CALL CLASSIFICATION FOR AUTOMATED TROUBLESHOOTING ON LARGE CORPORA

Keelan Evanini[1,2]*, David Suendermann*[2]*, Roberto Pieraccini*[2]

[1]University of Pennsylvania, Philadelphia, USA
[2]SpeechCycle, Inc., New York City, USA
keelan2@ling.upenn.edu, {david, roberto}@speechcycle.com

ABSTRACT

This paper compares six classification algorithms for statistical semantic analysis in the framework of a dialog system for automated troubleshooting. The comparison is carried out on large datasets, each consisting of over 100,000 utterances (or 500,000 words) from two domains: Televison (TV) and Internet (INT). In spite of the high number of classes (79 for TV and 58 for INT), the best classifier (maximum entropy on word bigrams) achieved more than 77% classification accuracy on the TV dataset and 81% on the INT dataset.

***Index Terms*—** call classification, automated trouble shooting, large corpora

1. INTRODUCTION

State-of-the-art dialog systems for automated troubleshooting feature a very high complexity involving hundreds of caller-system interactions and human-agent-like problem solving behaviour [1]. Due to the large variety of call reasons such systems are able to handle, the identification of the call reason becomes an important issue. So far, most automated troubleshooting solutions have used either dual-tone multi-frequency signaling [2] or directed dialogs for call classification. Directed dialogs are driven by multiple choice questions, in which the user is prompted to respond with one from a small set of responses. However, a directed dialog is not practical for the task at hand for several reasons

- The number of call reasons, in the following referred to as *classes*, is much too large to be handled in a single directed dialog. Due to short-term memory limitations, it would be impossible to ask the caller to choose one out of 79 distinct choices. Even a hierarchically structured directed dialog would prove unwieldy with such a large number of classes.
- Callers often describe their problems using their own words, which might not be covered by the rule-based grammar typically used in conjunction with directed dialogs.
- Callers may not understand the terms used in a directed dialog. For example in response to the prompt: *Do you have a hardware, software, or configuration problem?*, they may respond unexpectedly (*My CD-ROM does not work!*), ask for help or an operator, etc.

corpus	utterances	classes
Gorin at al. [3] (1997)	10,000	15
Carroll and Carpenter [4] (1999)	3,753	23
Kuo and Lee [5] (2000)	4,000	23
Tur et al. [6] (2003)	21,953	49
Goel at al. [7] (2005)	33,274	35
TV	100,202	79
INT	137,570	58

Table 1. Comparison of corpora used in literature on call classification and those used in the current study.

In the late 90s, Gorin et al. [3] proposed the use of a statistical classifier to overcome these challenges. It is based on an open prompt, allowing the callers to freely describe the problem in their own words. For the current experiments, the utterances are all taken from callers' responses to the prompt: *Please describe the problem you're having in one short sentence*.

For training, a large number of utterances were collected from the two troubleshooting domains: Television (TV) and Internet (INT). The utterances were collected from customer support calls to an automated dialog system. They were manually transcribed and classified into one of several distinct call reasons that are acted upon by the dialog system, such as *ChannelMissing* for TV or *CantLoginPasswordEmail* for INT (for the corpus statistics, refer to Section 2.

Compared to previous studies reported in the literature, both the amount of training data and the number of classes in the current study are substantially larger. Table 1 shows a breakdown for corpora used in other similar systems and those presented in this study (TV and INT).

The training pairs consisting of the utterance and its corresponding class are then used to train a statistical model, which later, in the application phase, is used to determine the most likely class for a new caller utterance. This paper compares six algorithms for call classification:

- naïve Bayes,

- bag of words matching + naïve Bayes,
- naïve Bayes + boosting,
- decision trees,
- balanced winnow,
- maximum entropy.

These algorithms are briefly discussed in Section 2. Then, in Section 3, the corpora used in this study and the experimental framework are described, and detailed results of the experiments are reported. Section 4 discusses the outcomes taking the specifics of corpora and algorithms into account.

2. OVERVIEW OF CLASSIFIERS

This section provides a brief overview of the classification algorithms that were compared. The first two methods (bag-of-words matching and Naïve Bayes) were implemented by the authors. The next three were selected because they were three top performing classifiers from Mallet, a Java-based machine learning package tailored to natural language processing [8]. Finally, we implemented a boosting algorithm on several of the classifiers.

