Text Mining Using Markov Chains of Variable Length

Björn Hoffmeister[1] and Thomas Zeugmann[2]

[1] RWTH Aachen, Lehrstuhl für Informatik VI, Ahornstr. 55, 52056 Aachen
hoffmeister@i6.informatik.rwth-aachen.de
[2] Division of Computer Science, Hokkaido University, N-14, W-9, Sapporo 060-0814, Japan
thomas@ist.hokudai.ac.jp

Abstract. When dealing with knowledge federation over text documents one has to figure out whether or not documents are related by context. A new approach is proposed to solve this problem.
This leads to the design of a new search engine for literature research and related problems. The idea is that one has already some documents of interest. These documents are taken as input. Then all documents known to a classical search engine are ranked according to their relevance. For achieving this goal we use Markov chains of variable length.
The algorithms developed have been implemented and testing over the Reuters-21578 data set has been performed.

1 Introduction

When one is aiming at knowledge federation over the web, one is often looking for information around a specific topic. In a first step, one may find one or more papers dealing with the topic of interest. Then, the next task is to find related papers. Another situation to which our research may apply is to enable documents to communicate to one another when trying to form a knowledge federation over the web. Again, in such cases it may be very important to answer a question like "is document A on the same subject as document B?" If the answer is affirmative, then a federation is made, otherwise it is rejected.

For dealing with such problems, we propose an approach based on Markov Chains of variable length. We exemplify this approach by constructing a search engine taking as inputs papers and returning a list of semantically related papers.

Currently used search engines do not take documents as input. They rely on queries of one or a few words describing the desired information. Basically, there are two different search strategies.

The first concept is based on catalogues. A catalogue contains similar objects, e.g., web-sites about machine learning. Hence, a query to such a catalogue system is answered with a certain set of catalogues. Each of them ideally carries objects relevant to the query. Search engines in libraries and web directories like Yahoo![1]

[1] http://www.yahoo.com

are based on this approach. The quality depends on the quality of the catalogues. Producing good catalogues is still time consuming and expensive.

The second strategy is to perform a full-text search over all available documents. Common web search engines like Google[2] and AltaVista[3] are based on this concept. The disadvantage of a full-text search is the large number of matches. Therefore, a ranking is introduced and only the top ranked documents are returned. Google's main ranking criterion is the linkage rate of a web-site, that is, the more pages link to the document or web-site the higher the rank.

AltaVista uses a syntactical concept. It ranks the results depending on criteria like the positions of and distances between the queried words in the document. So, the alignment of the words should reflect the relevance of the document.

Both strategies have their advantages and disadvantages. Moreover, both approaches fail, for example, if the query allows ambiguities (cf. [13]). And the ranking criteria may overlook relevant documents or give them a low ranking, since simple queries do not allow a fine-grained ranking of relevance.

Now, the idea is to combine the advantages of both approaches. Our search engine takes a set of documents as query, classifies them, and ranks all the documents known by the search engine according to their relevance. To receive a ranking based on semantical relevance we use a model, which can keep more of the meaning of a document than common *data representation models*.

Following Ron *et al.* [18], we tried to use the *variable memory Markov model* defined as a *prediction suffix tree* (abbr. PST). So, we arrive at a *Markov model with variable memory*, or *n-gram VMM model* for short which is used for text representation. The *n-gram VMM model* is learned by statistical inference, a special form of inductive learning. Then we combine text retrieval and text classification.

We shortly outline the underlying mathematical background, describe the workflow of the resulting search engine, and report experimental results.

2 Preliminaries

Natural language is the most common form to exchange information between human beings, e.g., news stories are published in natural language as well as scientific papers. These documents often contain additional information encoded in structured text, like tables or formulas, or in graphical form. However, we shall only use the text in a document. Such a reduction may waste information. But for the particular setting we study within this paper, i.e., the Reuters Data set, it is sufficient. Additionally, all documents in this data set are written in English. Therefore, we restrict ourselves to deal with English texts.

We assume familiarity with formal language theory (cf., e.g., [9]). The word is used as smallest unit. In the literature, one also finds many other possible atomic units. Research has been done using sub-word units like letters or morphemes

[2] http://www.google.com

[3] http://www.av.com

on the one hand and multi-word units, i.e., combinations of one or more words, on the other hand, e.g., see [13], [10], and [19].

We continue with technical notations. $\mathbb{N} = \{0, 1, 2, \ldots\}$ denotes the set of all natural numbers, and $\mathbb{N}^+ = \mathbb{N} \setminus \{0\}$. By Σ we denote a fixed finite alphabet, Σ^* denotes the free monoid over Σ, and $\Sigma^+ = \Sigma^* \setminus \{\epsilon\}$, where ϵ is the empty word. An *n-gram* is a string of $n \in \mathbb{N}^+$ concatenated letters. The set of all *n-grams* over Σ is denoted by Σ^n, where $\Sigma^0 = \{\epsilon\}$. We use $\Sigma^{\leq n}$ to denote $\bigcup_{i=1}^{n} \Sigma^i$.

Our alphabet is the set of all English words, i.e., a suitable subset of the English vocabulary which we denote by $\mathbb{V}$. Thereby we have to assure that $\mathbb{V}$ is a set of indivisible symbols such that there exist an one-to-one mapping between the symbols in $\mathbb{V}$ and the words in the English dictionary. The words of the vocabulary are written in another alphabet which we denote by $\mathbb{A}$. The relation between a word symbol in $\mathbb{V}$ and its representation in $\mathbb{A}^+$ is expressed by a mapping $\omega : \mathbb{V} \to \mathbb{A}^+$, where $\omega(\cdot)$ is injective. This can be easily achieved by introducing a delimiter symbol β such that $\beta \notin \mathbb{A}$.

Note that we use the term *word* to refer to member of $\mathbb{V}$. Therefore, an *n-gram* $s = \sigma_1 \ldots \sigma_n$, $\sigma_i \in \mathbb{V}$, $1 \leq i \leq n$, is a concatenation of n words and a string refers to any *n-gram*, $n \in \mathbb{N}$. A phrase is a meaningful concatenation of two or more words; technically any *n-gram*, $n > 1$, occurring in a document is a phrase. And finally, a term is either a word or a phrase. A document is then a sequence of sentences, where a sentence is a concatenation of words from $\mathbb{V}$.

For dealing with document classification and retrieval we use probabilistic language models. The idea is that documents dealing with different subjects also use a different subset of the vocabulary $\mathbb{V}$ and even different phrases over these subsets. For example, a document about stock exchange might contain words like "hausse" and "baisse", which will almost never appear in a text about machine learning. So, the observation is that texts about different subjects differ in the used words. Furthermore, terms like "machine learning" or "conditional mutual information" are surely not part of texts about stock markets, but the single words "machine", "learning", "conditional", "mutual", and "information" may occur in such a text. Moreover, the idea is to look at how likely a word is, if the previous words are known. In a text about machine learning it is very likely that "machine" is followed by "learning", where in a text about stock market exchange it is probably followed by "manufacture" or "supplier", but not by "learning".

The task of predicting the next word given the previous words is called *language modeling task* and a model solving the task is called a *generative model*, see [13] and [8]. Therefore, we continue with the following definitions.

Definition 1 (Stochastic model). *A stochastic model or process is a sequence of random variables* $(X_t)_{t \in \mathbb{N}}$.

Let us assume every random variable in $(X_t)_{t \in \mathbb{N}}$ has the same range $\mathcal{X}$. Thus, the statistical properties of $(X_t)_{t \in \mathbb{N}}$ are completely determined by the nth-order probability distribution $p(x_0, x_1, \ldots, x_n) := P(X_0 = x_0, X_1 = x_1, \ldots, X_n = x_n)$, $x_i \in \mathcal{X}$, $0 \leq i \leq n$, $n \in \mathbb{N}$, see [16].

Moreover, we use $\mathcal{L}$ to denote the language used by subject S, i.e., $\mathcal{L} \subset \mathbb{V}^*$. We then expect two documents to be about the same subject and hence semantically related, if the subjects of the documents use the same language. But we shall use probabilities instead of absolute statements. That is, we do not wish to decide whether or not a string or a sentence is in $\mathcal{L}$. Instead the language model we are aiming at returns for every string $s \in \mathbb{V}^*$ the probability for s to be in $\mathcal{L}$.

Let S be a subject, let $\mathcal{L}$ be the language of S, and let p_S be the probability distribution underlying $\mathcal{L}$. Furthermore, let M be a *generative model* for S. Thus, M solves the *language modeling task* for S, if $p_S(\sigma|s) = p_M(\sigma|s)$ for every $\sigma \in \mathbb{V}$ and for every $s \in \mathbb{V}^*$, where s is the sequence of all preceding words. Obviously, if M solves the *language modeling task* for S, the strings generated by M are distributed according to p_S and hence M is a probabilistic language model for S.

How many of the previous words are necessary for making a good prediction for the next word? The surprising answer is: most often only a few. For example, if we see the word "machine" in a text about machine learning, "learning" is very likely to be the next word, and knowing the words previous to "machine" does not provide much additional information about the likeliness. Manning *et al.* [13] claim that it takes quite a big effort to beat a *generative model* for natural language, which predicts the next word on the previous two words.

In general, good estimations for the next word in natural language are context dependent. An example is provided by this text. As mentioned before, the word "machine" is very likely to be followed by "learning"; but what about "Markov"? In the following, the words "model" and "chain" occurs after "Markov", but the 3-*gram* "variable memory Markov" is always followed by "model". Hence, we want a model, which can capture this property of natural language.

The model which has the desired properties, is an *n-gram Markov model with variable memory*, *n-gram VMM model* for short, which is defined by a *variable memory Markov model*, *VMM model* for short. A *VMM model* in turn is a special kind of the well-known *Markov model*. So, first *Markov models* are shortly repeated, followed by the definition of the *VMM model*, from which we derive the *n-gram VMM model*. In addition, the classical *n-gram Markov model* is presented and compared to our model, which proves to be superior.

If we regard *generative models* as stochastic processes, any random variable of the process has the property of only depending on the previous variables. A special kind of those dependencies is captured by the *Markov model*, where a random variable depends only on its direct predecessor. We shall see that, despite this restriction, the *Markov model* is a suitable base for a *generative model* for a language. In terms of a *Markov model* we call the value of a random variable a *state* and its range *state space*.

