Text Classification Improved through Multigram Models

Dou Shen[†], Jian-Tao Sun[‡], Qiang Yang[†], Zheng Chen[‡]

[†]Department of Computer Science and Engineering,
Hong Kong University of Science and Technology
{dshen,qyang}@cse.ust.hk

[‡]Microsoft Research Asia, Beijing, P.R.China
{jtsun, zhengc}@microsoft.com

ABSTRACT

Classification algorithms and document representation approaches are two key elements for a successful document classification system. In the past, much work has been conducted to find better ways to represent documents. However, most of the attempts rely on certain extra resources such as WordNet, or they face the problem of extremely high dimension. In this paper, we propose a new document representation approach based on n-multigram language models. This approach can automatically discover the hidden semantic sequences in the documents under each category. Based on n-multigram language models and n-gram language models, we put forward two text classification algorithms. The experiments on RCV1 show that our proposed algorithm based on n-multigram models alone can achieve the similar or even better classification performance compared with the classifier based on n-gram models but the model size of our algorithm is much smaller than that of the latter. Another proposed algorithm based on the combination of n-multigram models and n-gram models improves the micro-F1 and macro-F1 values from 89.5% to 92.6% and 87.2% to 91.1% respectively. All these observations support the validity of our proposed document representation approach.

Categories and Subject Descriptors

I.5.2 [**Pattern Recognition**]: Design Methodology—*Classifier design and evaluation*; I.5.4 [**Pattern Recognition**]: Applications—*Text processing*

General Terms

Algorithms, Performance, Experimentation

Keywords

N-Gram models, N-Multigram models, Document Representation, Text Classification

CIKM'06, November 5–11, 2006, Arlington, Virginia, USA.

1. INTRODUCTION

With the rapid explosion of text in digital form, automatic text classification (TC) has been a hot research topic. As a specific type of pattern classification tasks, TC has two essential aspects. One is the classification algorithm and the other is document representation. During the past decades, a large number of categorization algorithms have been proposed for TC such as naïve bayes [16], k-nearest neighbor [28], support vector machines [9], boosting [24] and rule learning algorithms [25]. However, the performance of the classification systems also tightly depends on the document representation.

Document representation refers to the selection of appropriate features to represent documents. A bag-of-word representation scheme is widely used in text classification due to its simplicity and efficiency [23]. Under this scheme, documents are represented by bags of terms, each term being an independent feature of its own. A document can be represented as a vector. Each item in the vector corresponds to an individual term and its value can be defined as a binary indicator or the absolute frequency or more elaborated measures like TF*IDF [23]. Although bag-of-words has been applied in many Information Retrieval (IR) fields, it has many shortcomings. For example, it can not reflect the relationship between words, such as the order in which the terms appear in the document and the syntax etc. Also, it can not distinct the senses of a single word under different context even if the senses are totally different.

One way to address the above problems is to use features with coarser granularity, such as phrases to replace or augment single words. Intuitively, phrases can reduce the uncertainty of the meaning of single words. For example, in "Java Script", the meaning of the word "Java" can only be "a type of programming language" instead of "island of Indonesia". Some researchers manage to get phrases through the background knowledge base in form of simple ontologies such as WordNet [17], while others apply Natural Language Processing (NLP) methods together with some heuristic rules to generate phrases [13]. Part of these approaches can obtain certain improvement of the classification performance, but others do not work. Another way to address the drawbacks of bag-of-words representation is to use sense-based features instead of word-based features [11]. To detect the senses of a word, some extra resources, such as WordNet are needed. There are also some other approaches which represent documents through hidden concepts discovered from the documents or use language models, such as n-gram models [3, 1, 21].

Most of the above methods either need extra resources which may not be available for some text classification tasks, or they need very large space for models. In this paper, we enrich the document representation approaches for text classification purposes by applying a language model named n-multigram [6]. Our proposed approach is supposed to overcome the disadvantages of the previous document representation schemes. n-multigram can automatically discover the hidden semantic sequences in the documents under each category. A sequence is a frequently-occurring pattern in a given collection which consists of several continuous words in documents. It is not required to be a meaningful phrase, although it always turns out to be. The lengths of the discovered sequences are not necessarily same as in n-gram which makes it more flexible. The sequences are discovered through an Expectation-Maximization algorithm together with Viterbi training. In the training process, the vocabulary of the sequences and their probabilities are updated iteratively to maximize the likelihood of the training data.

Based on n-multigram models we put forward a text classifier and compare it with the classifiers based on n-gram models. The experiments on a subset of RCV1 dataset show that the classifier based on n-multigram models can achieve similar or even better performance compared to the classifier based on n-gram models in terms of both micro-F1 and macro-F1. However, the model size of our algorithm is much smaller than that of the latter. Another classifier we put forward is based on the combination of the n-multigram models and the n-gram models. This classifier improves the micro-F1 value from 89.5% to 92.6% and the macro-F1 value from 87.2% to 91.1%.

The rest of this paper is organized as follows: In Section 2, we present the related work on text representation. Then we give some detailed descriptions about the language models including n-gram models and n-multigram models in Section 3. In Section 4, we present the baseline classifier based on n-gram models together with our proposed classifiers based on n-multigram models and the combination of n-gram models and n-multigram models. Section 5 describes the experiments on RCV1 as well as some discussions. Finally, we conclude our work in Section 6.

2. RELATED WORKS

Much work has been conducted to find out effective approaches to represent document for text classification. We classify them into four groups: 1) Represent a document by phrases [2, 18, 13]; 2) Use the senses of words to represent a document [11, 22]; 3) Augment document representation by hidden concepts in a document [3]; 4) Employ language models, such as n-gram models [1, 21].

[2] and [18] represent documents by extracted phrases. The phrase extraction depends on the background knowledge embedded in existing ontologies such as WordNet, the MeSH (Medical Subject Headings) Tree Structure Ontology. In [2], Stephan et al. studied the representation approach on three data corpus and the experimental results proved the effectiveness of the approach. By extracting phrases based on a large publicly available knowledge base called the Unified Medical Language System (UMLS), Yetisgen-Yildiz and Pratt came to the same conclusion as in [2] on a dataset composed of medicine documents [18].

