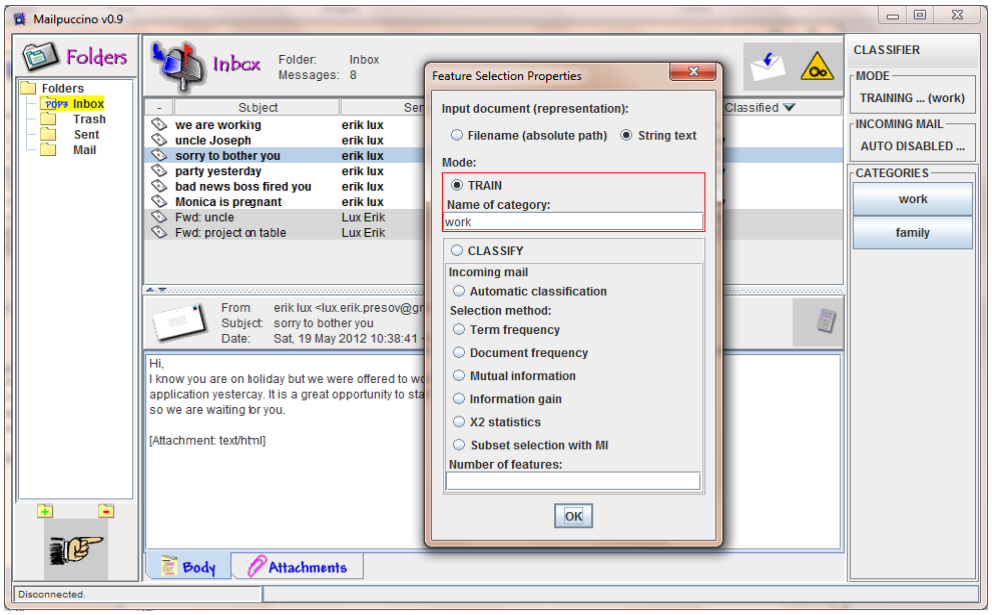


Figure 4.4: The extension of Mailpuccino is in training mode.

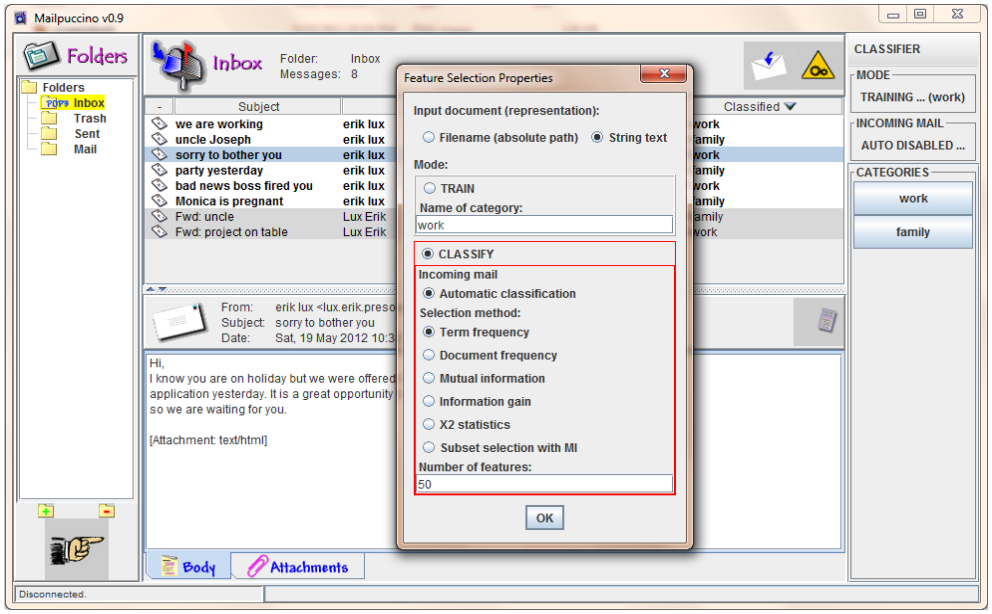


Figure 4.5: The extension of Mailpuccino is in classifying mode.