Definition 2 (Markov model). *Let $(X_t)_{t\in\mathbb{N}}$ be a stochastic model and let $\mathcal{X}$ be the state space for all random variables $X_t, t \in \mathbb{N}$. $(X_t)_{t\in\mathbb{N}}$ is a Markov model, iff it meets the Markov assumption*

$$P(X_{t+1} = x_{t+1}|X_0 = x_0, \ldots, X_t = x_t) = P(X_{t+1} = x_{t+1}|X_t = x_t) \ . \tag{1}$$

Let $p(\cdot|\cdot)$ be a function $p : \mathcal{X} \times \mathcal{X} \to [0, 1]$. The Markov model $(X_t)_{t\in\mathbb{N}}$ is homogeneous, iff it fulfills the time invariance assumption

$$P(X_{t+1} = x_{t+1}|X_t = x_t) = p(x_{t+1}|x_t), \quad \textit{for every } t \in \mathbb{N}. \tag{2}$$

p is the Markov core, where $p(x|y) \geq 0$ and $\sum_{x \in \mathcal{X}} p(x|y) = 1$ for all $x, y \in \mathcal{X}$.

If a random variable depends only on its predecessor, the question remains of how to predict the *state* of the initial random variable X_0. This is done by a special initial distribution. A *Markov model* together with an initial distribution for X_0 leads to the definition of a *Markov chain.* We follow the definition given by [4], because it fits our purpose best. Other definitions do not restrict *Markov chains* to be homogeneous, e.g., see [2].

Definition 3 (Markov chain). *A Markov chain is a homogeneous Markov model $(X_t)_{t \in \mathbb{N}}$ with state space $\mathcal{X}$, Markov core p and initial probability distribution π, where X_0 is distributed according to π.*

Before we continue with the definition of the *variable memory Markov model*, we use the *Markov chain* to define a first probabilistic language model, the classical *n-gram Markov model.* It shows how to use a *Markov model* to derive a language model; the *n-gram VMM model* will be defined analogously. We shall also use it to point to the advantages of our model.

The *n-gram Markov model* is a *generative model* predicting the next word in dependence on the previous n words. Since a *Markov chain*, by its definition, predicts the value of a random variable only on the value of its direct predecessor the following construction is necessary which uses overlapping random variables.

Let $(\Sigma_t)_{t \in \mathbb{N}}$ be a sequence of random variables, where each random variable in $(\Sigma_t)_{t \in \mathbb{N}}$ has range $\mathbb{V}$. We define a second sequence of random variables $(S_t)_{t \in \mathbb{N}}$, where each random variable in $(S_t)_{t \in \mathbb{N}}$ has range $\mathbb{V}^n$, $n \in \mathbb{N}^+$. The relation between $(\Sigma_t)_{t \in \mathbb{N}}$ and $(S_t)_{t \in \mathbb{N}}$ is given by the definition of the following equivalence. Let s be an *n-gram* and let $s = \sigma_0 \sigma_1 \ldots \sigma_{n-1}$, $\sigma_i \in \mathbb{V}$, $0 \leq i < n$. Then,

$$S_t = s \quad \overset{def}{\Longleftrightarrow} \quad \Sigma_t = \sigma_0, \Sigma_{t+1} = \sigma_1, \ldots, \Sigma_{t+n-1} = \sigma_{n-1}, \tag{3}$$

for every $t \in \mathbb{N}$. Thus, the random variables S_t overlap, i.e., S_t depends on its predecessors, where the dependency is completely described by the direct predecessor S_{t-1}, $t \in \mathbb{N}^+$.

S_t contains the information about n words and hence, for predicting the value of Σ_{t+n} given the previous n words, the knowledge of the value of S_t is sufficient. We express the probability of the value of Σ_{t+n} in terms of S_t and S_{t+1} as follows. Let $S_t = s_0$, let $S_{t+1} = s_1$, and let $s_0 = \sigma_0 \sigma_1 \ldots \sigma_{n-1}$, where s_0 and s_1 in $\mathbb{V}^n$, $\sigma_i \in \mathbb{V}$, $0 \leq i < n$. From (3) it follows that $s_1 = \sigma_1 \ldots \sigma_{n-1} \sigma_n$, $\sigma_n \in \mathbb{V}$, and hence

$$\begin{aligned} & P(S_{t+1} = s_1 |\, S_t = s_0) \\ =& P(\Sigma_{t+1} = \sigma_1, \ldots, \Sigma_{t+n-1} = \sigma_{n-1}, \Sigma_{t+n} = \sigma_n \\ & \qquad |\Sigma_t = \sigma_0, \Sigma_{t+1} = \sigma_1, \ldots, \Sigma_{t+n-1} = \sigma_{n-1}) \\ =& P(\Sigma_{t+n} = \sigma_n |\, \Sigma_t = \sigma_0, \Sigma_{t+1} = \sigma_1, \ldots, \Sigma_{t+n-1} = \sigma_{n-1}) \\ =& P(\Sigma_{t+n} = \sigma_n |\, S_t = s_0), \qquad \text{for every } t \in \mathbb{N} \ . \end{aligned} \tag{4}$$

Obviously, $(S_t)_{t\in\mathbb{N}}$ fulfills the *Markov assumption* and thus we see how a *Markov chain* can be used to predict a word in dependence on the previous n words. Thus, we arrive at the following definition.

Definition 4 (n-gram Markov model). *Let S denote a subject and let p_S be the probability distribution of the language of S. Furthermore, let $(S_t)_{t\in\mathbb{N}}$ be a Markov chain with state space $\mathbb{V}^n$, Markov core p and initial probability distribution π. $(S_t)_{t\in\mathbb{N}}$ is called n-gram Markov model for S, iff*

$$p_S(\sigma_0\sigma_1\ldots\sigma_{m-1}) = \pi(s_0)p(s_1|s_0)p(s_2|s_1)\ldots p(s_{m-n}|s_{m-n-1}), \qquad (5)$$

*where $s_i \in \mathbb{V}^n$, $s_i = \sigma_i\sigma_{i+1}\ldots\sigma_{i+n-1}$, $0 \le i \le m-n$, for all m-*grams *$\sigma_0\sigma_1\ldots\sigma_{m-1} \in \mathbb{V}^m$, $m \in \mathbb{N}$, $m \ge n$.*

Because p_S is to fulfill Kolmogorov's consistency condition the initial probability distribution π must have the following property, see [1].

Let $s = \sigma_1\sigma_2\ldots\sigma_n$, $\sigma_i \in \mathbb{V}$, $1 \le i \le n$, be an *n-gram*. Furthermore, let $\mathrm{suff}(s)$ denote the longest proper suffix of s, i.e., $\mathrm{suff}(s) = \sigma_2\ldots\sigma_n$. Then π must fulfill the equation

$$\pi(\mathrm{suff}(s)\,\sigma) = \sum_{\sigma'\in\mathbb{V}} p(\sigma|\sigma'\mathrm{suff}(s))\;\pi(\sigma'\mathrm{suff}(s)),$$

for every $s \in \mathbb{V}^n$, where $\sigma \in \mathbb{V}$. We get the desired property, if we define $\pi(\sigma_1\ldots\sigma_n)$ as $P(S_1 = s)$, where $s = \sigma_1\ldots\sigma_n$ for all *n-grams* $\sigma_1\ldots\sigma_n \in \mathbb{V}^n$.

Now, we have a first probabilistic language model. But the size of the *state space* is by definition $|\mathbb{V}|^n$. This will lead to problems if $n \ge 2$ when one wants to learn such a model and the documents are too short (cf., e.g., [2]). For seeing the problem, note that a normal vocabulary of natural language has a size of more than 20.000 words. So, in order to estimate all probabilities described above for an 2-*gram Markov model* one needs a sample of more than $20.000^3 = 8 \times 10^{12}$ words. Obviously, we normally do not possess such a large sample.

Therefore, we want to use the *variable memory Markov model* which has been defined in a different context by Ron *et al.* [18]. A variable memory Markov model is defined as a *prediction suffix tree*(*PST*).

Definition 5 (suffix tree). *Let Σ be an alphabet, let $\mathcal{T}$ be a tree and let $\mathbf{E}$ denote the set of edges between the nodes in $\mathcal{T}$. Furthermore, let each edge be labeled by a symbol $\sigma \in \Sigma$ and each node by a string $s \in \Sigma^*$. The two functions $l_{\mathbf{E}} : \mathbf{E} \to \Sigma$ and $l_{\mathcal{T}} : \mathcal{T} \to \Sigma^*$ return the label of an edge and of a node, respectively. $\mathcal{T}$ is a suffix tree over Σ, iff it has the following properties:*

i) *$\mathcal{T}$ has degree $|\Sigma|$.*
ii) *The root node n_0 of $\mathcal{T}$ has label ϵ.*
iii) *For every node $n_l \in \mathcal{T}$, $l \in \mathbb{N}$, and $n_0 \to n_1 \to \ldots \to n_{l-1} \to n_l$, the walk from the root node n_0 to node n_l, the label of n_l equals the concatenated labels of the passed edges, i.e., $l_{\mathcal{T}}(n_l) = l_{\mathbf{E}}(e_{0,1})\, l_{\mathbf{E}}(e_{1,2})\, \ldots\, l_{\mathbf{E}}(e_{l-1,l})$*
iv) *Neither two edges of one node nor two nodes have the same label.*

Definition 6 (next symbol probability function). *Let Σ be an alphabet and let $\gamma_s, s \in \Sigma^*$, be a function. The function γ_s is called next symbol probability function over Σ, iff it defines a probability distribution over Σ.*

Definition 7 (prediction suffix tree). *Let Σ be an alphabet and let $\mathcal{T}$ be a suffix tree over Σ. $\mathcal{T}$ becomes a prediction suffix tree by expanding the label of every node in $\mathcal{T}$ to (s, γ_s), where s is the label of the node in the suffix tree and γ_s is a next symbol probability function over Σ.*

For the sake of readability, we use the string $s \in \Sigma^*$ synonymously for the label of a node and for the node itself. Ron *et al.* [18] proved the following for *VMM models*. Every *VMM model* can be described by a *Markov chain*, whose size grows exponentially in the maximum depth of the *VMM model*. And almost every *Markov chain* can be described by an *VMM model*. In particular, an *n-gram Markov model* can be simulated by a *VMM model*.