In [13], Lewis used syntactic parsing to create indexing phrases. These phrases correspond to pairs of words in one of several specified syntactic relationships in the original document (e.g. verb and head noun of subject, noun and modifying adjective, etc.). The sparseness of the phrases, that is, the large number of different phrases and the low frequency of occurrence of individual phrases makes it hard to estimate the relative frequency of phrases, which further limits the contribution of the syntactic phrase representation. What is more, a syntactic phrase representation is highly redundant (there are large numbers of phrases with essentially the same meaning), and noisy (since redundant phrases are not assigned to the same set of documents). To address these weaknesses, Lewis tried to cluster the phrases to recognize groups of redundant phrases and replace them with a single one. While the phrase clusters can improve performance, the improvement is not significant. Lewis ascribes the poor performance to the formation of phrases through syntactic parsing and the poor semantics of the phrases. In this paper, we propose a fully automatic approach to generate semantic phrases. We do not limit the length of a phrase to two as Lewis did and there is no constraint on the relationship between the words in the phrases. The discovered phrases are supposed to indicate the document content more precisely than single words.

Another attempt for document representation is to use sense-based features instead of word-based features [11, 22]. While words are immediately *observable* within a document, senses (meanings) are *hidden*. For instance, when the word "apple" appears in a document, it is not obvious whether it means a kind of fruit or an IT company. Therefore, a procedure is needed for recovering the senses from the words used in a specific context, which itself is a hot research problem: Word Sense Disambiguation [4, 8]. In [11], Kehagians et al. worked on a subset of the annotated Brown Corpus in which the sense of each word in the document is known. The sense space is defined by WordNet.

Kehagians et al. compared four document representation approaches. The first approach named Word Boolean(WB) is to represent a document by a vector in which each item is 0 or 1 to indicate whether the term appears in the document. The second approach named Word Frequency(WF) represents a document by a vector with each item being the number of times the word appear in the document. The other two approaches named Sense Boolean(SB) and Sense Frequency(SF) are computed in an exactly analogous manner to WB and WF, making use of senses rather than words. By testing the four representation approaches with several algorithms, the authors conclude that the sense-based representation approaches do not present an attractive alternative to word-based approaches.

Instead of converting each word to its sense and representing a document by the senses, Ramakrishnan and Bhattacharyya represent a document by a vector of ranked synsets [22]. More information about synsets can be found in [17]. To get the ranked synsets, they construct a semantic graph for each document based on the words in the document and the synsets in WordNet as well as the lexical relations between the synsets. After that, some graph-based ranking algorithms such as pagerank [20] are employed to rank the synsets. The experiments on the 20 Newsgroups dataset show that the representation approach based on certain ranking algorithms can improve the classification performance.

Cai and Hofmann proposed to use concept-based document representation generated through Probabilistic Latent Semantic Analysis (pLSA) to supplement word- or phrase-based features [3]. The overall approach can be decomposed into three stages: In the unsupervised learning stage, pLSA is used to derive concepts and to create semantic document representations over these concepts. In the second step, weak hypotheses are constructed based on both term features and concept features. The third stage is the combination stage, which uses AdaBoost to combine weak hypotheses and to integrate term-based and concept-based information. The experiments on two standard document collections from different domains support the validity of their approach.

Last but not least existing work to represent the documents is by language models, specifically speaking, by n-gram models, which can exploit the relationship between words. N-gram models model the language with the probability distribution of one word coming after the previous n-1 words. N-gram models can not only represent documents, it can also be directly used to classify documents as shown in [1, 21].

Although the approaches given above can make up for the bag-of-words representation scheme to some extent, there is still much space for improvement: (1) for the first two kinds of approaches, most of them need extra resources such as WordNet, which limit their generalization. For example, we can depend on WordNet to handle collections in English. However, there are many collections in some languages which have no counterparts of WordNet. (2) in the first two kinds of approaches, the phrases are generated without considering the labels of the documents. So the ability to discriminate documents between categories for the purpose of classification can be improved. (3) a complex framework is needed to leverage the concept-based document representation in [3]. What is more, the concepts are extracted without considering the category information, which is actually useful for classification. (4) n-gram can improve the classification performance as shown in [21, 1]. However, the size of the vocabulary of grams in n-gram is huge due to the combination. Then a large amount of training data are needed to obtain a creditable model.

3. N-GRAM AND N-MULTIGRAM MODELS

Language can be viewed as a stream of words put out by a source. Due to the syntactic and semantic constraints, the words are not independent. Many language models have been proposed to catch the characteristics of natural languages. In this part, we will describe two kinds of language models which can serve as document representation approaches. In the next part, we would introduce two text classifiers based on these models separately together with a novel classifier based on the combination of the two kinds of models.

3.1 n-gram models

N-gram models are a kind of widely used language models. It assumes that the probability of one word in a document depends on its previous $n-1$ words. Given a word sequence $W = w_1 w_2, \ldots, w_T$, the probability of W can be calculated as follows by the chain rule of probability:

$$p(W) = \prod_{i=1}^{T} p(w_i|w_1 ... w_{i-1}) \quad (1)$$

Under n-gram models' assumption, the only words relevant to predicting $p(w_i|w_1 ... w_{i-1})$ are the previous n-1 words, so $p(s)$ can be written as:

$$p(W) = \prod_{i=1}^{T} p(w_i|w_{i-n+1} ... w_{i-1}) \quad (2)$$

$p(w_i|w_{i-n+1} ... w_{i-1})$ can be estimated from a corpus with Maximum Likelihood criteria. That is:

$$p(w_i|w_{i-n+1} ... w_{i-1}) = \frac{c(w_{i-n+1} ... w_i)}{c(w_{i-n+1} ... w_{i-1})} \quad (3)$$

where $c(\cdot)$ denotes the number of occurrence.

In real-world applications, $p(w_i|w_{i-n+1} ... w_{i-1})$ is often under-estimated due to the data sparseness in a training data set. For example, many grams are assigned zero probability when they do not appear in the training data. This is unreasonable since the zero count may be generated because of the insufficient data. To solve this problem, some mechanisms are indispensable to assign non-zero probability of potentially missing n-grams. The widely used methods include linear interpolation or back-off estimators. Linear interpolation involves an EM procedure to optimize the weight for each component [5]. Back-off models are relatively simple and are used in this paper, as employed by [21].