4.2.1 Mailpuccino – extension call

Extension API is used while working with the right-side panel. The methods of API are called directly when a menu option is selected or the panel of categories is modified.
Extension API is also called while working with mail folders. Options to train or classify mail are added to the the email menu (Figure 4.7 on page 29). Classification parameters are taken from the extension menu.

4.2.2 Call of extension API

The results of classification are propagated using marks. They are displayed in a separate column that is on the left side Classification panel. This column is implemented as a class that extends class Column. It stores the information about the classification into a key-value map. The key of the map is represented as a concatenated string of column headers.
The map is a member item of package Extension. It is not a part of Mailpuccino because it stores the information available at the start of Mailpuccino.

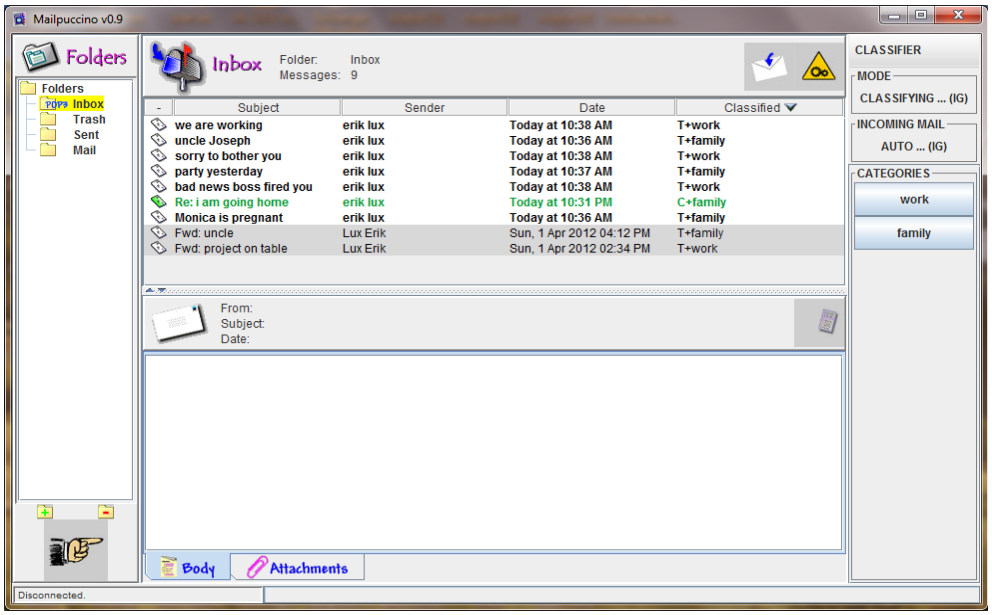


Figure 4.6: A new mail (green color) has been classified to the category family.

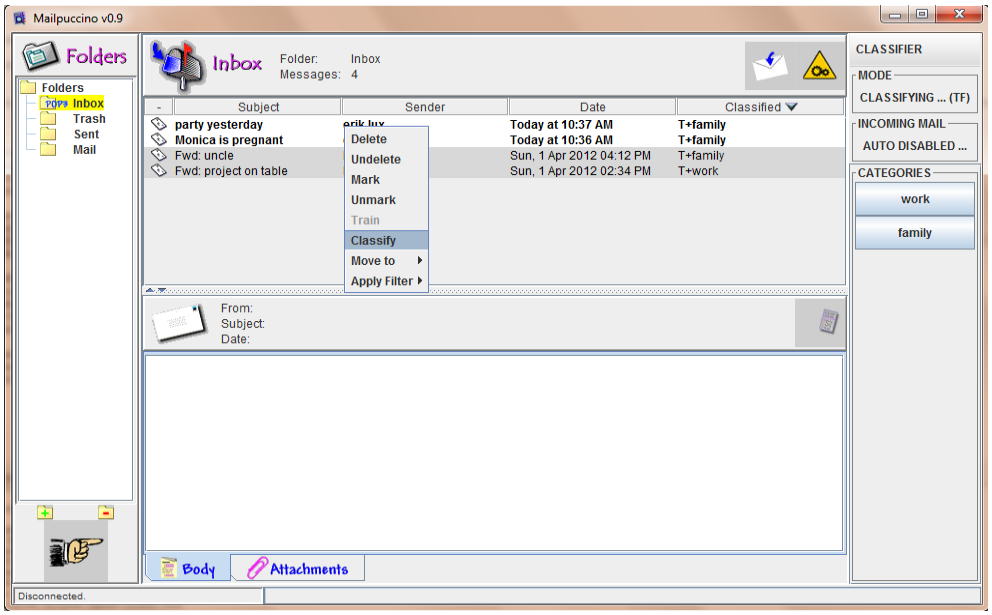


Figure 4.7: Email menu.