Let $\mathcal{T}$ be a *VMM model* over Σ with maximal depth n. A similar *VMM model* $\tilde{\mathcal{T}}$ over Σ with maximal depth n can be learned with arbitrary precision from an example generated by $\mathcal{T}$ in linear time in the length of the example. The sufficient length of the example is bounded by a polynomial function, which depends on the number of nodes $|\mathcal{T}|$, where we assume $|\mathcal{T}| \geq |\Sigma|$ and $|\mathcal{T}| \geq n$, and on the desired precision.

Using the terminology of *Markov models*, we call the set of nodes in an *VMM model state space* and a single node *state*.

Now we have the desired model, which uses a variable amount of memory. The last step is to define a language model, the *n-gram VMM model*, which is based on a *VMM model*.

Definition 8 (n-gram Markov model with variable memory). *Let S denote a subject and let p_S be the probability distribution of the language of S. Furthermore, let $\mathcal{T}$ be a* VMM model *over $\mathbb{V}$ having maximal depth n.*

$\mathcal{T}$ is called n-gram Markov model with variable memory, or n-gram VMM model for short, for S iff

$$p_S(\sigma_0\sigma_1 \ldots \sigma_{m-1}) = \gamma_{s^0}(\sigma_0)\gamma_{s^1}(\sigma_1)\gamma_{s^2}(\sigma_2) \ldots \gamma_{s^{m-1}}(\sigma_{m-1}), \tag{6}$$

*where s^i, $0 \leq i \leq m-1$, is the longest suffix of $\sigma_0\sigma_1 \ldots \sigma_i$ labeling a node in $\mathcal{T}$, for all m-*grams *$\sigma_0\sigma_1 \ldots \sigma_{m-1} \in \mathbb{V}^m$, $m \in \mathbb{N}$, $m \geq n$.*

So, the *state space* of the *n-gram VMM model* is a subset of $\mathbb{V}^n$, whereas, by Definition 4, the *state space* for the *n-gram Markov model* is the full set $\mathbb{V}^n$.

Here again, we can derive a special property, which $\mathcal{T}$ must have for p_S to fulfill Kolmogorov's consistency condition.

Lemma 1. *Let $\mathcal{T}$ be an n-*gram VMM model *over $\mathbb{V}$. Then the next probability function γ_s must fulfill the equation*

$$\gamma_{\mathit{suff}(s)}(\sigma) = \sum_{\sigma' \in \mathbb{V}} \gamma_{\sigma' \mathit{suff}(s)}(\sigma), \tag{7}$$

for each $s \in \mathbb{V}^+$ labeling a node in $\mathcal{T}$ and for every $\sigma \in \mathbb{V}$.

From the definition of the conditional probability it follows that $p(\sigma|s)$ equals $\gamma_{s'}(\sigma)$ for every $\sigma \in \mathbb{V}$ and every $s \in \mathbb{V}^n$, $n \in \mathbb{N}$, where s' is the longest suffix of s labeling a node in $\mathcal{T}$, and $p(s) > 0$. This notation immediately shows that we have a probabilistic language model, where the prediction of the next word depends on a variable number of previous words.

Let us assume that we want to learn the *n-gram VMM model* $\mathcal{T}_S$, which describes the probability distribution of the language of the subject S. Learning is done from sample strings, where a sample string t for S, $t \in \mathbb{V}^*$, is just a finite string generated by $\mathcal{T}_S$. According to Definition 8, the value $\gamma_s(\sigma)$ equals $p_S(\sigma|s)$, which in turn equals approximately $\tilde{P}_t(\sigma|s)$ for every $\sigma \in \mathbb{V}$ and every $s \in \mathbb{V}^{\leq n}$, if only t is sufficiently large. $\tilde{P}_t(\sigma|s)$ denotes the conditional empirical probability of σ given s achieved from t.

In order to derive the conditional empirical probabilities we need to count $(n+1)$-*grams*. The function $\#_t^{n+1}(\cdot)$ counts the number of occurrences of a certain $(n+1)$-*gram* in the sample string t and is defined as

$$\#_t^{n+1}(s) := \text{the number of occurrences of } s \text{ in } t \ ,$$

where $s \in \mathbb{V}^{n+1}$. Furthermore, let N_t be the number of all $(n+1)$-*grams* in t. Now, we are able to define the empirical probabilities for $(n+1)$-*grams* and from there we derive the desired conditional empirical probabilities:

$$\tilde{P}_t(s\sigma) := \frac{\#_t^{n+1}(s\sigma)}{N_t}, \quad \tilde{P}_t(\sigma|s) = \frac{\tilde{P}_t(s\sigma)}{\sum\limits_{\sigma' \in \mathbb{V}} \tilde{P}_t(s\sigma')} = \frac{\#_t^{n+1}(s\sigma)}{\sum\limits_{\sigma' \in \mathbb{V}} \#_t^{n+1}(s\sigma')},$$

where $s \in \mathbb{V}^n$ and $\sigma \in \mathbb{V}$. The conditional empirical probabilities of lower order are derived by the following recursion:

$$\tilde{P}(\sigma|\mathrm{suff}(s)) = \sum_{\sigma' \in \mathbb{V}} \tilde{P}(\sigma|\sigma'\mathrm{suff}(s))$$

So, $\sum_{s \in \mathbb{V}^{n+1}} \tilde{P}_t(s) = 1$ holds and therefore Lemma 1 is fulfilled by construction.

A catalogue or class is a collection of related subjects, e.g., if the subject of a document is “pruning algorithms for decision tree learning”, it might be part of the classes “decision tree learning” and “machine learning”. Obviously, most classes can be divided into subclasses yielding a hierarchy of classes, e.g., “decision tree learning” is a subclass of “machine learning”. Hence, a subject can be viewed as the indivisible element on the bottom of a class hierarchy.

A class c is a set of documents, i.e., $c = \{d_0, d_1, \ldots, d_q\}$, $q \in \mathbb{N}$. Let S_i be the subject of document d_i, let $\mathcal{L}_{S_i}$ be the language of S_i, and let p_{S_i} be the probability distribution underlying $\mathcal{L}_{S_i}$, $0 \leq i \leq q$. So, the language of c is given by $\mathcal{L}_c = \bigcup_{i=0}^{q} \mathcal{L}_{S_i}$. We denote the probability distribution underlying $\mathcal{L}_c$ by p_c.

Obviously, if each probability distribution p_{S_i} can be described by an *n-gram VMM model*, p_c can be described by an *n-gram VMM model*, too.

The problem of finding the class a document is part of is known as the *text classification* task. The *text classification* task consists of a set of classes $\mathbf{C} =$

$\{c_0, c_1, \ldots, c_r\}$, $r \in \mathbb{N}$, a set of documents $\mathbf{D}$, and a function $k : \mathbf{D} \rightarrow \{0, 1, \ldots, r\}$ called classification rule. $\mathbf{D}$ is the set all documents such that each document $d \in \mathbf{D}$ belongs to exactly one class in $\mathbf{C}$. Then, one wants to find a classification rule k_{opt} approximating k_{true} best, where k_{true} returns the correct class label for each document in $\mathbf{D}$, i.e., if $d \in \mathbf{D}$ belongs to $c_i \in \mathbf{C}$, then $k_{true}(d) = i$.

The definition of the best approximation varies; often one tries to minimize the *error rate* defined as the ratio of misclassifications to the total number of classifications. We follow this definition, because there exists a classification rule known as *Bayes' classification rule* that achieves the minimal *error rate*, if $p(c|d)$ is known for every $c \in \mathbf{C}$ and for every $d \in \mathbf{D}$.

Normally, the classification rule does not work on the documents and classes themselves, but on a model for either the classes or for both, classes and documents. A *text classifier* is defined as a 3-tuple $(\mathbf{C}, M, k)$ consisting of a set of classes $\mathbf{C}$, a model M, and a classification rule k, where k is performed on M. If k is *optimal* for every set of classes with respect to M, $(\mathbf{C}, M, k)$ is called an *optimal text classifier*. If k is *optimal* on the documents and classes themselves, i.e., k does not use a model, then k is called a *perfect text classifier*, see [5].

Finally, we formalize a search engine. In general, a search engine works as follows. The users states a query, the search engine estimates the relevance of each document to the query, the documents are sorted according to the estimates, and finally the list of the sorted documents are returned to the user. In practice, the list is usually truncated but it should contain the maximal possible number of relevant documents. We use the key concepts of *text retrieval* for the definition of a search engine. Note that our search engine uses queries consisting of documents.

Let $\mathbf{Q}$ be a nonempty, finite set of documents, we call $\mathbf{Q}$ a *query*. Let $\mathbf{D}$ be a finite, nonempty set of documents, where $\mathbf{D}$ is split into two sets $\mathbf{R}$ and $\mathbf{N}$. $\mathbf{R}$ is the set of all documents in $\mathbf{D}$ being relevant to query $\mathbf{Q}$ and $\mathbf{N} = \mathbf{D} \setminus \mathbf{R}$ is the set of all irrelevant documents. A *text retrieval* system is an algorithm which assigns a rank $r \in \{1, 2, \ldots, |\mathbf{D}|\}$ to each document in $\mathbf{D}$ such that no two documents get the same rank, where we refer to 1 as the highest rank. A search engine is just a *text retrieval* system.

Let $\mathbf{D}_n \subset \mathbf{D}$ be the set of the n documents having the highest ranks. A *text retrieval* system is called *optimal*, if

$$|\mathbf{R}_n| \text{ is maximal for all } n \in \{1, 2, \ldots, |\mathbf{D}|\} \text{ for every query } \mathbf{Q} \ .$$

Similar to *text classification*, we distinguish between an *optimal* and a *perfect text retrieval* system. An *optimal text retrieval* system performs the ranking task on models of the documents in $\mathbf{D}$. A *perfect text retrieval* system achieves the *optimal* result on the documents themselves. Since we use *n-gram VMM model* as a model for a document we have *probabilistic text retrieval* system. For a *probabilistic text retrieval* system the optimum criterion becomes

$$E[|\mathbf{R}_n|] = \sum_{d \in \mathbf{D}_n} P(\mathbf{R}|d) \text{ is maximal for all } n \in \{1, 2, \ldots, |\mathbf{D}|\},$$

where $P(\mathbf{R}|d)$ is called *probability of relevance*. Robertson [17] has shown that such an *optimal probabilistic text retrieval* system exists and that it can be derived by the *probability ranking principle*(*PRP*). A short definition of the *PRP* is given in [13], p. 538.