In back-off models, the smoothing function is as follows:

$$p(w_i|w_{i-n+1} ... w_{i-1}) = \begin{cases} \widehat{p}(w_i|w_{i-n+1} ... w_{i-1}), & if\ c(w_{i-n+1} ... w_i) > 0 \\ \beta(w_{i-n+1} ... w_{i-1}) \times p(w_i|w_{i-n+2} ... w_{i-1}) & otherwise \end{cases} \quad (4)$$

where

$$\widehat{p}(w_i|w_{i-n+1} ... w_{i-1}) = \frac{discounted\, c(w_{i-n+1} ... w_i)}{c(w_{i-n+1} ... w_{i-1})} \quad (5)$$

is the discounted probability and $\beta(w_{i-n+1} ... w_{i-1})$ is a normalization constant calculated to be:

$$\beta(w_{i-n+1} ... w_{i-1}) = \frac{1 - \sum_{x:c(w_{i-n+1} ... w_{i-1} x)>0} \widehat{p}(x|w_{i-n+1} ... w_{i-1})}{1 - \sum_{x:c(w_{i-n+1} ... w_{i-1} x)>0} \widehat{p}(x|w_{i-n+2} ... w_{i-1})} \quad (6)$$

An n-gram is first matched against the language model to see if it has been observed in the training corpus. If that fails, the n-gram is then reduced to an n-1-gram by shortening the context by one word. The discounted probability (5) can then be computed using different smoothing approaches. A properly smoothing method can not only help prevent zero probabilities, but also improve the estimation accuracy of the language model. The standard smoothing approaches include linear smoothing [19, 27], absolute smoothing [19], Good-Turing smoothing [10] and Witten-Bell smoothing [27]. In this paper, we just consider the last two methods for simplicity.

Good-Turing smoothing: Good-Turing smoothing discounts the frequency of r by $GT_r = (r+1)n_{r+1}/n_r$ where n_r denotes the number of events which occur exactly r times

in training data [19]. The discounted probability is calculated as [10]:

$$\widehat{p}(w_i|w_{i-n+1}...w_{i-1}) = \frac{GT_{c(w_{i-n+1}...w_i)}}{c(w_{i-n+1}...w_{i-1})} \quad (7)$$

Witten-Bell smoothing: Witten-Bell smoothing is very similar to Laplace smoothing [15], except it reserves probability mass for out of vocabulary (OOV) values, whereas Laplace smoothing does not. Here the discounted probability is calculated as:

$$\widehat{p}(w_i|w_{i-n+1}...w_{i-1}) = \frac{c(w_{i-n+1}...w_i)}{c(w_{i-n+1}...w_{i-1}) + D(w_{i-n+1}...w_{i-1})} \quad (8)$$

where $D(w_{i-n+1}...w_{i-1})$ is the number of distinct words that can follow $w_{i-n+1}...w_{i-1}$ in the training data [27]. In the uni-gram model, this corresponds to the size of vocabulary.

3.2 n-multigram models

While in n-gram models, the statistical dependencies between words are of fixed length n along the whole sentence, in n-multigram models, the dependencies are of variable lengths. n-multigram models assume that a sentence is a concatenation of independent variable-length sequences of words and the number of words in each sequence is not necessarily same but is at most n. The parameters of n-multigram models consist of the probability of each sequence. An estimation of the set of parameters from a training corpus W can be obtained through a Maximum Likelihood (ML) estimation from incomplete data [7], where the observed data is the string of words W, and the unknown data is the segmentation S underlying the string of words. Thus iterative ML estimation of the parameters can be computed through an EM algorithm. The algorithm is given in Algorithm 1.

Algorithm 1 : EM algorithm for parameter estimation

Step1: Collect all possible sequences from the training text; remove the sequences appearing strictly less than a given number of times c_0 to avoid overfitting; calculate their initial probability as their relative frequency;

Step2a: E-step: to segment the training data into sequences according to the estimated probability distribution;

Step2b: M-step: to estimate the probability distribution from the segmentation; sequence probabilities falling under a threshold p_0 are set to 0, except those of length 1 which are assigned a minimum probability p_0;

Step2c: go to step2a if the likelihood of the training text can be improved or the number of iterations has not reached the predefined number K;

In Algorithm 1, Step 1 initializes the parameters; Step2a is the E-step of the EM algorithm which focuses on the segmentation of training text into sequences. When segmenting the text, we need to calculate the likelihood; Step2b is the M-step which estimates the parameters from the segmentation obtained in Step2a. Now we give the explanation of the key components of this algorithm.

Likelihood Calculation: Let $W = w_1w_2...w_T$ denote a string of words, and S denote a possible segmentation which contains q sequences of words: $S = s_1s_2...s_q$. The n-multigram computes the joint likelihood $L^{'}(W,S)$ of the text stream associated with the segmentation S as the product of the probabilities of the successive sequences:

$$L^{'}(W,S) = \prod_{t=1..q} p(s_t) \quad (9)$$

Denoting as **S** the set of all possible segmentations of W into sequences of words, the likelihood of W is :

$$L(W,S) = \sum_{S\in\{\mathbf{S}\}} L^{'}(W,S) \quad (10)$$

Algorithm 2 : Viterbi for segmentation

Initialization:
$\delta(0) = 1; \psi(0) = 0$
//$\delta(t)$ denotes the maximal likelihood of $w_1w_2...w_t$ based on the previously estimated probability distribution of sequences.
//$\psi(t)$ refers to the last segmenting position when we obtain $\delta(t)$

Iteration:

$$\delta(t) = \max_{l=1...n} \delta(t-l)p([w(t-l+1)...w(t)]) \quad (11)$$

$$\psi(t) = t - \arg\max_{l=1..n} \delta(t-l)p([w(t-l+1)...w(t)]) \quad (12)$$

Termination:
$L^*(W) = \max_{S\in\{S\}} L^{'}(W,L) = \delta(T)$
$i(T) = \psi(T)$

Backtracking:
$i(k) = \psi(i(k+1)-1)$
//$i(k)$ represents the starting position of each sequence.

Segmentation: Given a stream of text $W = w_1w_2...w_T$, we can segment it in different ways. The number of the segmentations can be up to $O(2^T)$. In fact, among the set of all possible segmentations of W, one segmentation usually contributes most of the likelihood of W according to our observation. We can use Viterbi algorithm to get the most likely segmentation. The algorithm is given in Algorithm 2.