5. Results

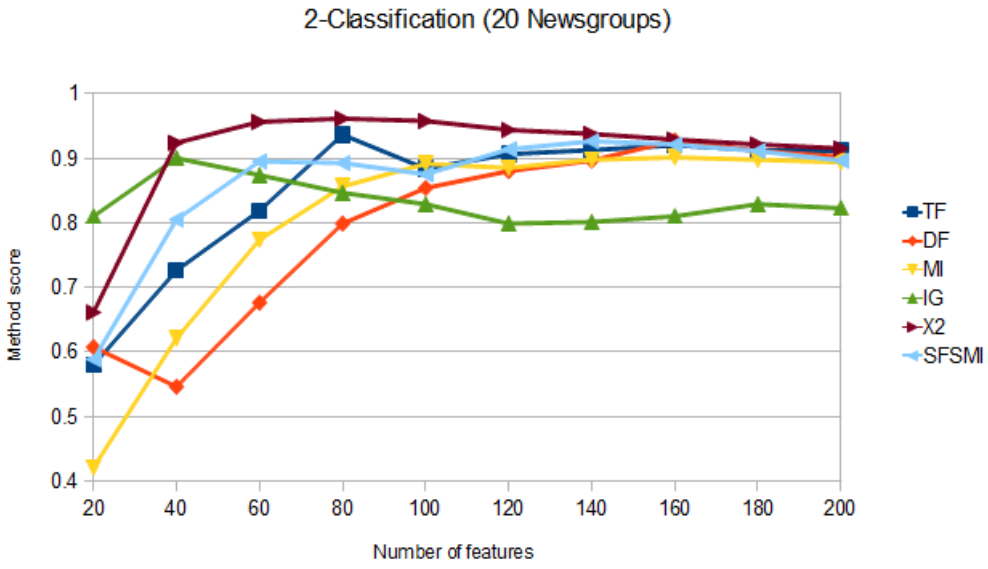
The following chapter shows the test results of the classification library performed on 20 Newsgroup and Reuters-21578 datasets [8] [9]. They depict the scores of selection methods (the ratio of correctly classified documents to the number of classified documents, given the number of selected features). The scores are obtained from two, three and four categorization tasks for each dataset.

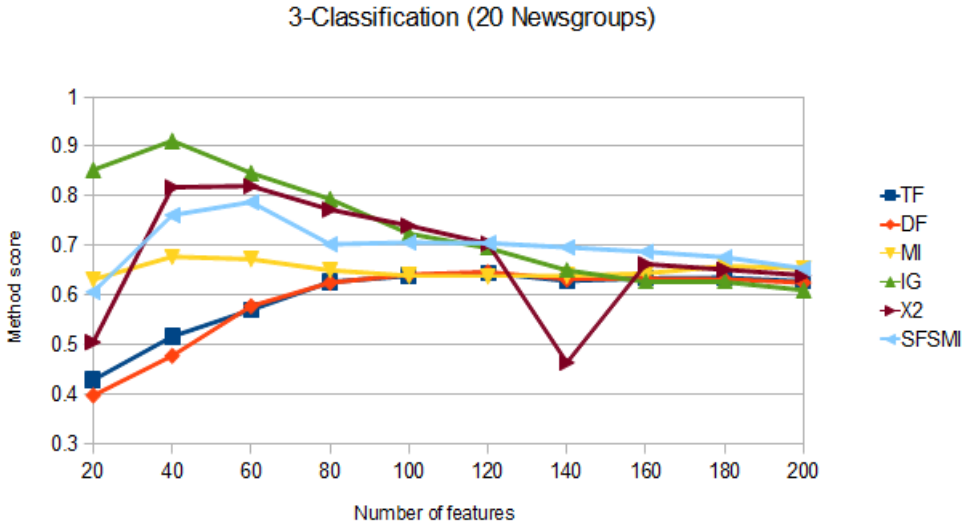
20 Newsgroup dataset contains several categories. There are 1000 documents in a category. 60% of documents are used for training and 40% for testing. An average document is 230 preprocessed words long.