> **Probability Ranking Principle.** Ranking documents in order of decreasing probability of relevance is optimal.

Therefore, our goal in a *probabilistic text retrieval* system is to find a suitable model for estimating the *probabilities of relevance*.

We use the following two observations. First, if d and the documents in $\mathbf{Q}$ are about the same subject, d is with high probability relevant to $\mathbf{Q}$. Note that we model d and each document in $\mathbf{Q}$ by a *n-gram VMM model* and compare these *n-gram VMM models*. If the *n-gram VMM models* are similar, we expect d to be in $\mathbf{R}$.

Second, normally d and the documents in $\mathbf{Q}$ belong to the same class, if d is relevant to $\mathbf{Q}$ and vice versa. We shall use a *text classifier* based on *n-gram VMM models* to determine whether or not d and $\mathbf{Q}$ belong to the same class.

3 Learning the N-Gram VMM Model

A *language learner* is an algorithm learning the probability distribution underlying a language. We define it as 5-tuple $(\mathbf{T}_S, \mathbb{V}, n, \mathcal{H}, L)$. $\mathbf{T}_S = \{d_0, d_1, \dots, d_q\}$, $q \in \mathbb{N}$, is a set of documents called training set for S. $\mathbb{V} = \{w_0, w_1, \dots, w_k\}$, $k \in \mathbb{N}$, is an alphabet; in our setting we use the English vocabulary. $n \in \mathbb{N}^+$ is the order of the model to be learned, i.e., we will learn *n-gram VMM models*, and $\mathcal{H} = \{\mathcal{T} : \mathcal{T}$ is an *n-gram VMM model*$\}$ is the hypothesis space.

$L : \mathbf{T}_S \to \mathcal{H}$ is the learning algorithm. Let $\mathcal{L} \subset \mathbb{V}^*$ be the language of subject S and let p the probability distribution underlying $\mathcal{L}$. The aim of L is to map the training set $\mathbf{T}_S$ to the *n-gram VMM model* $\tilde{\mathcal{T}}_S \in \mathcal{H}$ approximating p best. Often, $\mathbf{T}_S$ consists only of a single document d. In that case, we denote the outcome of L equally as $\tilde{\mathcal{T}}_d$. We present two algorithms for learning $\tilde{\mathcal{T}}_S$, the *CPR-principle* and the *LLR-principle*. Both algorithms define different success criteria.

Next, we define a *discriminative learner* for a *multiclass model*. In terms of natural language processing *discriminative learning* consists of solving the *text classification* task, i.e., to find that *multiclass model* having most discriminative power among $\mathbf{C}$, where $\mathbf{C}$ is a set of classes.

In particular, we use a collection of *n-gram VMM models*, one for each class in $\mathbf{C}$, as *multiclass model*. *Bayes' classification rule* is used for classification (cf. Mitchell [15]). The learner is called *multiclass learner*. We define it as a 6-tuple $(\mathbf{C}, \mathbf{T}_\mathbf{C}, \mathbb{V}, n, \mathcal{H}, L)$. Here $\mathbf{C} = \{c_0, c_1, \dots, c_r\}$, $r \in \mathbb{N}$ is a set of classes or categories. Furthermore, $\mathbf{T}_\mathbf{C} = \{\mathbf{T}_{c_0}, \mathbf{T}_{c_1}, \dots, \mathbf{T}_{c_r}, \}$ is the training set for $\mathbf{C}$ and consists of a training set for each class in $\mathbf{C}$, where $\mathbf{T}_{c_i} = \{d_0, d_1, \dots, d_{q_i}\}$, $q_i \in \mathbb{N}$, is a training set for class c_i, $c_i \in \mathbf{C}$.

$\mathbb{V}$, n, and $\mathcal{H}$ are defined in the same way as for the *language learner*.

$L : \mathbf{T}_\mathbf{C} \to \mathcal{H}^r$ is the learning algorithm. The result of L is an *n-gram VMM model* for each class in $\mathbf{C}$. That is, L maps $\mathbf{T}_\mathbf{C}$ to $(\tilde{\mathcal{T}}_{c_0}, \tilde{\mathcal{T}}_{c_1}, \dots, \tilde{\mathcal{T}}_{c_r})$, $\tilde{\mathcal{T}}_{c_i} \in \mathcal{H}$,

$0 \leq i \leq r$, where $\tilde{\mathcal{T}}_{c_i}$ is the model for class c_i. We refer to $\tilde{\mathcal{T}}_{c_i}$ as *discriminative* class model or just as class model.

Both, the *language learner* and the *multiclass learner*, learn by statistical inference. Here we have to approximate the probability distribution of a language.

Let $\mathcal{L} \subset \mathbb{V}^*$ be a language and let p be the probability distribution underlying $\mathcal{L}$. Let $\mathbf{T} = \{d_0, d_1, \ldots, d_q\}$, $q \in \mathbb{N}$, be a sample or training set for $\mathcal{L}$. Therefore, each document $d_i \in \mathbf{T}$, $0 \leq i \leq q$, consists of samples taken from $\mathcal{L}$. Formally, a document $d_i = (t_{ij})_{j=0}^{o}$, $o \in \mathbb{N}$, is a list of sentences, where each sentence is in $\mathcal{L}$, i.e., $t_{ij} \in \mathcal{L}$ for every $i \in \{0, 1, \ldots, q\}$ and for every $j \in \{0, 1, \ldots, o\}$.

Furthermore, we assume that the sentences are mutually independent. The goal is to learn p from the samples in $\mathbf{T}$. Without prior knowledge the relative frequencies in $\mathbf{T}$, i.e., the empirical probability distribution, are most likely to equal p. We gain the empirical probability $\tilde{P}(s)$ for an *n-gram* $s \in \mathbb{V}^n$ by computing the ratio between the number of occurrences of s in $\mathbf{T}$ and the total number of *n-grams* in $\mathbf{T}$. The number of occurrences of s in training set $\mathbf{T}$ is denoted by $\#_{\mathbf{T}}(s)$, where

$$\begin{aligned}
\#_{\mathbf{T}}(s) &:= \sum_{i=0}^{q} \#_{d_i}(s), \qquad \mathbf{T} = \{d_0, d_1, \ldots, d_q\}, \\
\#_{d_i}(s) &:= \sum_{j=0}^{o} \#_{t_{ij}}(s), \qquad d_i = (t_{ij})_{j=0}^{o}, \\
\#_{t_{ij}}(s) &:= \text{the number of occurrences of } s \text{ in } t_{ij},\ t_{ij} \in \mathbb{V}^*.
\end{aligned} \tag{8}$$

The number of all *n-grams* is denoted by $N_{\mathbf{T}} = \sum_{s \in \mathbb{V}^n} \#_{\mathbf{T}}(s)$.
With the help of the counts we can calculate the empirical probabilities for all *l-grams*, $1 \leq l \leq n$. Let $\sigma_i \in \mathbb{V}$, $1 \leq i \leq n$, the empirical probabilities are calculated as

$$\tilde{P}(\sigma_i \sigma_{i+1} \ldots \sigma_n) = \begin{cases} \frac{\#_{\mathbf{T}}(\sigma_1 \sigma_2 \ldots \sigma_n)}{N_{\mathbf{T}}}, & \text{if } i = 1, \\ \sum\limits_{\sigma \in \mathbb{V}} \tilde{P}(\sigma \sigma_i \sigma_{i+1} \ldots \sigma_n), & \text{if } 1 < i \leq n, \end{cases} \tag{9}$$

where $\tilde{P}(\epsilon) := 1$.

Unfortunately, the empirical probability distribution is not a good estimator for the distribution of natural language, because of the sparse data problem. To overcome the problem we smooth the empirical probabilities, i.e., we assign a probability greater than zero to every *l-gram*, $1 \leq l \leq n$. We use a common smoothing technique known as *Lidstone's law*, see [13]. The smoothed empirical probability $P_{Lid}(s)$ for an *l-gram* s, $1 \leq l \leq n$, is calculated as

$$P_{Lid}(s) := \mu \tilde{P}(s) + (1 - \mu) \frac{1}{|\mathbb{V}|^l}, \qquad \mu = \frac{N_{\mathbf{T}}}{N_{\mathbf{T}} + \lambda |\mathbb{V}|}, \tag{10}$$

where $P_{Lid}(\epsilon) := 1$. The parameter $\lambda \in (0, \infty)$ is a constant, which is most often set to 0.5. Obviously, $\tilde{P}(s)$ equals $P_{Lid}(s)$ for $n \to \infty$.

For learning an *n-gram VMM model* we need to estimate conditional probabilities. We derive the estimate for the conditional probability $\sigma \in \mathbb{V}$ given

$s \in \mathbb{V}^{\leq n}$ in the following way. Let $\sigma_i \in \mathbb{V}$, $1 \leq i \leq n+1$. The conditional empirical probabilities of σ_{n+1} given the previous up to n words is computed as

$$P_{Lid}(\sigma_{n+1}|\sigma_i\sigma_{i+1}\dots\sigma_n) = \frac{P_{Lid}(\sigma_i\sigma_{i+1}\dots\sigma_n\sigma_{n+1})}{\sum\limits_{\sigma\in\mathbb{V}} P_{Lid}(\sigma_i\sigma_{i+1}\dots\sigma_n\sigma)}, \tag{11}$$

where $P_{Lid}(\sigma_{n+1}|\epsilon) = P_{Lid}(\sigma_{n+1})$.

Now, we are ready to present the learning algorithms. We use two already known algorithms and a new one. The existing algorithms are the *CPR-principle* and the *MMI-principle*. The *CPR-principle* has been introduced for learning the probability distribution of a language and the *MMI-principle* has been introduced for learning class models.

The new one is the *LLR-principle*. There are two instances of the *LLR-principle*, one for learning the probability distribution of a language and one for learning class models.

3.1 The CPR-Principle

The *CPR-principle* was introduced by Ron *et al.* [18] for learning a probabilistic language model. The learner consists of $(\mathbf{T}_S, \mathbb{V}, n, \mathcal{H}, L_{CPR})$. The goal is to approximate p, the probability distribution of $\mathcal{L} \subset \mathbb{V}^*$, where $\mathcal{L}$ is the language used by subject S.