Estimation: After getting the segmentations of the training text, we can estimate the parameters in two ways. One is to take all the segmentations into consideration. Another one is to consider the most likely segmentation only. For the former approach, the parameters can be estimated as follows:

$$p(s_i)^{(k+1)} = \frac{\sum_{S\in\{\mathbf{S}\}} c(s_i,S) \times L^{'}(S|W;\Theta^{(k)})}{\sum_{S\in\{\mathbf{S}\}} c(S) \times L^{'}(S|W;\Theta^{(k)})} \quad (13)$$

$c(s_i,S)$ represents the number of occurrence of sequence s_i in the segmentation S; $c(S)$ represent the total number of sequences in S; $\Theta^{(k)}$ is the parameters obtained at iteration k; $L^{'}(S|W;\Theta^{(k)})$ is the conditional likelihood of the segmentation S given W, at iteration k.

When considering the most likely segmentation alone, we can simplify (13) as follows:

$$p(s_i)^{(k+1)} = \frac{c(s_i, S^{*(k)})}{c(S^{*(k)})} \tag{14}$$

$S^{*(k)}$ denotes the most likely segmentation based on the parameter obtained in iteration k.

As shown in [6], the performances of the two above approaches are similar. So in this paper, we adopt the second one which is much simpler.

To calculate the probability of a test document under a given n-multigram model, the main problem we concern is how to handle unknown words which do not appear in the training data. In this paper, we first scan the test document to find out the number of the unique unknown words (instead of the number of occurrence of the unknown words) and then assign each of them the probability p_0. After that, we normalize the probability distribution to make sure the summation of all sequences' probability is 1.

4. CLASSIFIERS BASED ON LANGUAGE MODELS

4.1 n-gram models based classifier

It is straightforward to construct text classifiers based on the n-gram models [21]. Given an n-gram model, it is easy to get the probability of a document to indicate the likelihood that the document is generated by this model. After training the n-gram model on the training data in each category, we can use the model to classify a test document d by:

$$c^* = \arg\max_{c \in C}\{p(c|d)\} \tag{15}$$

By using Bayes rule, this can be written as:

$$\begin{aligned} c^* &= \arg\max_{c \in C}\{p(c)p(d|c)\} \\ &= \arg\max_{c \in C}\{p(c)\prod_{i=1}^{T} p(w_i|w_{i-n+1}...w_{i-1}, c)\} \\ &= \arg\max_{c \in C}\{p(c)\prod_{i=1}^{T} p_c(w_i|w_{i-n+1}...w_{i-1})\} \end{aligned} \tag{16}$$

Here, $p(d|c)$ is the likelihood of d under category c, which can be computed by an n-gram language model. The prior $p(c)$ can be estimated from training data or can be used to incorporate more assumptions. $p_c(w_i|w_{i-n+1}...w_{i-1})$ is estimated from the training data in category c using a back-off model introduced in section 3.1.

Thus the classifier based on n-gram models overall is to learn a separate back-off language model for each category by training on a data set from that category. Then, to categorize a new text d, we supply d to each language model, evaluate the likelihood of d under the model, and pick the winning category according to Equation (16). In this paper, two smoothing methods are employed, one is good-turing smoothing and the other is witten-bell smoothing. We use C_{NG} and C_{NW} to represent the classifiers based on n-gram models with these two smoothing methods respectively.

4.2 n-multigram models based classifier

Just as n-gram models can act as text classifiers, n-multigram models can be classifiers also. The frameworks of them are similar. During the training stage, we construct a separate n-multigram model for each category by training on the text from that category. Given a test document, we calculate its probability under each model, and assign the category under which the document has the largest probability to it. When calculating the probability under a model, we first find out the most likely segmentation of the test document using the viterbi algorithm shown in algorithm 2 and then compute the probability according to equation (9). C_M is used to denote the classifier based on n-multigram models.

By intuition, the longer a sequence, the more specific its meaning and the more powerful its discriminative ability for text classification. So we put forward two variations of C_M to give bonus to longer sequences. We can give the bonus when doing segmentation by changing equation (11) to equation (17) which will affects the segmentation results and tends to produce longer sequences. We can also give bonus after the segmentation by changing equation (9) to (18) which just adds bonus to the calculated likelihood. In equation (17) and (18), $b(s)$ is the bonus function of s. Two bonus functions are given in equation (19) and (20).

$$\delta(t) = \max_{l=1...n} \delta(t-l)p([w(t-l+1)...w(t)])*b(w(t-l+1)...w(t)) \tag{17}$$

$$L^{'}(W,S) = \prod_{t=1..q} p(s_t)b(s_t) \tag{18}$$

$$b_1(s) = e^{|s|*|s|} \tag{19}$$

$$b_2(s) = e^{|s|*|s|*|s|*|s|} \tag{20}$$

Now we prove the correctness that $b_1(s)$ will give bonus to longer sequences. The proof of $b_2(s)$ is similar. Given a sequence s, the bonus for it is $e^{|s|*|s|}$. If s is segmented into s_1, s_2, the bonus for them are $e^{|s_1|*|s_1|}$ and $e^{|s_2|*|s_2|}$ respectively. It is easy to see that $e^{|s|*|s|} = e^{(|s_1|+|s_2|)*|(|s_1|+|s_2|)}$ is larger than $e^{|s_1|*|s_1|} * e^{|s_2|*|s_2|}$ which is $e^{|s_1|*|s_1|+|s_2|*|s_2|}$.

It is obvious that $b_2(s)$ gives much more bonus to the longer sequences than $b_1(s)$. Depending on the time we give the bonus (during or after segmentation) and the function of the bonus, we get four variations of the classifier based on n-multigram, the denotations of the variations is summarized in Table 1.

	During segmentation	After segmentation
$b_1(s)$	C_{MB11}	C_{MB12}
$b_2(s)$	C_{MB21}	C_{MB22}

Table 1: Denotations of the four variations of C_M

4.3 Classifier based on the combination of n-gram and n-multigram models

In this section, we will put forward a classifier based on the combination of the n-multigram models and the n-gram models. The workflow of the proposed classifier is shown in Figure 1. From the above description, we can see that the n-multigram models can detect the frequent and usually meaningful sequences in the text. So it is natural to use the sequences generated by n-multigram models to represent the documents instead of the single words. Then a document changes from a sequence of words to a sequence of "sequences". From the text represented by sequences,

we can construct classifiers based on n-gram models. After that, we can apply the classifiers on the test documents which have been converted to sequences according to the corresponding n-multigram models.