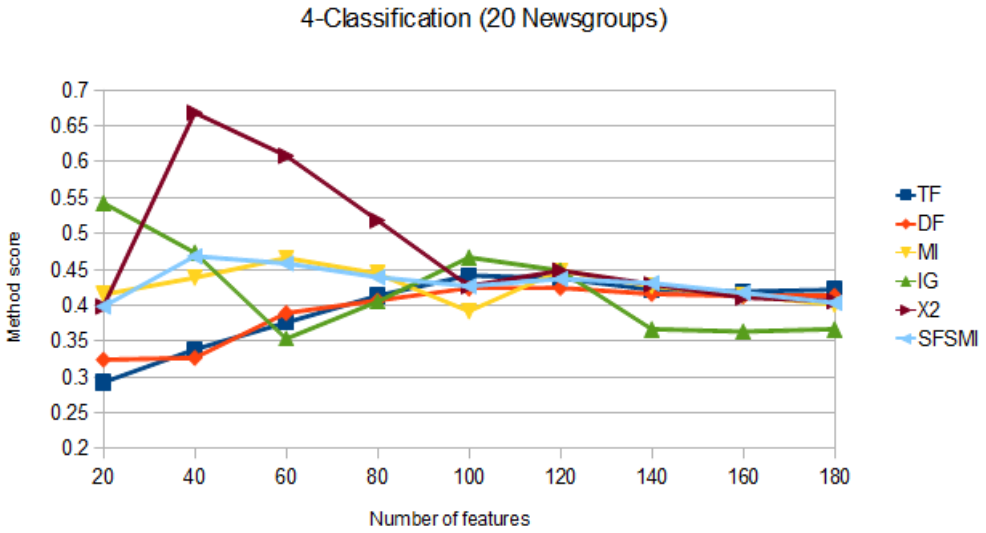
Reuters-21578 dataset contains more than 10 categories. The categories consist of a variable number of documents, which are separated into the training and testing data sets. The separation ratio is 70:30.