Ron *et al.* [18] make the simplifying assumption that p can be described by an *n-gram VMM model* $\mathcal{T}_S \in \mathcal{H}$. Henceforth, let $\tilde{\mathcal{T}}_S \in \mathcal{H}$ denote the *n-gram VMM model* learned by L_{CPR} from training set $\mathbf{T}_S$. L_{CPR} aims to minimize the divergence between $\mathcal{T}_S$ and $\tilde{\mathcal{T}}_S$, where the divergence is measured in terms of the *Kullback-Leibler (KL) divergence* defined as

$$D(p\|q) = \sum_{x\in\mathcal{X}} p(x) \ \log\frac{p(x)}{q(x)}, \text{ where } p \text{ and } q \tag{12}$$

are two probability distributions defined over the finite, nonempty set $\mathcal{X}$.

Ron *et al.* [18] proved that the *KL divergence* between $\mathcal{T}_S$ and $\tilde{\mathcal{T}}_S$ converges to zero for sufficient large training sets.

The main idea of the algorithm introduced by [18] is to add a node $s \in \mathbb{V}^{\leq n}$ into $\tilde{\mathcal{T}}_S$, if $\tilde{P}(\sigma|s) > \tilde{P}(\sigma|\text{suff}(s))$ for any $\sigma \in \mathbb{V}$. More precisely, s is added if

$$\frac{\tilde{P}(\sigma|s)}{\tilde{P}(\sigma|\text{suff}(s))} \geq \varepsilon_2, \qquad s \in \mathbb{V}^{\leq n},\ \varepsilon_2 \in [1, \infty), \tag{13}$$

for any $\sigma \in \mathbb{V}$. Because of the equation, we termed the algorithm the *conditional probability ratio (CPR) principle*.

The *next symbol probabilities* for each node in $\tilde{\mathcal{T}}_S$ are set to the corresponding smoothed conditional empirical probabilities. If we would use the unsmoothed probabilities, the divergence between $\mathcal{T}_S$ and $\tilde{\mathcal{T}}_S$ would not converge to zero. This is caused by the fact that the *KL divergence* between $\mathcal{T}_S$ and $\tilde{\mathcal{T}}_S$ becomes infinite if $\mathcal{T}_S$ assigns a probability greater than zero to an *n-gram* s, while $\tilde{\mathcal{T}}_S$ assigns zero probability to s; see [18] for further details.

3.2 The MMI-Principle

The *MMI-principle* has been introduced by Slonim *et al.* [19]. In contrast to the *CPR-principle* the *MMI-principle* aims to learn class models. Therefore, the learner consists of $(\mathbf{C}, \mathbf{T_C}, \mathbb{V}, n, \mathcal{H}, L_{MMI})$.

The result of L_{MMI} is a *multiclass model* consisting of an *n-gram VMM model* for each class in $\mathbf{C}$. The class model for class $c \in \mathbf{C}$ is used to calculate the probability that a document belongs to c. Thus, the goal is to learn those class models minimizing the classification error rate. More precisely, the algorithm proposed by [19] aims to minimize *Bayes' error rate*. For information about *Bayes' classification rule* and *Bayes' error rate* see e.g., [15].

Let D be a random variable whose range is the set of all documents and let C be a random variable with range $\mathbf{C}$. From the definition of *Bayes' classification rule* it follows that *Bayes' error rate* decreases, if the uncertainty in $p(C|D)$ is reduced, for details see [7].

The uncertainty of the probability distribution underlying a random variable is measured in terms of the entropy, see e.g., [16]. Let X and Y be two random variables. Henceforth, we denote the entropy of X by $H(X)$ and the mutual information between X and Y by $I(X;Y)$, where $I(X;Y) := H(X) - H(X|Y)$. Thus, what we need is a way to minimize $H(C|D)$. We can model a document as a sequence of random variables $(S_i)_{i=1}^{o}$, $o \in \mathbb{N}^+$, where each S_i, $1 \leq i \leq o$, has range $\mathbb{V}^+$. We assume the random variables S_i, $1 \leq i \leq o$, to be equally distributed and mutually independent.

Let us assume that every sentence has exactly length m. Consequently, $S_i = \Sigma_1, \Sigma_2, \ldots, \Sigma_m$ for every $i \in \{1, 2, \ldots, o\}$, where Σ_j, $1 \leq j \leq m$, is a random variable with range $\mathbb{V}$.

If we assume that all documents have exactly o sentences, we derive the following equation, see [16].

$$\begin{aligned} H(D) &= H((S_i)_{i=1}^{o}) \\ &= o\, H(\Sigma_1, \Sigma_2, \ldots, \Sigma_m) \\ &= mo\, H(\Sigma_{n+1}|\Sigma_1, \Sigma_2, \ldots, \Sigma_n), \end{aligned} \tag{14}$$

where the last step follows from the fact that $\Sigma_1, \Sigma_2, \ldots, \Sigma_m$ is distributed according to an *n-gram VMM model* which in turn fulfills the *Markov assumption*. We simplify the notation by introducing two extra random variables, $\Sigma := \Sigma_{n+1}$ and $S := (\Sigma_1, \Sigma_2, \ldots, \Sigma_n)$. Now, we obtain our main result.

$$\begin{aligned} H(C|D) &= H(D|C) + H(C) - H(D) \\ &= H(C) - mo\,(H(\Sigma|S) - mo\; H(\Sigma|S, C)) \\ &= H(C) - mo\, I(\Sigma; C|S), \end{aligned} \tag{15}$$

where the equality of $H(D)$ and $mo\, H(\Sigma|S)$ as well as the equality of $H(D|C)$ and $mo\, H(\Sigma|S, C)$ follows from (14).

We conclude, maximizing $I(\Sigma; C|S)$ minimizes $H(C|D)$ and therefore minimizes *Bayes' error rate*, too.

We call the resulting algorithm the *maximum mutual information (MMI) principle*, as it tries to maximize $I(\Sigma; C|S)$. Further details are omitted due to the lack of space.

3.3 The LLR-Principle

The *LLR-principle* is a new way either to learn an *n-gram VMM model* for a subject or to learn *n-gram VMM models* serving as class models. The main motivation of the *LLR-principle* is a common drawback of the *CPR-* and the *MMI-principle*. Both make the assumption that the training set is always sufficiently large in order to get reliable estimates for the unknown probability distribution. Let p be an unknown probability distribution and let us assume p is described by the *n-gram VMM model* $\mathcal{T}$. Furthermore, $|\mathcal{T}|$ denotes the number of nodes in $\mathcal{T}$. The size of the needed sample set for achieving reliable results for p is bounded by a polynomial function over $|\mathcal{T}|$, for details see [18]. In our setting we have no knowledge about $|\mathcal{T}|$, besides the fact that $|\mathcal{T}|$ is upper-bounded by $|\mathbb{V}|^n$, but in general we do not possess a sample set larger than $|\mathbb{V}|^n$. Hence, we normally cannot determine, whether or not the training set is sufficiently large.

So, what can happen if the training set is not large enough? Dependencies might be preserved, which are true for the training set, but not for p, i.e., the learned *n-gram VMM model* overfits. Overfitting in turn can reduce classification and retrieval performance, see e.g., [15] and [12].

Thus, the idea of the *LLR-principle* is to add a node s only, if there is strong evidence that s is an indispensable node in $\mathcal{T}$. In particular, we use a statistical test to determine whether or not s is indispensable.

First, we use the idea to learn the probability distribution of the language of a subject S. Let $(\mathbf{T}_S, \mathbb{V}, n, \mathcal{H}, L_{LLR})$ be the learner and let $L_{LLR}(\mathbf{T}_S) = \tilde{\mathcal{T}}_S$. Furthermore, let p be the probability distribution of the language used by subject S. Thus, our goal is to approximate p.

Formally, we want to add a node $s \in \mathbb{V}^{\leq n}$ into $\tilde{\mathcal{T}}_S$, if $p(\sigma|s) > p(\sigma)$ for any $\sigma \in \mathbb{V}$. In other words, we add s into $\tilde{\mathcal{T}}_S$, if the probabilities of some of the words following s are not independent of s.

Since we do not know $p(\sigma|s)$, we use the conditional empirical probability achieved from $\mathbf{T}_S$ as estimator for $p(\sigma|s)$. Furthermore, we use a statistical test to decide whether or not a dependency occurring in $\mathbf{T}_S$ exists in p. Thus, we use the following two hypotheses for the test:

$$\begin{aligned} &H_0\colon p(\sigma|s) = p = p(\sigma) \\ &H_1\colon p(\sigma|s) = p_1 > p_2 = p(\sigma) \end{aligned} \tag{16}$$

If the null hypotheses H_0 is rejected for any $\sigma \in \mathbb{V}$, we add s to $\tilde{\mathcal{T}}_S$.

As statistical test we use the *log-likelihood ratio (LLR) test* proposed in [3].

Dunning [3] applied the *LLR-test* on the task of finding dependencies between words in texts written in natural language and compared it to *Pearson's χ^2-test*. Especially for small word counts, i.e., for short samples, the *LLR-test* performed superior to *Pearson's χ^2-test*.

The *LLR-test* seems to be the statistical test fitting best for finding dependencies between words when dealing with small word counts (cf. [13, 20]).