Specifically, similar to n-gram and n-multigram based classifiers, the classifier based on the combination of n-gram and n-multigram models detects the label of a test document as follows:

$$c* = \arg\max_{c \in C}\{p(c|d)\} = \arg\max_{c \in C}\{p(d|c) * p(c)\} \quad (21)$$

where:

$$p(d|c) \propto p(d'|c_{n-gram}) \quad (22)$$

We use $c_{n-Multigram}$ to denote the n-multigram model constructed from the documents of category c and c_{n-gram} to denote the n-gram model trained on the segmented results of the training documents according to $c_{n-Multigram}$. d' is the most likely segmentation of d according to $c_{n-Multigram}$, which is essentially a sequence of "sequences". $p(d'|c_{n-gram})$ is the probability of d' under c_{n-gram} . For simplicity, we set n = 3 and employ the Witten-Bell smoothing approach when training the n-gram models on the training data represented by sequences. We use C_{M+N} to represent the classifier based on the combination of n-multigram models and n-gram models.

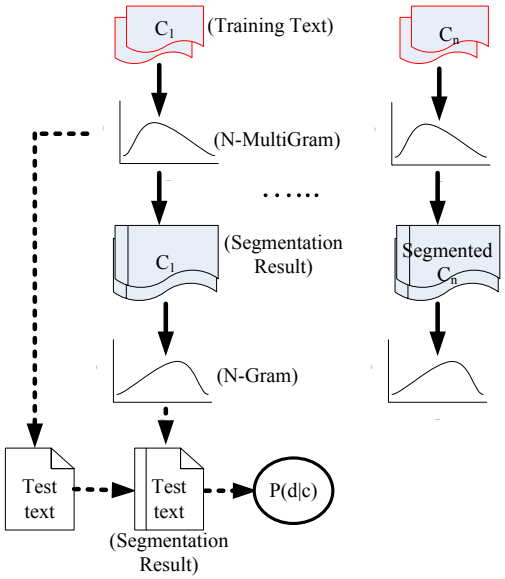

Figure 1: Framework of the classifier based on the combination of n-gram and n-multigram models.

4.4 Discussions of classifiers

Different categories have different usage of words and expressions. For example, among the several different disciplines in computer science, "data mining", as a sequence, appears more frequently in "Artificial Intelligence" than in "Network" while "data transition" behaves oppositely. These sequences are more discriminative when we perform text classification. We can label a document with the sequence "data transition" as "Network" with higher confidence than to label a document with the single word "data" or "transition" or even both of the two words which appear in different parts of the document. However, it is hard to construct the vocabulary of the specific sequences for each category manually. So in this paper, we propose to use n-multigram models to automatically generate such sequences and construct classifiers based the generated sequences.

N-gram models capture the specialty of natural language from another aspect, that is the occurrence of a word depends on its previous words. Traditionally, n-gram models are trained on single words. In this paper, we propose a novel classifier by applying the n-gram models on the sequences produced through n-multigram models. This classifier is supposed to perform better than the classifiers based on n-gram models and n-multigram models separately.

From the description of the training algorithm of the n-multigram models, it is easy to see that the time complexity for training classifiers based on n-multigram models is O(K$\times$ M $\times$ T $\times$ n) where K is the number of iteration; M is the number of training documents; T is the average length of the training documents in terms of words and n is the maximum length of the generated sequences. The time complexity for training classifiers based on the n-gram models is similar to that for the n-multigram models based classifier except that n-gram models training do not need run K iterations. However, the back-off step to improve the accuracy of n-gram models takes extra time. The training time for the classifier based on the combination of the n-gram models and the n-multigram models is about the sum of the complexity for training classifiers based on each single model. The time complexity for testing under each kind of classifiers is similar to the training complexity. The space complexity of the three classifiers is at the same order in theory. However, in reality, the model size of C_M is much smaller than that of C_N and the sizes are similar for C_N and C_{M+N}.

5. EXPERIMENT

As shown in [21], the classifier based on n-gram models is one of the most promising classifiers for text classification which beats Support Vector Machine (SVM) on some corpora. Therefore, in this paper, we take n-gram as the baseline and compare it with our proposed classifiers in two aspects: (1) we compare it with the classifier based on n-multigram models; (2) we compare it with a classifier based on combination of n-gram models and n-multigram models.

5.1 Dataset

Our experiments are conducted on RCV1, a new benchmark collection for text classification research [14]. RCV1 is drawn from one of the online databases produced by Reuters which is the largest international text and television news agency. RCV1 was intended to consist of all and only English language stories produced by Reuters journalists between August 20, 1996, and August 19, 1997. The stories cover the range of content typical of a large English language international newswire. They vary from a few hundred to several thousand words in length. The number of documents contained in RCV1 is about 35 times more than that of the popular Reuters-21578 collection and its variants [12].

In this paper, in order to speed up our experiments, we select 10% documents from the top 10 categories which results in 47042 documents. Table 2 shows the number of the sampled documents in the three largest categories and three smallest categories. In order to reduce the uncertainty of data split, a 3-fold cross validation procedure is applied in

our experiments. That is, we randomly split the documents into 3 folds and pick up one fold as the test data and the other two folds as the training data each time.

Table 2: The number of sampled documents in the three largest and smallest categories.

	Number of Documents
Three largest categories	19433
	7977
	5271
Three smallest categories	2158
	2102
	1978

5.2 Evaluation metric

We employ the standard measures to evaluate the performance of text classification, i.e. precision, recall and F1-measure [26]. Precision (P) is the proportion of actual positive class members returned by the system among all predicted positive class members returned by the system. Recall (R) is the proportion of predicted positive members among all actual positive class members in the data. F1 is the harmonic average of precision and recall which is defined as $F1 = 2 \times P \times R/(P + R)$.

To evaluate the average performance across multiple categories, there are two conventional methods: micro-average and macro-average. Micro-average gives equal weight to every document, while macro-average gives equal weight to every category, regardless of its frequency. Statistical significance tests are useful in order to verify to which extent the claim of an improvement can be backed by the observations on the test set. For the experiments we report in this paper, we focused on a pair-difference t-test on an improvement of individual F1 scores for the different classes that have been evaluated in each experiment. By convention, we say that a difference between means at the 95% level is "significant", a difference at 99% level is "highly significant" and a difference at 99.9% level is "very highly significant".

5.3 Preprocessing

Some preprocessing steps are applied. All the documents are converted into lower-case. Each document is tokenized with a stop-word remover and Porter stemming. To speed up the classification, a simple feature selection method, known as "document frequency thresholding (DF)" [29], is applied in our experiments, where we only select the top 60000 words.