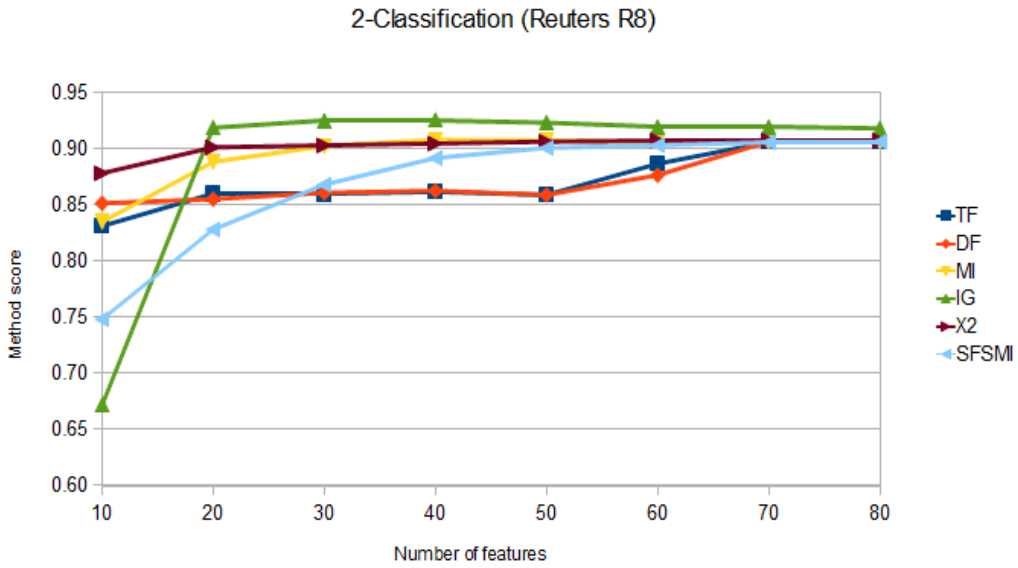
5.1 Feature selection methods

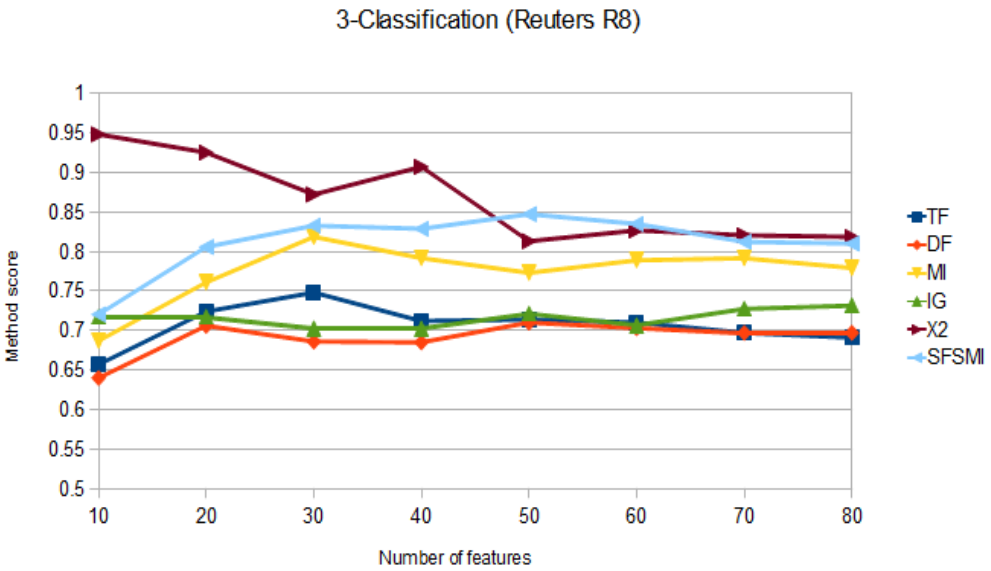
All figures display an interval of selected features. It is the interval of the most significant difference among selection methods.

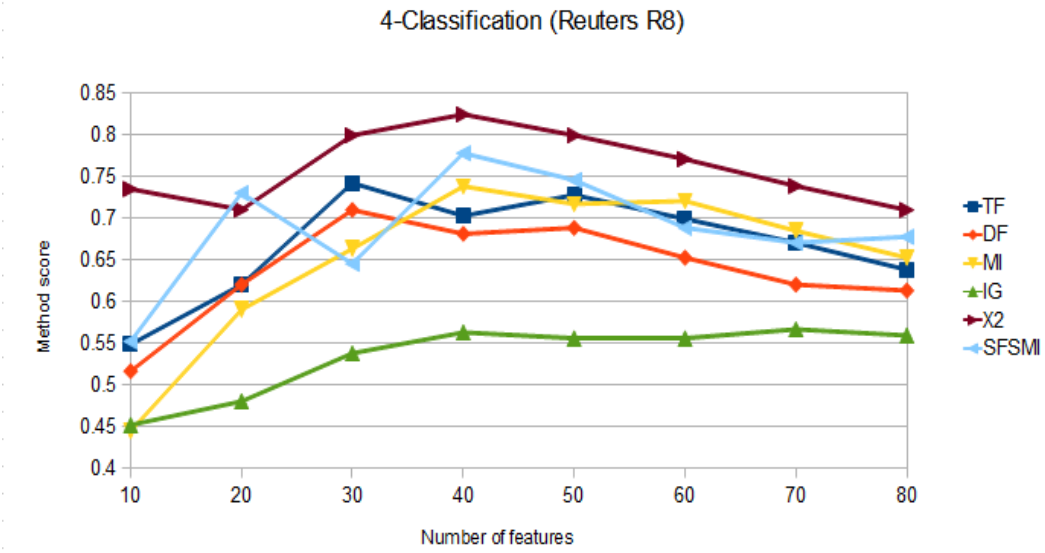



3-Classification (20 Newsgroups)
Method score
Number of features
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
20
40
60
80
100
120
140
160
180
200
TF
DF
MI
IG
X2
SFSMI