[3] used $p(\sigma|s) = p(\sigma|\neg s)$ as null hypothesis, where $p(\neg s) := 1 - p(s)$. However, Dunning's null hypothesis is equal to H_0:

$$\begin{aligned}
& p(\sigma|s) = p(\sigma|\neg s) \\
\Longleftrightarrow \quad & p(\sigma|s) = \tfrac{p(\neg s\ \sigma)}{p(\neg s)} \\
\Longleftrightarrow \quad & p(\sigma|s) = \tfrac{\sum_{s' \in \mathbb{V}^{|s|}} p(s'\ \sigma) - p(s\ \sigma)}{1-p(s)} \\
\Longleftrightarrow \quad & (1 - p(s))p(\sigma|s) = p(\sigma) - p(s\ \sigma) \\
\Longleftrightarrow \quad & p(\sigma|s) = p(\sigma)
\end{aligned}$$

Computing the test statistic proposed in [3] for H_0, is done using the counts:

$$k_1 := \#_{\mathbf{T}_S}(s\sigma), \quad n_1 := \#_{\mathbf{T}_S}(s), \quad k_2 := \#_{\mathbf{T}_S}(\sigma) - k_1, \quad n_2 := N_{\mathbf{T}_S} - n_2$$

As estimates for the tested probabilities we simply use the empirical probabilities. In terms of the above counts the estimates can be expressed as

$$\tilde{p}_1 = \frac{k_1}{n_1}, \quad \tilde{p}_2 = \frac{k_2}{n_2}, \quad \tilde{p} = \frac{k_1 + k_2}{n_1 + n_2}.$$

Now, the test statistic for hypothesis H_0 is calculated as

$$\begin{aligned}
\xi = -2\, log\, \lambda = 2\,(log\, L(\tilde{p}_1, k_1, n_1) \ + \ & log\, L(\tilde{p}_2, k_2, n_2) \\
& - \ log\, L(\tilde{p}, k_1, n_1) - \ log\, L(\tilde{p}, k_2, n_2)), \quad (17)
\end{aligned}$$

where $L(p, k, n) := p^k(1 - p)^{(n-k)}$. For further details about the *LLR-test* and the derivation of the *LLR-test* statistic, see [3].

The value λ is the ratio between the likelihood of H_0 and the likelihood of H_1. So, λ decreases and thus ξ increases, if H_1 is more likely than the null hypothesis H_0. Moreover, ξ approximates the χ^2-distribution with one degree of freedom.

Putting it all together, we add $s \in \mathbb{V}^{\leq n}$ to $\tilde{\mathcal{T}}_S$, if $\xi_{\sigma|s} \geq (\chi^2_1)^{-1}(1 - \alpha)$, $\alpha \in (0, 1)$, for any $\sigma \in \mathbb{V}$. $\xi_{\sigma|s}$ denotes the value of the *LLR-test* statistic computed for σ and s. The complete algorithm is given in Figure 1.

In the algorithm, the parameter ε is the critical value for the χ^2_1-distribution for a chosen significance level $\alpha \in (0, 1)$, i.e., $\varepsilon = (\chi^2_1)^{-1}(1 - \alpha)$.

The advantage of the *LLR-principle* is that a node is only added, if it is with high probability an indispensable node in an *n-gram VMM model* describing p. Unfortunately, some nodes may not be added even when they are necessary for modeling a dependency in p. So, the question is if the learned *n-gram VMM model* still fits for solving the *text retrieval* and *text classification* task, respectively.

In the worst case the learned *n-gram VMM model* $\tilde{\mathcal{T}}_S$ consists only of the root node, i.e., words are regarded as to be independent to one another. Hence, $\tilde{\mathcal{T}}_S$ can be described by a 0-*gram VMM model*. *Bayes' classification rule* applied to the 0-*gram VMM model* yields the *naive Bayes classifier*, which shows an excellent performance for the *text classification* task, see e.g., [14, 15].

First step (prediction suffix tree)

```
// initialize the prediction suffix tree $\tilde{\mathcal{T}}_S$
$\tilde{\mathcal{T}}_S$ := $\{\epsilon\}$;
// build the prediction suffix tree $\tilde{\mathcal{T}}_S$
for ($i$ = 1 to $n$) {
    foreach ($s \in \mathbb{V}^i$) {
        $\xi_{max}$ := 0;
        foreach ($\sigma \in \mathbb{V}$) {
            // test for a dependency in the language of $S$
            $k_1$ := $\#_{\mathbf{T}_S}(s\sigma)$; $n_1$ := $\#_{\mathbf{T}_S}(s)$;
            $k_2$ := $\#_{\mathbf{T}_S}(\sigma) - k_1$; $n_2$ := $N_{\mathbf{T}_S} - n_1$;
            $\tilde{p}$ := $(k_1 + k_2)/(n_1 + n_2)$; $\tilde{p}_1$ := $k_1/n_1$; $\tilde{p}_2$ := $k_2/n_2$;
            $\xi$ := llr_test($\tilde{p}$, $\tilde{p}_1$, $\tilde{p}_2$; $k_1$, $n_1$, $k_2$, $n_2$);
            // find the maximal value for $\xi$
            $\xi_{max}$ := $\max\{\xi,\ \xi_{max}\}$;
        }
        // add only those nodes, which pass the test
        if ($\xi_{max} \geq \varepsilon$) {
            add $s$ into $\tilde{\mathcal{T}}_S$;
        }
    }
}
```

Second step (next symbol probabilities)

```
// compute the next symbol probabilities
foreach ($s \in \tilde{\mathcal{T}}_S$) and ($\sigma \in \mathbb{V}$) {
    $\tilde{\gamma}_s(\sigma)$ := $P_{Lid}(\sigma|s)$;
}
```

Fig. 1. Pseudo-code for the LLR-principle (subject case).

In fact, other classifiers and text retrieval systems assuming independence between words got good results in *text classification* and *text retrieval*, see [10, 12, 13]. So, we expect the *n-gram VMM model* $\tilde{\mathcal{T}}_S$ learned by the *LLR-principle* to yield good results, even if some dependencies in p are not modeled by $\tilde{\mathcal{T}}_S$.

Next, we apply the *LLR-test* to the task of learning class models. The learner consists of $(\mathbf{C}, \mathbf{T_C}, \mathbb{V}, n, \mathcal{H}, L_{MMI})$. Similar to the *MMI-principle* we aim to find the suffix tree $\tilde{\mathcal{T}}$ containing all the nodes, which have discriminative power among $\mathbf{C}$. Thus, we add a node $s \in \mathbb{V}^{\leq n}$ to $\tilde{\mathcal{T}}$, if s fulfills the following two requirements.

First, for some class $c \in \mathbf{C}$ there exist a dependency between s and the words following s. This requirement yields the following hypotheses for a statistical test:

$$\begin{array}{l} H_0^1\colon p(\sigma|s,c) = p = p(\sigma|c) \\ H_1^1\colon p(\sigma|s,c) = p_1 > p_2 = p(\sigma|c) \end{array} \tag{18}$$

Therefore, the first requirement is fulfilled, if the null hypotheses H_0^1 is rejected for any $\sigma \in \mathbb{V}$ and $c \in \mathbf{C}$.

For testing the null hypotheses we apply the *LLR-test* on the training set $\mathbf{T_C}$. Concretely speaking, for testing H_0^1 for $s \in \mathbb{V}^{\leq n}$, $\sigma \in \mathbb{V}$, and $c \in \mathbf{C}$ the training set $\mathbf{T}_c$ is used. From $\mathbf{T}_c$ we get the following counts needed for calculating the *LLR-test* statistic:

$$k_1 := \#_{\mathbf{T}_c}(s\sigma), \quad n_1 := \#_{\mathbf{T}_c}(s), \quad k_2 := \#_{\mathbf{T}_c}(\sigma) - k_1, \quad n_2 := N_{\mathbf{T}_c} - n_2$$

The value of the *LLR-test* statistic is computed by (17).

The second requirement is that a node s has discriminative power among $\mathbf{C}$. That is, a dependency between s and the words following s depends on $\mathbf{C}$, too. In other words, if $p(\sigma|s,c)$ is equal for every $c \in \mathbf{C}$ and for every $\sigma \in \mathbb{V}$, s has no discriminative power among $\mathbf{C}$.

We formalize the second requirement by the following hypotheses for a statistical test:

$$\begin{array}{l} H_0^2\colon p(\sigma|s,c) = p = p(\sigma|s) \\ H_1^2\colon p(\sigma|s,c) = p_1 > p_2 = p(\sigma|s) \end{array} \tag{19}$$

Thus, if H_0^2 is rejected for any $\sigma \in \mathbb{V}$ and $c \in \mathbf{C}$, then the second requirement is fulfilled.

For computing the *LLR-test* statistic for $s \in \mathbb{V}^{\leq n}$, $\sigma \in \mathbb{V}$, and $c \in \mathbf{C}$ we need counts over the training set $\mathbf{T}_c$ and over $\mathbf{T_C}$. The number of occurrences of an *n-gram* $s \in \mathbb{V}^n$ in the training set $\mathbf{T_C}$ is defined as

$$\#_{\mathbf{T_C}}(s) := \sum_{i=0}^{r} \#_{\mathbf{T}_{c_i}}(s), \quad \mathbf{T_C} = \{\mathbf{T}_{c_0}, \mathbf{T}_{c_1}, \ldots, \mathbf{T}_{c_r}\}, \tag{20}$$

and hence $N_{\mathbf{T_C}} := \sum_{i=0}^{r} N_{\mathbf{T}_{c_i}}$. Consequently, the *LLR-test* is performed on:

$$k_1 := \#_{\mathbf{T}_c}(s\sigma), \quad n_1 := \#_{\mathbf{T}_c}(s), \quad k_2 := \#_{\mathbf{T_C}}(\sigma) - k_1, \quad n_2 := N_{\mathbf{T_C}} - n_2$$

If a node $s \in \mathbb{V}^{\leq n}$ fulfills both requirements, then s is added to $\tilde{\mathcal{T}}$. We omit further details.

4 Text Classification and Text Retrieval

The *text classification* task consists of a set of classes $\mathbf{C} = \{c_0, c_1, \ldots, c_r\}$, $r \in \mathbb{N}$, a set of documents $\mathbf{D}$, and a function $k : \mathbf{D} \to \{0, 1, \ldots, r\}$ called classification rule. $\mathbf{D}$ is the set of all documents d such that each $d \in \mathbf{D}$ belongs to exactly one class in $\mathbf{C}$. The aim of the *text classification* task is to find that classification rule that approximates k_{true} best, where k_{true} returns the correct class label for each document in $\mathbf{D}$, i.e., if $d \in \mathbf{D}$ belongs to $c_i \in \mathbf{C}$, then $k_{true}(d) = i$.

Function k only relies on the knowledge about $\mathbf{C}$ and hence, if the knowledge about $\mathbf{C}$ is not sufficient for certain classifications, k is expected to make some misclassifications. The classification rule theoretically making the least number of misclassifications is called *Bayes' classification rule* and it is denoted by k_{Bayes}. The ratio of misclassifications made on average by *Bayes' classification rule* is called *Bayes' error rate*. For further details see e.g., [15].