2.1. Data Representation

For all of the classifiers (except for the bag-of-word matching), each utterance was represented as a feature vector in which there was one feature for each lexical type (distinct word) in the dataset for the given domain. The values of the features are the token counts for each word that is present in the output of the ASR engine deployed in the automated system. If a type is not represented, then the feature value is 0. Since most utterances only contain a single instance of any type, this method often results in binary-valued feature vectors. Furthermore, the feature vectors are sparse, since, in average, there are only about five types with non-zero counts out of a feature vector with more than 4,000 components representing the corpus vocabulary (see Table 2 for the corpus characteristics).

As experience from other natural language processing tasks like language modeling suggests, not only the presence or pure counts of word should be taken into account, but also contextual information. Therefore, we also included word bigrams and trigrams as features. This increased the number of features as reported in Table 2.

2.2. Bag-of-Word Matching

As mentioned above, most of the utterances in the dataset are quite short, with an average of 5.1 words per utterance. Furthermore, due to the nature of the troubleshooting task, many of the utterances recur frequently. As an extreme example of this, over 50% of the utterances in a frequently occurring TV class are instances of the same type. For such cases that have been seen in the training data, the simplest classifier would construct a rule mapping the test utterance to the class provided for the identical training utterance. We refer to this as *matching*. The existence of a large amount of annotated training data makes this approach viable for at least part of the corpus.

In order to reduce redundant information and enable the classifier to match a larger percentage of the test utterances, we transformed utterance into their bag-of-word representation by performing the following steps:

- Stop words were removed according to a list including 38 function words.
- The remaining words were stemmed using the Porter stemmer algorithm [9].
- Multiple occurances of words were eliminated.
- The order of the words was regularized by an alphabetic sort.

2.3. Naïve Bayes

The goal of the Naïve Bayes classifier is to provide the most likely class label, $\hat{c}$, from a set of class labels, C, given an utterance expressed by the word sequence $w_1^N := w_1, \ldots, w_N$

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}}\, p(c|w_1^N). \tag{1}$$

Using Bayes' Rule, this can be rewritten as:

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}}\, \frac{p(w_1^N|c)P(c)}{P(w_1^N)}. \tag{2}$$

Since the term $p(w_1^N)$ remanis constant, it can be removed from Equation 2. Finally, the classifier uses the Naïve Bayes conditional independence assumption to determine $P(w_1^N|c)$. This assumes that the probability of the utterance given a class is simply the product of the probabilities of each word in the utterance given the class yielding

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}}\, p(c) \prod_{n=1}^{N} p(w_n|c). \tag{3}$$

Both the prior probability, $p(c)$, and the conditional probability $p(w|c)$ were estimated by using the maximum likelihood estimate based on the training data. Laplacian smoothing with a floor value of 0.1 was applied.

2.4. Balanced Winnow

Balanced Winnow is an online, mistake-driven learning algorithm [10], [11]. The classifier proceeds by taking the dot

product of the feature vector $\boldsymbol{x}$ for test utterance and a weight vector $\boldsymbol{\omega}$ for each class:

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}}\, \boldsymbol{x} \cdot \boldsymbol{\omega_c} \tag{4}$$

If $\hat{c}$ is incorrect, the weight vector for the correct class is updated by multiplying each component corresponding to a non-zero feature in the feature vector by a constant $1 + \epsilon$, and the weight vector for the incorrect class by $1 - \epsilon$, with $0 < \epsilon \ll 1$. This procedure is conducted for multiple iterations over the training data.

2.5. Maximum Entropy

The maximum entropy paradigm [12] expresses the probability $p(c|w_1^N)$ introduced in Equation 1 by applying the following multiplicative decomposition

$$\begin{aligned} p(c|w_1^N) &= \frac{\prod_n \alpha(c|w_n)}{\sum_{c'} \prod_n \alpha(c'|w_n)} \\ &= \frac{\prod_w \alpha^{N(w)}(c|w)}{\sum_{c'} \prod_w \alpha^{N(w)}(c|w)} \\ &= \frac{\exp\left[\sum_w N(w) \log \alpha(c|w)\right]}{\sum_{c'} \exp\left[\sum_w N(w) \log \alpha(c|w)\right]}. \end{aligned} \tag{5}$$

Performing the argmax operation of Equation 1 ignoring the terms which are constant with respect to c, yields

$$\begin{aligned} \hat{c} &= \underset{c \in C}{\operatorname{argmax}}\, p(c|w_1^N) \\ &= \underset{c \in C}{\operatorname{argmax}} \sum_w N(w) \log \alpha(c|w). \end{aligned} \tag{6}$$