4-Classification (20 Newsgroups)
Method score
Number of features
0.7
0.65
0.6
0.55
0.5
0.45
0.4
0.35
0.3
0.25
0.2
20
40
60
80
100
120
140
160
180
TF
DF
MI
IG
X2
SFSMI


2-Classification (Reuters R8)
Method score
Number of features
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
10
20
30
40
50
60
70
80
TF
DF
MI
IG
X2
SFSMI


3-Classification (Reuters R8)
Method score
Number of features
1
0.95
0.9
0.85
0.8
0.75
0.7
0.65
0.6
0.55
0.5
10
20
30
40
50
60
70
80
TF
DF
MI
IG
X2
SFSMI


4-Classification (Reuters R8)
Method score
0.85
0.8
0.75
0.7
0.65
0.6
0.55
0.5
0.45
0.4
10
20
30
40
50
60
70
80
Number of features
TF
DF
MI
IG
X2
SFSMI

5.2 Overall score

Overall score presents the overall results of implemented feature selection methods on the concrete datasets (Tables 5.1 and 5.2). In addition to overall results, there is a table that demonstrates how much time the classification takes 5.3, especially the serialization and the feature selection process. The time is measured in seconds and relates to 100 features and the classified file of 192 terms.

20 Newsgroups	
Selection method	Overall score
TF	0.75
DF	0.73
MI	0.77
IG	0.72
X2	0.84
SFSMI	0.79

Table 5.1: Overall performance of feature selection methods on the dataset 20 Newsgroups including two, three and four categorization tasks.

Reuters-21578	
Selection method	Overall score
TF	0.70
DF	0.68
MI	0.73
IG	0.79
X2	0.81
SFSMI	0.78

Table 5.2: Overall performance of feature selection methods on the dataset Reuters-21578 including two, three and four categorization tasks.

20 Newsgroups			
100 features	2-Classification	3-Classification	4-Classification
Deserialization	18.52	43.03	64.18
TF	0.04	0.06	0.08
DF	0.08	0.11	0.15
MI	0.18	0.39	0.71
IG	0.19	0.38	0.52
X2	0.20	0.41	0.72
SFSMI	1.12	1.40	1.87

Table 5.3: Classification time.

5.3 Summary

The section summarizes the overall scores on datasets (Section 5.2). The best classification results to the given tasks are achieved by the methods X2 (χ^2 statistics) and SFSMI (Subset feature selection with the precalculated value of mutual information). They considerably outperforms the others. However, X2 is more suitable as it consumes less time. The method IG (Information gain) and MI(Mutual information) show a similar performance. Whereas IG is more successful on the dataset Reuters-21578, MI scores higher on the dataset Newsqroups 20. The last two methods TF (Term frequency) DF (Document frequency) are also stable approaches but in this case less efficient that the rest of the methods.

Conclusion

In this thesis I have investigated several feature selection methods in combination with the Naive Bayes' classifier for document classification. The work has involved four different tasks:

- exploration of the existing techniques for text classification
- description of some of feature selection methods suitable to the Naive Bayes' classifier
- implementation of a library for text classification
- verification of its functionality by using it in an open-source product

The main experimental results of this work are the following:

- implementation of a classification library and the interface it provides (Chapter 3 and Appendix A on page 40)
- extension of the open-source email client Mailpuccino (Chapter 4)
- statistical comparison of the implemented feature selection methods (Chapter 5)

The performed experiments fit the general assumption, that quadratic feature methods, e.g., mutual information and information gain, outperform the linear methods, e.g., term frequency and document frequency. We also performed a more detailed comparison and showed which methods tend to be more or less suitable for this type of classifier.

One of the main goals of this work was to create a robust and easily extensible library. The goal was fullfilled by designing the library as a set of independent packages. Together they form the application for document classification. However, they can be used separately to perform specific tasks including serialization, dimension reduction or the classification. Therefore, the packages are not restricted to be used only within the library, and moreover, an arbitrary combination of the packages can be used to solve a concrete problem.

The library was tested on two large datasets: 20 Newsgroups [8] and Reuters-21578 [9]. The tests included two, three and four categorization tasks. They focused on the performance of implemented feature selection methods with a growing number of selected features. The most dramatic difference in performance is presented in Chapter 5.