Let $(\mathbf{C}, \mathbf{T_C}, \mathbb{V}, n, \mathcal{H}, L)$ be an *n-gram VMM model* learner for class models, where $L(\mathbf{T_C}) = (\tilde{\mathcal{T}}_{c_i})_{i=0}^r$, $\tilde{\mathcal{T}}_{c_i} \in \mathcal{H}$, $c_i \in \mathbf{C}$, $0 \leq i \leq r$. Furthermore, let $\tilde{k}_{Bayes}$ be *Bayes' classification rule* for $\mathbf{C}$, where $\tilde{k}_{Bayes}$ uses $\tilde{\mathcal{T}}_{c_i}$ to estimate the value of $p(d|c_i)$ for every $c_i \in \mathbf{C}$, $0 \leq i \leq r$, and for every $d \in \mathbf{D}$. We call the triple $(\mathbf{C}, (\tilde{\mathcal{T}}_{c_i})_{i=0}^r, \tilde{k}_{Bayes})$ nst order *naive Bayes classifier*, where the 0st order *naive Bayes classifier* is a member of the family of the common *naive Bayes classifiers*.

Following McCallum *et al.* [14] we also look at the commonly used *multivariate naive Bayes classifier* and the *multinomial naive Bayes classifier*. Both classifiers use the vector space model of $\mathbb{V} = \{w_0, w_1, \ldots, w_k\}$, $k \in \mathbb{N}$, to represent each document in $\mathbf{D}$. In the multinomial model the likelihood of $d \in \mathbf{D}$, $d = (t_l)_{l=0}^o$, $t_l \in \mathbb{V}^+$, $0 \leq l \leq o$, given class $c \in \mathbf{C}$ is

$$p(d|c) = N_d! \prod_{i=0}^{k} \frac{p(w_i|c)^{v_i}}{v_i!}, \qquad N_d = \sum_{i=0}^{k} \#_d(w_i), \tag{21}$$

which is just the multinomial distribution. Let $\mathbf{T}_c$ be a training set for c. The parameter $p(w_i|c)$, $0 \leq i \leq k$, of the multinomial distribution are estimated by the smoothed conditional empirical probabilities achieved from $\mathbf{T}_c$.

Additionally, we shall use two classification rules relying on the vector space model over $\mathbb{V}$ for comparative reasons, the *k-nearest-neighbor (k-nn)-classifier* and the *centroid based classifier*(cf., e.g., [8] and [13]). The most common similarity measure used in *text classification* and *text retrieval* is the cosine of the angel between two vectors v and u in $\mathcal{V}$. Therefore, we also use it. For further details on the cosine function as similarity measure see e.g., [8, 13].

A drawback of the vector space model over $\mathbb{V}$ is its limited representation power, as it only regards single words, i.e., 1-*grams*. There exist two commonly used ways for improving the representation power: either to use the vector space over all *n-grams* for an $n \in \mathbb{N}$, $n > 1$, or the vector space over all *l-grams*, $1 \leq l \leq n$. Both kind of vector spaces have been tested in practical experiments, see e.g., [12, 6]. The results indicate that classification performance increases for small n, but decreases for larger n; the reason is overfitting caused by the high dimensionality of the vector space. The way to overcome the problem of high dimensionality is to build the vector space only over those *l-grams*, $1 \leq l \leq n$, having relevance for classification. Many different approaches for measuring the relevance of a single *l-gram*, $1 \leq l \leq n$, have been proposed, see e.g., [6, 20].

Here, we examine two approaches, one based on mutual information, which we derive from the *MMI-learning-principle*. The other one is based on statistical tests and it is derived from the *LLR-learning-principle*.

The measures used for evaluating classification results are micro-averaged precision/recall.

Last but not least, we use a probabilistic text retrieval system. Recall that a query $\mathbf{Q}$ as a set of documents. For every query $\mathbf{Q}$ we split $\mathbf{D}$ into two sets, $\mathbf{R}$ (the relevant documents) and $\mathbf{N}$ (the irrelevant ones). A *text retrieval* system is an algorithm that assigns a rank $r \in \{1, 2, \ldots, |\mathbf{D}|\}$ to each document in $\mathbf{D}$ such that no two documents get the same rank, where 1 is referred to as the highest rank. Informally speaking, an *optimal text retrieval* system makes at any time and for every query the least number of wrong assignments of ranks. A wrong assignment has happened, if a document in $\mathbf{N}$ gets a higher rank than any document in $\mathbf{R}$. $\mathbf{D}_n \subset \mathbf{D}$ is the set of the n documents having the highest ranks, i.e., the ranks 1 to n, and $\mathbf{R}_n = \mathbf{D}_n \cap \mathbf{R}$. A probabilistic *text retrieval* system is called *optimal*, if

$$E[|\mathbf{R}_n|] = \sum_{d \in \mathbf{D}_n} P(\mathbf{R}|d) \text{ is maximal for all } n \in \{1, 2, \ldots, |\mathbf{D}|\}, \qquad (22)$$

where $P(\mathbf{R}|d)$ is called *probability of relevance*. Robertson [17] has shown that such an *optimal probabilistic text retrieval* system exists and can be derived by the *probability ranking principle*(*PRP*). Thus, the performance of the system relies on the reliability of our estimates for $P(\mathbf{R}|d)$.

In Section 2 we already presented two observations about d, $\mathbf{Q}$, $\mathbf{R}$, and hence about $P(\mathbf{R}|d)$. First, d is probably in $\mathbf{R}$, if d and the documents in $\mathbf{Q}$ are about the same subject. Second, if d and the documents in $\mathbf{Q}$ belong to the same class, we expect d to be in $\mathbf{R}$, too. Both observation can be quantified and the results are used to get an estimate for $P(\mathbf{R}|d)$. We omit details.

5 Implementation

Next, we outline the workflow of our search engine and take a short look at the core algorithms used within the single steps of the workflow. Figure 2 shows the workflow of the search engine and the single steps making up the workflow. Every document processed by the search engine passes these steps with exception of documents used for learning a class model, which pass only the first six steps.

We distinguish three levels of representation of a document. First, the document is a collection of bits. In this step we extract the text of a document. Then it has to be split into words and sentences for further processing.

On the multi-word level the computer has a more advanced view of the document. Instead of single words, it regards *l-grams*, $1 \leq l \leq n$, i.e., words are regarded within their local context. In order to be able to receive *l-grams* we need the sentences boundaries as we do not want to regard *l-grams* across multiple sentences.

The next level is the *n-gram VMM model* itself. Actually, our hope is to gain a representation capturing the meaning of the document. However, we do not aim to represent the semantic of a document in an operational form. That is, we cannot answer the question “What subject do you deal with?”, but our hope is that we are able to answer the question “Do you deal with the same subject as document x?”.

Next, we take a short look at the steps presented in Figure 2. The first task is text extraction. In this step, we extract the parts of natural language within a document. For documents published in HTML format, we use a simple HTML-filter. The filter just removes all HTML-tags and all the text between HTML-tags, which are known to mark structured text.

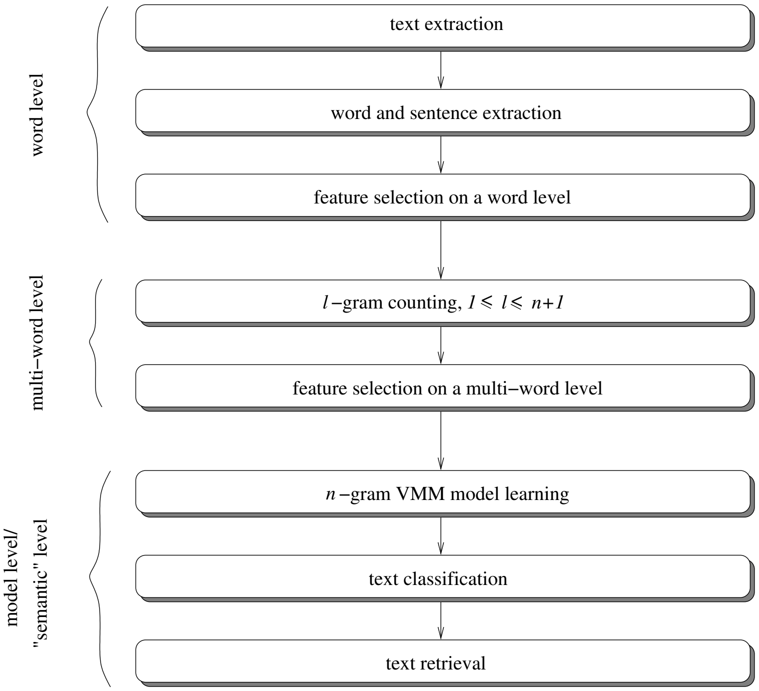
word level
text extraction
word and sentence extraction
feature selection on a word level
multi–word level
l–gram counting, $1 \leqslant l \leqslant n+1$
feature selection on a multi–word level
model level/ "semantic" level
n–gram VMM model learning
text classification
text retrieval

Fig. 2. Workflow of the search engine.

Scientific papers published in the Internet are normally encoded as postscript-files or as PDF-files. The postscript- and the PDF-format are strongly related and were invented to provide a good, platform independent readability of natural and structured text for an human reader. The result is a format, which makes automatic text extraction quite complicated.

Here, we use a two-step-process for *text extraction*. First, the `ps2ascii` tool delivered with the `ghostscript` package[4] is used to interpret the postscript-document and to get a list of word fragments. The second step is done by a self-written filter, which combines the fragments into words, finds paragraphs of natural text, and removes words belonging to structures like formulas, etc.

In the next step, the extracted text is converted into unicode. Then we cut the text into words and sentences. In English words are normally separated by spaces. Hence, we regard every sequence of characters flanked by spaces as a word. Finding sentences is not trivial mainly due to the fact that a period does not necessarily mark the end of a sentence. We use an algorithm based on the sentence boundary detection algorithm proposed in [13]. The algorithm uses an heuristic approach, is fast and on common texts very reliable. After finding sentence boundaries, all punctuation marks are removed.