This expression includes the variables

- $N(w)$, which is the count of a word type in the utterance. The general principle of maximum entropy, however, allows for arbitrary (binary, integer, or real-valued) features to be used instead of the raw word count. In this paper's investigations, we used both word counts and bigram counts as features.
- $\alpha(c|w)$ with $\alpha(c|w) \geq 0$ and $\sum_c \alpha(c|w) = 1$, which are parameters depending on the class c and the particular word w. These parameters are estimated in training using algorithms like generalized iterative scaling [13] or the Broyden-Fletcher-Goldfarb-Shanno [14, 15, 16, 17] method, the latter being used in this study as it was observed to be more efficient [18].

2.6. C4.5

C4.5 is a decision tree classifier [19]. The classifier constructs a branching tree consisting of a set of features to test and the most likely class given the decision. The feature to test at each node is determined by calculating the maximum information gain over all possible splits. The information gain for splitting at a feature is defined as the difference in entropy of the distribution before the split $H(D)$ and the weighted sum of the entropies of the nodes after the split (for a split that has K possible outcomes):

$$IG = H(D) - \sum_{k=1}^{K} \frac{|D_k|}{|D|} \times H(D_k) \tag{7}$$

In order to make the classifier training computationally tractable, feature selection was conducted first on the datasets. The maximum number of features, which the algorithm could handle in a reasonable amount of time (about 24 hours on a 3 GHz Intel Xeon processor and 2 GB of memory) on the full dataset was determined to be 50. Two methods of feature selection were used: χ^2 and TFIDF [20]. Both produced similar results; those reported below used χ^2 for feature selection for C4.5.

2.7. Boosting

Boosting is an on-line learning algorithm in which the results of several classifiers (*weak learners*) are combined, as a function of each classifier's accuracy, to form a weighted majority prediction rule. The boosting algorithm used in the current experiments is AdaBoost.M2 (also implemented in Mallet), which is specificaly designed for multiclass classification tasks. The boosted classifier's decision is determined by the equation (see [21] for details):

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} \sum_{t=1}^{T} (\log \frac{1}{\beta_t}) h_t(x, c) \tag{8}$$

This expression includes the variables:

- $t = 1, 2, ..., T$, a round of boosting in which the weight vector over the weak learner is updated as a function of each weak learner's accuracy
- β_t, a variable determined by the pseudo-loss of the hypothesis
- h_t, a hypothesis from the weak learner in the form of a vector $X \times C -> [0, 1]$ with a confidence score for each class

3. EXPERIMENTS

In this section, we describe the characteristics of the automatic troubleshooting corpora and report on the experimental results of the classifier comparison.

	TV	INT
training utterances	91,746	125,665
test utterances	8,456	11,905
classes	79	58
average words per utterance	5.1	4.4
features (1grams)	4,125	4,475
features (1+2grams)	40,176	70,469
ASR word error rate	31.0	32.7

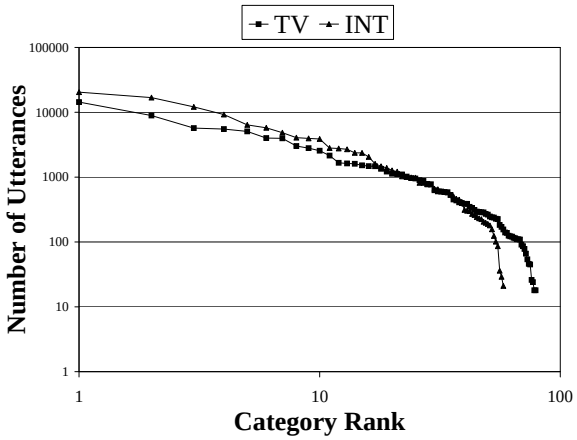

Fig. 1. Frequency distribution of the classes.

3.1. Corpora

The classification tests were conducted on a 90 / 10 split of each corpus into training and testing partitions. Table 2 shows the number of utterances in the training and test partitions for the full data sets.

The partitions were constucted such that the per-class distribution in each partition reflects the distribution in the corpus as a whole, i.e. if a class contains 2% of the overall utterances, then it will also contain 2% of the training and 2% of the test utterances. This was done to ensure that none of the classes would be omitted from the test set by a purely random sampling (some of the least frequent classes in the TV dataset contain fewer than 0.05% of the overall utterances).