Finally, the extension of Mailpuccino proposes the way how the library can be used in practice. It adds a graphical panel which wraps the public interface of the library. The panel allows to train and classify email documents available in mail folder quickly and comfortably. Furthermore, if demanded, the classification can be performed automatically as soon as an email is received.

The whole project was written and maintained in the Java language due to its functionality on several platforms including Unix or Windows. It was developed in the Netbeans IDE under the Linux operating system.

There are several possible further research regarding the library which is beyond the scope of this thesis, either in terms of further testing of provided feature selection methods or any additional extensions to the existing software, some of them are described as follows.

The library could be applied to the classification tasks that include more than four categories. It could be also compared to other classification tools based on different classifiers. Moreover, there are some extensions that would improve the performance of the library including: new selection methods, new preprocessing tools or new classifiers.

Bibliography

[1] jbnc - bayesian network classifier toolbox. http://jbnc.sourceforge.net/.

[2] Ribosomal database project - release 10. Website, 1992–2012. http://rdp.cme.msu.edu.

[3] Alpaydin, E. *Introduction to Machine Learning*, 2. ed. MIT Press, Cambridge, MA, 2010.

[4] Jackson, P., and Moulinier, I. *Natural language processing for online applications. Text retrieval, extraction and categorization*, vol. 5 of *Natural Language Processing*. Benjamins, Amsterdam, Philadelphia, 2002.

[5] Kachites, A. Mallet homepage. Website, 2002. http:/http://mallet.cs.umass.edu/.

[6] Kim, S.-B., Han, K.-S., Rim, H.-C., and Myaeng, S.-H. Some effective techniques for naive bayes text classification. *IEEE Trans. Knowl. Data Eng 18*, 11 (2006), 1457–1466.

[7] Kotsiantis, S., and Athanasopoulou..., E. Logitboost of multinomial bayesian classifier for text classification.

[8] Lang, K. 20 newsgroups data set. http://www.ai.mit.edu/people/jrennie/20Newsgroups/.

[9] Lewis, D. D. Reuters-21578. http://www.daviddlewis.com/resources/testcollections/reuters21578.

[10] Lothian, N. Classifier4j - classifier4j. Website, 2003–2005. http://classifier4j.sourceforge.net/.

[11] Manning, C. D., Raghavan, P., and Schütze, H. *Introduction to information retrieval*, 1 ed. Cambridge University Press, July 2008.

[12] Mitchell, T. *Machine Learning*. McGraw-Hill, 1997.

[13] Novovicová, J., Malík, A., and Pudil, P. Feature selection using improved mutual information for text classification. In *SSPR/SPR* (2004), A. L. N. Fred, T. Caelli, R. P. W. Duin, A. C. Campilho, and D. de Ridder, Eds., vol. 3138 of *Lecture Notes in Computer Science*, Springer, pp. 1010–1017.

[14] Porter, M. The porter stemming algorithm. Website, 2006. tartarus.org/~martin/PorterStemmer/.

[15] Rish, I. An empirical study of the naive bayes classifier. In *International Joint Conference on Artificial Intelligence* (2001), American Association for Artificial Intelligence, pp. 41–46.

[16] Weka Machine Learning Project. Weka. http://www.cs.waikato.ac.nz/ml/weka/.

[17] Whitaker, R. Nclassifier - .net text classification and summarization library. Website, October.

[18] Yang, Y., and Pedersen, J. O. A comparative study on feature selection in text categorization. In *Proceedings of the Fourteenth International Conference on Machine Learning* (San Francisco, CA, USA, 1997), ICML '97, Morgan Kaufmann Publishers Inc., pp. 412–420.

[19] Zhang, H. The optimality of naive bayes. In *Proceedings of the Seventeenth International Florida Artificial Intelligence Research Society Conference (FLAIRS 2004)* (2004), V. Barr and Z. Markov, Eds., AAAI Press.

Appendix A

This thesis contains an attached CD with an installer of Mailpuccino, source code of the classification library, javadoc documentation generated from the source code and an electronic version of this work.
The disc contains the following directories:

- SourceCode – contains the complete source code in the form of Netbeans project folder
- Documentation – contains javadoc documentation generated from comments in the source code
- Installer – contains the installer of Mailpuccino with all required files and dependencies
- Thesis – contains this text in PDF format