[4] http://www.ghostscript.com

Up to this point we used the English dictionary as vocabulary. Note that the large size large size of the vocabulary and the resulting huge number of *l-grams*, $1 \leq l \leq n$, can cause serious problems with respect to time and memory requirements. In order to overcome the problem there exist a variety of techniques to select those *l-grams*, $1 \leq l \leq n$, being somehow relevant for learning the desired *n-gram VMM models* and to discard all irrelevant ones. The general task is called *feature selection* and is done as a pre-processing step before inferring any models, see e.g., [13, 10]. In the workflow *feature selection* occurs on two levels, on the word and on the multi-word level. At both levels, we use *stopword elimination* and *word frequency thresholding* among others.

Then, we count all *l-grams*, $1 \leq l \leq n$, in a sample set. After counting, *l-gram frequency thresholding* is applied at the result list in order to remove all *l-grams* occurring less than two times. The remaining frequencies are used to compute the normal and the smoothed conditional empirical probabilities. The probabilities and the counts are required for learning the *n-gram VMM model.*

Counting is probably the most important step in the workflow of the search engine. It is most expensive in time and the structure build up over all *l-grams* during the counting step is later on used for a fast find, calculation, and comparison of *l-gram* frequencies and probabilities. Word counting is done by applying a hash function which uses the single words as key and stores the counts in a hash map. The resulting algorithm runs in time $O(m)$ and uses $O(m)$ memory, where m is the number of words in the training set.

We can easily extend the approach for counting all *l-grams*, $1 \leq l \leq n$. That is, we count the number of occurrences of each *l-gram*, $1 \leq l \leq n$, separately. The resulting algorithm has a runtime of $O(mn^2)$ and needs $O(mn^2)$ memory, but we can do better. We omit the details here.

The data structure developed for counting fits in general for a *prediction suffix tree*(PST), too. With this in mind the objects stored in the array of hash maps will serve as nodes and *next symbol probabilities* at the same time. This design allows us to store a PST in a very compact manner. For learning, we use the principles described in Section 3.

6 Experimental Results

We have used the Reuters-21578 data set for testing[5]. The data set consists of 21578 Reuters newswire stories from the year 1987 all related to financial topics. In order to support retrieval and routing, Reuters defined a set of 135 category labels. The stories have been manually indexed using these labels, where each story may be indexed by zero, one, or several category labels.

Lewis was the first making extensive use of the Reuters data set for evaluating *text classification* systems, e.g., see [11]. Henceforth, the data set has become a standard benchmark for text classifiers, e.g., see [19, 10].

In general, we use the newswire stories as documents and the categories as classes for our search engine. However, it is difficult to make a selection of

[5] http://www.daviddlewis.com/resources/testcollections/reuters21578/

documents for a training set and a test set, respectively, because many of the stories are only of limited use or completely unusable. The most common way to split the data set is the so called modified Apte split[6], which we will use, too. It defines a training set of 9603 documents and a test set of 3299 documents. The remaining 8676 documents are not used, because of one or several of the following reasons. They have no class label assigned, the assigned class label is obviously false, or the document contains no text. Furthermore, Reuters allowed *multi-classification*. Our *text classification* system is restricted to *single-classification*. Thus, we only use the documents belonging exactly to one class.

A single newswire story is often very short and consists only of one or a few sentences; a story is on average 152 words long. Therefore, we normally cannot learn an *n-gram VMM model* from a single or a few stories, because the limitedness of training data does not allow reliable statistical inference. Therefore, we use only those Reuters categories as classes, where the training set provides a sufficient large number of training documents. In particular, we only use the ten most populous categories yielding a minimum number of 90 documents per category. This approach follows several other studies like [10, 14, 19].

The ten categories are listed in the following table. Additionally, the number of documents and words per category are given. Word counting is done without prior *feature selection*, but every word was converted to lowercase.

category	topic	number of documents	number of words
earn	Earnings and Earnings Forecasts	2840	221992
acq	Mergers/Acquisitions	1596	192130
crude	Crude Oil	253	50376
trade	Trade	251	57693
money-fx	Money/Foreign Exchange	206	35014
interest	Interest Rates	190	22471
money-supply	Money Supply	123	12605
ship	Shipping	108	16488
sugar	Sugar	97	17012
coffee	Coffee	90	18757

So, we get a training set of 5754 documents and a test set of 2254 documents. Without any kind of *feature selection* the vocabulary of the training set consists of 37965 words and 2051584 different *l-grams*, $1 \leq l \leq 6$, where we will not use larger *l-grams* for testing.

The second consequence of the shortness of the newswire stories directly concerns our *text retrieval* system. A single document is too small to be used for learning an *n-gram VMM model* and hence we cannot use the *KL divergence* based similarity measure to predict the similarity between two documents. Therefore, we compute the vector space representation for each document and use the cosine function as similarity measure. The following example shows the

[6] http://www.daviddlewis.com/resources/testcollections/reuters21578/readme.txt

difficulties in using such short documents as contained in the Reuters data set. The example is taken from the used test set and belongs to "interest".

> **Bundesbank's Schlesinger says no plan to cut discount rate - Nihon Keizai newspaper**
> Blah blah blah.
> *(Reuters-21578, NEW_ID=17445)*

We present here only some text classification results due to lack of space. We compare the nst order *naive Bayes classifier* learned by the *MMI-principle* with the one learned by the *LLR-principle*. Additionally, for both principles we let the nst order *naive Bayes classifier* compete against the *centroid based* and the *k-nn-classifier*, where both use the vector space $\mathcal{V}^n_{MMI(LLR)}$.

We evaluate the general improvement gained by applying a learning algorithm. Thus, we run *Bayes' classification rule* without prior learning, i.e., we build a *prediction suffix tree* containing a node s for each $s\sigma$ occurring in the training set, $s \in \mathbb{V}^{\leq n}$, $\sigma \in \mathbb{V}$.

We use the following *feature selection* techniques on a word level. Every word is converted to lowercase, every number is replaced by the word <NUMBER>, word frequency thresholding is applied, and stopwords are eliminated. On a multi-word level *l-gram* frequency thresholding, $1 \leq l \leq n$, is applied. We used this set-up for every *text classification* and *text retrieval* task reported here.

	$n = 0$	1	2	3	4	5
no learning						
Bayes	0.95874	0.95209	0.9512	0.95164	0.95075	0.9512
MMI-principle						
Bayes	0.95874	0.95874	0.95918	0.9543	0.95386	0.9543
centroid	0.48048	0.93966	0.93478	0.9299	0.9339	0.93256
1-*nn*	0.92902	0.92902	0.93523	0.93833	0.93789	0.937
LLR-principle						
Bayes	0.95874	0.95608	0.95519	0.95608	0.95608	0.95563
centroid	0.48048	0.92414	0.90816	0.90595	0.90639	0.90639
1-*nn*	0.91926	0.92014	0.92103	0.9197	0.92014	0.92014

Table 1. Classification results. The values inside the table are micro-averaged *precision/recall*, where the column determines the used *n-gram VMM model* and the row determines the used combination of decision rule and learning principle. Here, "*Bayes*" is short for nst order *naive Bayes classifier*, "*centroid*" is short for *centroid based classifier*, and "*1-nn*" is short for *k-nn-classifier*, where k is set to 1.

Furthermore, we use the micro-averaged *precision/recall* to measure the classification performance of a single combination of classification rule, learning principle, and memory depth n. The micro-averaged *precision* and *recall* are equal because of the *single classification* setting. However, for a single class *precision* and *recall* are normally different.

Table 1 summarizes the results for all combinations of classification rule and learning principle. All combinations were evaluated for $n = 0$ to 5.

References

[1] J. L. Doob. *Stochastic Processes.* Wiley, 1990.

[2] L. Dümbgen. *Stochastik für Informatiker.* Springer, 2003.

[3] T. E. Dunning. Accurate methods for the statistics of surprise and coincidence. *Computational Linguistics*, 19(1):61–74, 1994.

[4] W. Feller. *An Introduction to Probability Theory and Its Applications*, volume 1. Wiley, third edition, 1968.

[5] N. Fuhr. Probabilistic models in information retrieval. *The Computer Journal*, 35(3):243–255, 1992.

[6] J. Fürnkranz. A study using n-gram features for text categorization. Technical report, Austrian Institute for Artificial Intelligence, 1998.

[7] A. Garg and D. Roth. Understanding probabilistic classifiers. In L. D. Raedt and P. A. Flach, editors, *Machine Learning: EMCL 2001, 12th European Conference on Machine Learning, Freiburg, Germany, September 5-7, 2001, Proceedings*, volume 2167 of *Lecture Notes in Computer Science*, pages 179–191. Springer, 2001.

[8] D. Hand, H. Mannila, and P. Smyth. *Principles of Data Mining.* MIT Press, 2002.

[9] M. A. Harrison. *Introduction to Formal Language Theory.* Addison Wesley, 1978.

[10] T. Joachims. *Learning to Classify Text using Support Vector Machines: Methods, Theory, and Algorithms.* Kluwer Academic Publishers, 2002.

[11] D. D. Lewis. Feature selection and feature extraction for text categorization. In *Proceedings of Speech and Natural Language Workshop*, pages 212–217, San Mateo, California, 1992. Morgan Kaufmann.

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

[13] C. D. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing.* MIT Press, 2002.

[14] A. McCallum and K. Nigam. A comparison of event models for naive bayes text classification. In *Proceedings of the AAAI-98 Workshop on Learning for Text Categorization*, 1998.

[15] T. M. Mitchell. *Machine Learning.* WCB/McGraw-Hill, 1997.

[16] A. Papoulis. *Probability, Random Variables, and Stochastic Processes.* WCB/McGraw-Hill, third edition, 1991.

[17] S. E. Robertson. The probability ranking principle in ir. *Journal of Documentation*, 33:294–304, 1977.

[18] D. Ron, Y. Singer, and N. Tishby. The power of amnesia: Learning probabilistic automata with variable memory length. *Machine Learning*, 25(2–3):117–149, 1996.

[19] N. Slonim, G. Bejerano, S. Fine, and N. Tishby. Discriminative feature selection via multiclass variable memory markov model. In C. Sammut and A. G. Hoffmann, editors, *Machine Learning, Proceedings of the Nineteenth International Conference (ICML 2002), University of New South Wales, Sydney, Australia, July 8-12, 2002*, pages 578–585. Morgan Kaufmann, 2002.

[20] Y. Yang. An evaluation of statistical approaches to text categorization. *Information Retrieval*, 1(1/2):69–90, 1999.