This method of splitting the dataset was compared with a 10-fold cross validation on the INT dataset using a purely random 90 / 10 split for each iteration. The average performance of the 10 rounds was identical to the performance on the single dataset with the balanced 90 / 10 split (77.3%), thus demonstrating that this method of partitioning the data hardly influences the results.

Figure 1 shows how the utterances count per class sorted by descending counts. The distribution is nearly Zipfian, except for the fact that the most infrequent classes are too sparsely represented.

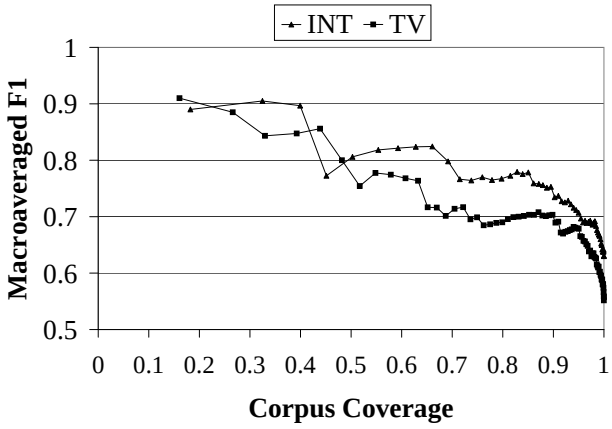

Fig. 2. Macroaveraged F_1 measure as a function of total coverage by the classes until that point.

3.2. Results

Tests were conducted on the datasets in Table 2 using all of the classification methods described in Section 2. Additional tests were conducted on smaller subsets of the TV corpus in order to see how the performance for each classifier changes with increased training data.

The accuracy is measured by overall percentage of correct classifications out of all test utterances. In general, the performance per class is better for the classes that are better represented in the datasets, as would be expected. The classes that have extremely poor performance only make up a small part of the dataset, as is shown in Figure 2 for the boosted Naïve Bayes classifier. This figure plots the F_1 measure [22] as defined by

$$F_1 = \frac{2 \cdot precision \cdot recall}{precision + recall} \tag{9}$$

for each class, ordered by decreasing frequency. The x-axis displays the percentage of utterances in the corpus that are covered by the classes until that point. The y-axis displays the macroaveraged F_1 measure for those classes. For example, the two most frequent classes in the TV corpus comprise 26.6% of the entire corpus, and they have an average F_1 of 0.89.

Figure 3 displays how the performance of the six classifiers improves with increased amounts of training data. The sizes of the training sets are approximately 1000, 2000, 5000, 10000, 20000, 50000, 100000 (the exact numbers differ slightly due to the fact that per-class distributions were preserved). All of these tests were performed on the test set of the TV corpus as specified in Table 2. Table 3 shows the results for the TV and INT corpora using the full train and test sets.

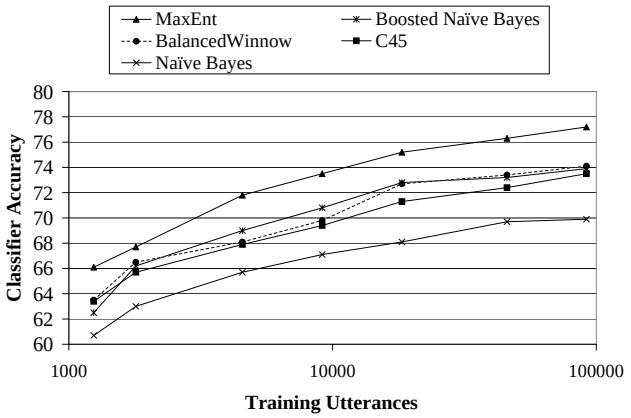

Fig. 3. Classification accuracy on the TV corpus as a function of amount of training data for the compared classifiers.

Classifier	TV	INT
Naïve Bayes	69.9	72.7
BOW + Naïve Bayes	74.1	75.5
C4.5	73.5	78.7
Boosted Naïve Bayes	74.9	77.3
Balanced Winnow	74.1	79.6
Maximum Entropy	**77.2**	**81.2**

Table 3. Comparison of the classification accuracy on full training set for TV and INT corpora. BOW stands for bag-of-word matching.

4. DISCUSSION

As the main outcome of the experiments reported in Figure 3 and Table 3, the performance of the maximum entropy classifier stands out. It consistently outperformed the competitors in all our experiments including all sets of corpora and n-gram order. This result agrees with experience from other classification tasks in natural language processing such as text categorization [23], part-of-speech tagging [24], or named entity recognition [25].