5.4 An example of the output of the n-multigram models

Before coming to the details of the experiment, we provide an example to illustrate the output of n-multigram models, which is helpful to obtain some straightforward ideas of the advantages of n-multigram models. Take the sentence "President Guntis Ulmanis arrived in France on a two-day official visit, whose goal is to draw attention to security issues in the Baltic region." as an example, after removing the stopwords and stemming, we apply n-multigram models to segment the sentence into sequences and the result looks like "/ presid gunti ulmani / arrive / franc / dai offici visit / goal / draw attent / secure issu / baltic / region /". As we can see, the resulted sequences are meaningful, such as "President Guntis Ulmanis (presid gunti ulmani)" and "security issues (secure issu)". This case shows the ability of the n-multigram models to generate reasonable text representations.

5.5 Results and analysis

Table 3 and Table 4 show the classification performance of different classifiers in terms of micro-F1 and macro-F1 respectively. The best result for each classifier is in bold. For the classifiers based on n-multigram models, besides the maximum length of sequences (n), there are three parameters which need tuning. They are the occurrence threshold (c_0), the probability threshold (p_0) and the number of iteration (K). The parameters are set as follows: $c_0 = 5$, $p_0 = 10^{-7}$ and K = 10, for the results shown in Table 3 and Table 4. We will study the parameters in next section.

Table 3: Classification performance of different classifiers in terms of micro-F1

	n = 1	n = 2	n = 3	n = 4
C_{NG}	0.856	0.890	**0.892**	0.889
C_{NW}	0.857	0.891	**0.895**	0.890
C_M	0.855	0.888	0.891	**0.893**
C_{MB11}	0.855	0.890	0.892	**0.895**
C_{M+N}	0.859	0.921	**0.926**	0.920

Table 4: Classification performance of different classifiers in terms of macro-F1

	n = 1	n = 2	n = 3	n = 4
C_{NG}	0.837	0.865	**0.868**	0.863
C_{NW}	0.838	0.871	**0.872**	0.869
C_M	0.837	0.867	0.868	**0.870**
C_{MB11}	0.837	**0.864**	0.863	0.863
C_{M+N}	0.840	0.908	**0.911**	0.909

From the above tables, we can see that C_{NG} and C_{NW} do not make much difference which means that the two smoothing approaches, good-turing smoothing and witten-bell smoothing, behave similarly when they are applied for text classification. The same observation is shown in [21].

The results obtained by C_M and C_{MB11} are similar when the length of the multigram is changed from 1 to 4, although C_{MB11} gets some minor improvement in terms of micro-F1 in some cases and C_M outperforms C_{MB11} in terms of macro-F1 in some cases. The reason to explain this observation is as follows: the categories with relatively more training data tend to generate more sequences which may coincide with those generated in the categories with less training data. When we give bonus to longer sequences, some test documents at the boundary of a large category and a small category will have higher probability to be classified into the larger categories. Though this bias can give the test documents at the boundary which are from the larger categories with right labels and improve the micro-F1 value but the wrong decision on the test documents which are from the relatively smaller categories will affect the F1 values of the smaller categories more obviously, which then reduces the macro-F1 value.

As shown in Table 3 and Table 4, the classifiers based on the n-multigram models achieve the similar or even better

performance compared to the classifiers based on the n-gram models. This observation is encouraging since the model size for the former classifiers is much smaller than that of the latter ones.

From Table 3 and Table 4, we can see that C_{M+N}, the classifier based on the combination of n-gram models and n-multigram models outperforms both the classifiers based on n-gram models (C_{NG} and C_{NW}) and n-multigram models (C_M and C_{MB11}) respectively. C_{M+N} improved the micro-F1 from 0.895 to 0.926 and the macro-F1 from 0.872 to 0.911 when compared to the best result obtained by classifiers based on a single model only. Using the t-test, we find that the difference between C_{M+N} and other classifiers is "very highly significant" at $\alpha = 0.0003$.

Table 5: Number of items for each classifier

	n = 1	n = 2	n = 3
C_{NW}	60000.0	229848.6	373656.7
C_M	60000.0	69034.5	72670.2
C_{M+N}	60000.0	265770.4	394064.7

The model size (in terms of the number of items) for C_M (a n-multigram models based classifier) is much less than that of C_{NW} (an n-gram models based classifier). The former is only 30.0% of the latter when n equals to 2 and 19.4% when n equals to 3. The model size of C_{M+N} is larger than that of C_{NW}, but it is only increased by 15.6%. From Table 5, we can also see that for C_M, the increase of the model size when we change n from 2 to 3 is much smaller than that when we change n from 1 to 2. Actually, when we increase n to some larger values, the size of the model does not change obviously. The reason may be that two-word and three-word sequences are common in English but the sequences with four or more words are rare. However, when we increase the parameter n of C_{NW}, the model size will be increased dramatically.

5.6 Parameters Tuning

For the classifiers based on n-multigram models, four parameters need tuning. They are the maximum length of sequences (n), the occurrence threshold (c_0), the probability threshold (p_0) and the number of iteration (K). For the variations of the classifier based on n-multigram models, we also need to test the different kinds of bonus functions.

Table 6 and Table 7 show the classification performance of C_M and its variations when c_0 varies from 1 to 15 in terms of micro-F1 and macro-F1 respectively. The values for p_0 and K are fixed: $p_0 = 10^{-7}$, $K = 7$. The best results for each case are in boldface. From the tables, we can see that nearly all the classifiers achieve their peak performance when c_0 equals to 5. The reason lies in the fact that when c_0 is too small, too many sequences which may be noise are kept. However when c_0 is too large, some meaningful sequences will be removed. In both cases, the classification performance will be reduced.