Let us now compare the winner to our two personal favorites: bag of words matching and boosting.

Bag of Word Matching. As pointed out earlier, bag of words matching does not cover all of the utterances, necessitating the use of a back-up classifier. Table 4 shows the percentages of test utterances, whose bag-of-words representation has been seen in training. It also reports the classifiation accuracy of the bag-of-words matching limited to the cases seen in training.

Unfortunately, the bag-of-word matching accuracy does not achieve 100% for the following drawbacks

Speech recognition errors (cf. Table 2) lead to erroneous bags of word and potentially wrong classes.

[%]	TV	INT
percentage of BOW seen in training	63.2	73.7
BOW accuracy on BOW data	88.3	85.1
maximum entropy accuracy on BOW data	88.6	89.0

Table 4. Results of bag-of-word matching compared to maximum entropy.

	TV	INT
utterances	100,202	137,570
bags of word	39,057	37,197
ambiguous bags of word	1,023	724

Table 5. Corpus statistics on bags of words with ambiguous classes.

In spite of the large amount of training data used in this study, there are more than 30% of the test utterances, whose bag-of-word representation has not been seen in the training data. Consequently, for this part of the data, another classification algorithm has to be applied; in this paper, we decided to use the Naïve Bayes classifier, described in Section 2.3, as update solution, since both algorithms can be integrated very easily.

The bag-of-word paradigm is based on the assumption that there is a non-ambiguous mapping from a given bag of word to a single class. In order to test this assumption, the training utterances of the corpora were collapsed into bags of word and those cases were isolated, which mapped to more than one class. Table 5 reports about the outcomes of this test.

These non-ambiguous cases could be due to a weakness of the bag-of-word approach, which assumes that only redundancy is removed. Therefore, all these cases were given to a human annotator for review. At the date of this publication, this review process is still in progress, but a the number of already finished cases suggest that the vast majority of them is due to inconsistent annotations. Only very few cases, where two utterances belonging to different classes but resulting in identical bags of words have been found. One example involved the following two utterances from different classes: "cancel a call" (*Appointment*) and "calling to cancel" (*ServiceCancel*). Both of these utterances were compressed to the bag of words "call cancel", and thus the bag-of-word classifier is not able to correctly distinguish them.

Interestingly, it turns out that the maximum entropy classifier outperformed bag of words matching even on the set of utterances whose bags of word have been seen in training, as shown in Table 4. Maximum entropy obviously features superior characteristics concerning data inconsistencies and recognition errors. It also takes context into account: as mentioned in Section 2.1, we ran experiments expanding unigrams to bigrams and trigrams extending the number of fea-

tures used for the classification. Table 6 shows results on TV data. This time, also a corpus variant comprising 50,000 utterances was used, since the test framework suffered memory problems when applying trigrams to the full 100,000 training utterances.

Enhancing unigrams by bigrams pushs the performance by 0.3 to 0.4%. Further extending the n-gram order does not seem to show a significant effect. At any rate, it seems that context, and consequently word order, plays a certain role distinguishing between classes.

Boosting. Attempts were made to improve the classifier performance through boosting. [26] demonstrates that boosting improves the performance of a C4.5 classifier on a wide variety of datasets, and [27] shows improved performance specifically for text categorization.

In our experiments, however, boosting only showed improved performance on the complete datasets with a Naïve Bayes classifier. The best results were obtained with 550 rounds of boosting, and are reported in Figure 3.

For the other classifiers, boosting showed no improvement, often even a slight decrease in performance when the entire training set was used. For smaller datasets, all classifiers did show some improvement with boosting. But when the training corpus is large and the classifier is strong enough, our results suggest that boosting is not helpful.

5. CONCLUSION

This paper reported on call classification experiments on large corpora comparing six classification algorithms. Most remarkable outcome is that the maximum entropy approach outperformed all other classifiers on all data sets. Furthermore, it turned out that boosting does not help on the investigated large data sets for all classifiers except for Naïve Bayes.

6. REFERENCES

[1] K. Acomb, J. Bloom, K. Dayanidhi, P. Hunter, P. Krogh, E. Levin, and R. Pieraccini, "Technical Support Dialog Systems: Issues, Problems, and Solutions," in *Proc. of the Workshop on Bridging the Gap: Academic and Industrial Research in Dialog Technologies*, Rochester, USA, 2007.

[2] "Interactive Services Design Guidelines," ITU, Geneva, Switzerland, Tech. Rep. ITU-T Recommendation F.902, 1995.