Table 8 and 9 show the classification performance of C_M and its variations when p_0 varies from 10^{-5} to 10^{-9} in terms of micro-F1 and macro-F1. The values of c_0 and K are fixed: $c_0 = 5$, $K = 7$. p_0 has two roles. One is to filter sequences whose probabilities fall below p_0, which can reduce the impact of noises. The other role is to guess the probability of unknown words at test stage. From the tables, we can see

Table 6: Classification performance of C_M and its variations in terms of micro-F1 when c_0 varies

	c_0	n = 1	n = 2	n = 3	n = 4
C_M	1	0.855	0.870	0.882	0.879
	5	0.855	0.888	0.891	**0.893**
	10	0.855	0.883	0.886	0.887
	15	0.855	0.882	0.885	0.855
C_{MB11}	1	0.855	0.866	0.880	0.880
	5	0.855	0.890	0.892	**0.895**
	10	0.855	0.885	0.887	0.887
	15	0.855	0.883	0.885	0.886
C_{MB12}	1	0.855	0.866	0.879	0.880
	5	0.855	0.891	0.891	**0.894**
	10	0.855	0.885	0.887	0.888
	15	0.855	0.883	0.885	0.885
C_{MB21}	1	0.855	0.860	0.877	0.875
	5	0.855	0.877	**0.878**	0.875
	10	0.855	0.873	0.875	0.873
	15	0.855	0.873	0.872	0.870
C_{MB22}	1	0.855	0.860	0.875	0.874
	5	0.855	0.876	**0.877**	0.874
	10	0.855	0.873	0.874	0.872
	15	0.855	0.873	0.872	0.869

Table 7: Classification performance of C_M and its variations in terms of macro-F1 when c_0 varies

	c_0	n = 1	n = 2	n = 3	n = 4
C_M	1	0.837	0.840	0.857	0.853
	5	0.837	0.867	0.868	**0.870**
	10	0.837	0.860	0.861	0.862
	15	0.837	0.857	0.860	0.859
C_{MB11}	1	0.837	0.836	0.854	0.854
	5	0.837	**0.864**	0.863	0.863
	10	0.837	0.858	0.858	0.857
	15	0.837	0.856	0.857	0.857
C_{MB12}	1	0.837	0.836	0.853	0.853
	5	0.837	**0.863**	**0.863**	0.861
	10	0.837	0.858	0.857	0.858
	15	0.837	0.856	0.857	0.856
C_{MB21}	1	0.837	0.830	0.849	0.844
	5	0.837	**0.846**	0.842	0.840
	10	0.837	0.839	0.838	0.834
	15	0.837	0.840	0.835	0.832
C_{MB22}	1	0.837	0.831	0.846	0.843
	5	0.837	**0.847**	0.843	0.840
	10	0.837	0.839	0.837	0.834
	15	0.837	0.839	0.835	0.832

that for almost all the classifiers, they reach their best performance when p_0 equals to 10^{-7}. In fact, when p_0 varies from 10^{-7} to 10^{-8}, the performance does not change much. Such a wide range makes it easy to tune the parameter in real applications.

From Table 6 to Table 9, we can see that for the variation of C_M, the time when we give the bonus (during or after segmentation) does not make much difference. But the bonus function impacts the classification results obviously. Compared to the function shown in equation (19), the function shown in equation (20) reduces the performance in most cases. This observation indicates that when we give

Table 8: Classification performance of C_M and its variations in terms of micro-F1 when p_0 varies (the log value of p_0 is shown)

	$log(p_0)$	n = 1	n = 2	n = 3	n = 4
C_M	-5	0.491	0.564	0.570	0.569
	-6	0.824	0.872	0.875	0.877
	-7	0.855	0.888	0.891	**0.893**
	-8	0.857	0.885	0.888	0.890
	-9	0.854	0.880	0.883	0.885
C_{MB11}	-5	0.491	0.594	0.601	0.595
	-6	0.824	0.878	0.882	0.880
	-7	0.855	0.890	0.892	**0.895**
	-8	0.857	0.887	0.889	0.892
	-9	0.854	0.884	0.887	0.888
C_{MB12}	-5	0.491	0.594	0.600	0.595
	-6	0.824	0.879	0.880	0.880
	-7	0.855	0.891	0.891	**0.894**
	-8	0.857	0.887	0.888	0.890
	-9	0.854	0.884	0.886	0.887
C_{MB21}	-5	0.491	0.626	0.631	0.626
	-6	0.824	0.876	0.877	0.876
	-7	0.855	0.877	**0.878**	0.875
	-8	0.857	0.876	0.876	0.874
	-9	0.854	0.874	0.874	0.873
C_{MB22}	-5	0.491	0.626	0.630	0.626
	-6	0.824	**0.877**	0.876	0.875
	-7	0.855	0.876	**0.877**	0.874
	-8	0.857	0.875	0.875	0.874
	-9	0.854	0.873	0.874	0.873

Table 9: Classification performance of C_M and its variations in terms of macro-F1 when p_0 varies (the log value of p_0 is shown)

	$log(p_0)$	n = 1	n = 2	n = 3	n = 4
C_M	-5	0.563	0.637	0.638	0.636
	-6	0.816	0.857	0.859	0.861
	-7	0.837	0.867	0.868	**0.870**
	-8	0.836	0.862	0.864	0.865
	-9	0.829	0.854	0.856	0.858
C_{MB11}	-5	0.563	0.657	0.659	0.653
	-6	0.816	0.859	0.858	0.858
	-7	0.837	**0.864**	0.863	0.863
	-8	0.836	0.860	0.861	0.859
	-9	0.829	0.856	0.859	0.858
C_{MB12}	-5	0.563	0.657	0.658	0.652
	-6	0.816	0.859	0.857	0.856
	-7	0.837	**0.863**	**0.863**	0.861
	-8	0.836	0.858	0.860	0.859
	-9	0.829	0.852	0.859	0.857
C_{MB21}	-5	0.563	0.673	0.671	0.668
	-6	0.816	**0.851**	0.850	0.847
	-7	0.837	0.846	0.842	0.840
	-8	0.836	0.845	0.842	0.840
	-9	0.829	0.842	0.841	0.838
C_{MB22}	-5	0.563	0.673	0.672	0.668
	-6	0.816	**0.852**	0.850	0.846
	-7	0.837	0.847	0.843	0.840
	-8	0.836	0.844	0.842	0.839
	-9	0.829	0.841	0.841	0.835

too much bonus to longer sequences, the negative effect of some noise sequences will affect the classification result seriously which makes the performance worse. Another conclusion which can be drawn from the tables is that the four variations do not make much contribution to the classification performance. Although some of them can improve the micro-F1 value in some cases, they tend to reduce the macro-F1 at the same time. The reason for this observation is given in Section 5.5.

Table 10 shows the micro-F1 and macro-F1 values obtained by C_M when the number of iterations changes from 1 to 9. The values of other parameters are fixed as: $c_0 = 5$; $p_0 = 10^{-7}$; $n = 4$. It is easy to see that the performance of C_M converges quickly. This property makes C_M applicable in real applications.