[3] A. Gorin, G. Riccardi, and J. Wright, "How May I Help You?" *Speech Communication*, vol. 23, no. 1/2, 1997.

	50,000	100,000
1gram	76.04	76.71
1+2gram	76.34	77.18
1+2+3gram	76.38	–

Table 6. Applying bigrams and trigrams as features to maximum entropy.

[4] J. Chu-Carroll and B. Carpenter, "Vector-Based Natural Language Call Routing," *Computational Linguistics*, vol. 25, no. 3, 1999.

[5] H.-K. Kuo and C.-H. Lee, "Discriminative Training in Natural Language Call Routing," in *Proc. of ICSLP'00*, Beijing, China, 2000.

[6] G. Tur, R. Schapire, and D. Hakkani-Tür, "Active Learning for Spoken Language Understanding," in *Proc. of the ICASSP'03*, Hong Kong, China, 2003.

[7] V. Goel, H.-K. Kuo, S. Deligne, and C. Wu, "Language Model Estimation for Optimizing End-to-End Performance of a Natural Language Call Routing System," in *Proc. of the ICASSP'05*, Philadelphia, USA, 2005.

[8] A. McCallum, "MALLET: A Machine Learning for Language Toolkit," `http://mallet.cs.umass.edu`, 2002.

[9] M. Porter, "An Algorithm for Suffix Stripping," *Program*, vol. 14, no. 3, 1980.

[10] I. Dagan, Y. Karov, and D. Roth, "Mistake-Driven Learning in Text Categorization," in *Proc. of the EMNLP'97*, Providence, USA, 1997.

[11] V. Carvalho and W. Cohen, "Single-Pass Online Learning: Performance, Voting Schemes and Online Feature Selection," in *Proc. of the KDD'06*, Philadelphia, USA, 2006.

[12] A. Berger, S. Pietra, and V. Pietra, "A Maximum Entropy Approach to Natural Language Processing," *Computational Linguistics*, vol. 22, no. 1, 1996.

[13] J. Darroch and D. Ratcliff, "Generalized Iterative Scaling for Log-Linear Models," *Annals of Mathematical Statistics*, vol. 43, no. 5, 1972.

[14] C. Broyden, "The Convergence of a Class of Double-rank Minimization Algorithms," *Journal of the Institute of Mathematics and Its Applications*, vol. 6, no. 1, 1970.

[15] R. Fletcher, "A New Approach to Variable Metric Algorithms," *Computer Journal*, vol. 13, no. 3, 1970.

[16] D. Goldfarb, "A Family of Variable-Metric Algorithms Derived by Variational Means," *Mathematics of Computation*, vol. 24, 1970.

[17] D. Shanno, "Conditioning of Quasi-Newton Methods for Function Minimization," *Mathematics of Computation*, vol. 24, 1970.

[18] R. Malouf, "A Comparison of Algorithms for Maximum Entropy Parameter Estimation," in *Proc. of the CoNLL'02*, Taipei, Taiwan, 2002.

[19] J. Quinlan, *C4.5: Programs for Machine Learning*. San Francisco, USA: Morgan Kaufmann, 1993.

[20] Y. Yang and J. Pedersen, "A Comparative Study on Feature Selection in Text Categorization," in *Proc. of the ICML'97*, Nashville, USA, 1997.

[21] Y. Freund and R. Schapire, "A Decision-Theoretic Generalization of on-Line Learning and an Application to Boosting," *Journal of Computer and System Sciences*, vol. 55, no. 1, 1997.

[22] C. van Rijsbergen, *Information Retrieval*. London, UK: Butterworths, 1979.

[23] K. Nigam, J. Lafferty, and A. McCallum, "Using Maximum Entropy for Text Classification," in *Proc. of the IJCAI'99*, Stockholm, Sweden, 1999.

[24] A. Ratnaparkhi, "A Maximum Entropy Model for Part-of-Speech Tagging," in *Proc. of the EMNLP'96*, Philadelphia, USA, 1996.

[25] A. Borthwick, "A Maximum Entropy Approach to Named Entity Recognition," Ph.D. dissertation, New York University, New York City, USA, 1999.

[26] J. Quinlan, "Bagging, Boosting, and C4.5," in *Proc. of the AAAI'96*, Portland, USA, 1996.

[27] R. Schapire and Y. Singer, "BoosTexter: A Boosting-Based System for Text Categorization," *Machine Learning*, vol. 39, no. 2/3, 2000.