Table 10: Performance of C_M when the number of iteration (K) changes

	1	3	5	7	9
micro-F1	0.879	0.887	0.893	0.893	0.893
macro-F1	0.856	0.867	0.870	0.870	0.870

6. CONCLUSION AND FUTURE WORK

In this paper, we proposed an approach of document representation based on the automatically extracted sequences generated through n-multigram models. Two text classifiers are put forward on the basis of the proposed document presentation. One is implemented by applying n-multigram models directly which can classify the documents while generating the sequences based representation at the same time. Another one is realized though the combination of n-multigram models and n-gram models. This algorithm works in three stages. In the first stage, n-multigram models are trained on the training documents for each category and the training documents are segmented according to the n-multigram models into sequences. In the second stage text classifiers based on n-gram models are trained on the sequences produced in the first stage. The third stage is to classify the test documents. For each test document, the pair of models (an n-multigram model and an n-gram model) from each category will be applied on the test document in turn. The document is segmented into sequences firstly according to an n-multigram model from a certain category and its probability of being generated by this category is calculated according to the corresponding n-gram model. The test document is assigned to the category which has the largest probability.

We conducted a series of experiments on a subset of RCV1. The experiments show that our proposed text classification algorithms work well. Although the model size of our proposed algorithm based on the n-multigram models directly is much smaller than that of the n-gram models based classifier, it can achieve similar or even better classification performance. The results also show that the proposed algorithm based on the combination of n-multigram models and n-gram models improves the micro-F1 and macro-F1 values from 89.5% to 92.6% and 87.2% to 91.1% respectively when compared with the classifiers based on n-gram models. The experiments are conducted through 3-fold cross validation and the t-test shows that the improvement is very highly significant to $\alpha = 0.0003$.

In our future work, we will test some other classification algorithms such as SVM and KNN based on our proposed document representation approaches to verify its validity. It is also necessary to conduct experiments on some other datasets to verify its adaptability.

7. ACKNOWLEDGMENTS

Dou Shen and Qiang Yang are supported by a grant from NEC (NECLC05/06.EG01). The authors would like to thank Prof. Mak Brian, Hui Zhao and the anonymous reviewers for their valuable comments and suggestions.

8. REFERENCES

[1] J. Bai and J.-Y. Nie. Using language models text classification. In *Proceedings of Asia Information Retrieval Symposium*, Beijing, China, Oct 2004.

[2] S. Bloehdorn and A. Hotho. Boosting for text classification with semantic features. In *the Workshop on Text-based Information Retrieval (TIR-04) at the 27th German Conference on Artificial Intelligence*, Sep 2004.

[3] L. Cai and T. Hofmann. Text categorization by boosting automatically extracted concepts. In *SIGIR 2003: Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 182–189, Toronto, CA, 2003.

[4] M. Carpuat and D. Wu. Word sense disambiguation vs. statistical machine translation. In *Proceedings of the 43th Annual Meeting of the Association for Computational Linguistics*, pages 387–394, June 2005.

[5] S. Chen and J. Goodman. An empirical study of smoothing techniques for language modeling. Tr-10-98, Harvard University, 1998.

[6] S. Deligne and F. Bimbot. Language modeling by variable length sequences: Theoretical formulation and evaluation of multigrams. In *Procedings of International Conference on Acoustics, Speech, and Signal Processing*, pages 169–172, Detroit, MI, 1995.

[7] A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society*, 39(B), 1977.

[8] A. Gliozzo, C. Giuliano, and C. Strapparava. Domain kernels for word sense disambiguation. In *Proceedings of the 43th Annual Meeting of the Association for Computational Linguistics*, pages 403–410, June 2005.

[9] T. Joachims. Text categorization with suport vector machines: Learning with many relevant features. In *Proceedings of European Conference on Machine Learning*, pages 137–142, 1998.

[10] S. Katz. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 35(3):400–401, 1987.

[11] A. Kehagias, V. Petridis, V. G. Kaburlasos, and P. Fragkou. A comparison of word- and sense-based text categorization using several classification algorithms. *Journal of Intelligent Information Systems*, 21(3):227–247, 2003.

[12] D. D. Lewis. Reuters-21578 text categorization test collection. September 26, 1997. http://www.daviddlewis.com/resources/testcollections/.

[13] D. D. Lewis. Representation quality in text classification: An introduction and experiment. In *Proceedings of a Workshop on Speech and Natural Language*, Hidden Valley, Pennsylvania, 1990.

[14] D. D. Lewis, Y. Yang, T. G. Rose, and F. Li. Rcv1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research*, 5:361–397, 2004.

[15] C. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, Massachusetts, 1999.

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

[17] G. Miller, R. Beckwith, C. Fellbaum, D. Gross, and K. Miller. Introduction to wordnet: an on-line lexical database. *International Journal of Lexicography*, 3(4):23–244, 1990.

[18] M.Yetisgen-Yildiz and W. Pratt. The effect of feature representation on medline document classification. In *Proceedings of the American Medical Informatics Association Fall Symposium*, Washington D.C., 2005.

[19] U. Ney, H. Essen and R. Kneser. On structuring probabilistic dependencies in stochastic language modeling. In *Computer Speech and Language*, pages 1–28. volume 8(1), 1994.

[20] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998.

[21] F. Peng, D. Schuurmans, and S. Wang. Augmenting naive bayes classifiers with statistical language models. *Information Retrieval*, 7(3-4):317–345, 2004.

[22] G. Ramakrishnan and P. Bhattacharyya. Text representation with wordnet synsets using soft sense disambiguation. *Ingénierie des Systèmes d'Information*, 8(3):55–70, 2003.

[23] G. Salton. *Automatic text processing: the transformation, analysis, and retrieval of information by computer*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.

[24] R. E. Schapire and Y. Singer. Boostexter: A boosting-based system for text categorization. *Machine Learning*, 39(2/3):135–168, 2000.

[25] S. Slattery and M. Craven. Combining statistical and relational methods for learning in hypertext domains. In *Proceedings of Inductive Logic Programming, 8th International Workshop*, pages 38–52, 1998.

[26] R. C. van. *Information Retrieval*. Butterworths, London, second edition edition, 1979.

[27] I. Witten and T. Bell. The zero-frequency problem: Estimating the probabilities of novel events in adaptive text compression. *IEEE Transactions on Information Theory*, 37(4), 1991.

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

[29] Y. Yang and J. O. Pedersen. A comparative study on feature selection in text categorization. In *Proceedings of International Conference on Machine Learning*, pages 412–420, 1997.