Improving Query Classification by Features' Weight Learning

by

Arash Abghari

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2013

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

This work is an attempt to enhance query classification in call routing applications. A new method has been introduced to learn weights from training data by means of a regression model. This work has investigated applying the *tf-idf* weighting method, but the approach is not limited to a specific method and can be used for any weighting scheme. Empirical evaluations with several classifiers including Support Vector Machines (SVM), Maximum Entropy, Naive Bayes, and k-Nearest Neighbor (k-NN) show substantial improvement in both macro and micro F1 measures.

Acknowledgements

I would like to thank my supervisor Prof. Karray for guiding and supporting me and providing feedback. I would like to acknowledge my readers Dr. Bishop and Dr. Dabbagh for their helpful feedback and suggestions. I thank Dr. Ghodsi and Dr. Kohandel for their time and fruitful discussion. Many thanks are also due to my wife Nilufar for putting up with me and supporting and believing in me. I also acknowledge Vestec Inc. for providing me with the data sets used in this work.

Dedication

To my wife Nilufar, my daughters Parnia and Kimia, and my parents.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

A call routing function routes an incoming call to an appropriate destination. Human agents do this task very well, however, many enterprises try to reduce their reliance on human agents by utilizing self-service systems. To this end, touch tone systems have been used, but they are difficult to navigate and the user must often go through several levels of menus. Natural language call routing is an alternative approach to overcome these limitations and problems. The prompt to the customers is general and open and they are expected to express their request freely. The performance of natural language call routing systems is far from ideal. Having worked with some telecom call centers was the motivation to introduce a method to improve the quality of the call centers' performance.

1.1 Objectives

The goal of this work is to introduce a method based on weight learning to improve the performance of a call routing system specifically, and the query classification task in general. To achieve this, it requires the following steps:

- Introduce a method to properly calculate the weights from the training query collection.
- Introduce and develop an algorithm to learn the weights to be used to train classifiers.
- Apply the learned weights to testing queries.
- Compare the performance of a call routing application with and without weighting.

1.2 Thesis Organization

The remainder of the thesis is arranged as follows. Chapter 2 reviews call routing applications and feature weighting methods in the literature. Chapter 3 describes

the architecture of the proposed approach, such as training and testing phases, as well as the methods and techniques used in this work including different classifiers, feature selection, and performance measuring. Chapter 4 presents the results and performance discussion. Finally Chapter 5 summarizes the work and discusses future work.

Chapter 2

Literature Review

This work is intended to introduce a new approach for improving automatic call routing applications based on feature weighting methods. Figure 2.1 depicts a simplified automatic call routing system. An incoming call gets transcribed by a speech-to-text engine and then the destination of the call is determined by a classifier. The focus of this work is on the classifier part. The first part of this chapter reviews different techniques which have been applied to call routing applications in the literature. The second part describes different feature weighting schemes used in text categorization and information retrieval task.

2.1 Call Routing Applications

Gorin *et al.* [16, 31] first investigated the implementation of such a system using a probabilistic approach by utilizing a connectionist network. They use a single layer network to map input requests to the appropriate destination. The mutual information between features and destinations are used as the initial weights for the network.

Carpenter *et al.* [8] took a different approach by constructing an $m \times n$ routing matrix A where m is the number of terms and n is the number of destinations. The entry of $A_{t,d}$ is the term frequency-inverse document frequency (*tf-idf*) weight of term t with regards to document d. They then applied Singular Value Decomposition (SVD) to matrix A, to reduce the dimensionality of the routing matrix.

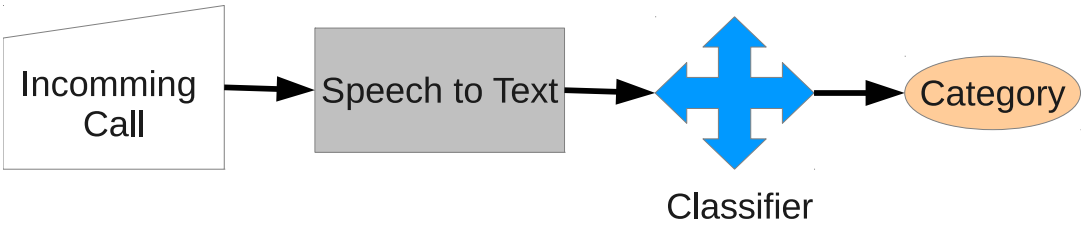

Figure 2.1: Simplified Automatic Call Routing System.

To determine the destination of each caller's request, the cosine similarity score between the request and each vector in the routing matrix is calculated. They achieved over 90% correct routing accuracy using the cosine scores.

Training a call router needs a remarkable amount of data. The data are usually transcribed by human experts from pre-recorded speech and require huge effort. To avoid this human effort, Huang *et al.* [18] experimented with building a call routing application without transcriptions. They used speech utterances directly to build a model. Their training algorithm has two steps. First extracting the important phoneme sequences from speech segments, and building a routing matrix based on them. Linear Discriminant Analysis (LDA) [43] is then applied to the routing matrix for classification and dimensionality reduction purposes. To extract the salient phoneme sequence and reduce error, different techniques are utilized, including mutual information (MI) and clustering. The accuracy they achieved is far below the one obtained by the system trained with the transcriptions, however considering the error rate of the phoneme recognizer, their result is encouraging. Note that they left the issue of building an independent language model (LM) for the phoneme recognizer (which usually depends on the transcriptions) for future research.

Kuo *et al.* [24] improved the method first introduced by Carpenter *et al.* [8]. They developed a method based on discriminative training to improve the performance of the routing matrix. They achieved this by adopting a Generalized Probabilistic Descent (GPD) algorithm. The actual minimization is applied to the loss function, which is a smooth differentiable function between 0 and 1, like a sigmoid function. Through the minimization, the elements of the routing matrix are adjusted to improve the separation between the correct class and the rest. By re-weighting the elements of the routing matrix, they achieved a 10% to 30% reduction of the relative error rate. The weights generated by this algorithm are very semantically intuitive and negate the need for applying a feature selection algorithm to filter out the least important feature from the training set. Since the irrelevant features get very low or negative weights during the training procedure, there is no need to remove stop words from the training data (a useful property considering the fact that the stop word list can be different from application to application, and needs to be fine-tuned by a human expert). Figure 2.2 shows the impact of discriminative training on the routing matrix entry for the "Deposit-Services" destination. One can see that after discriminative training, the irrelevant features are assigned either very low or negative values (anti-feature).

Cox [11] examined different discriminative techniques including Generalized Probabilistic Descent (GPD), Linear Discriminant Analysis (LDA), and Corrective Training (CT) [29]. The performance of these classifiers is compared to the baseline classifier, which is similar to the classifier introduced in [8] with a different weighting scheme. Although the error rate improvement over baseline classifier is very remarkable for the training set, they do not enjoy the same rate of improvement for the development set. In fact, the best result for the development set is

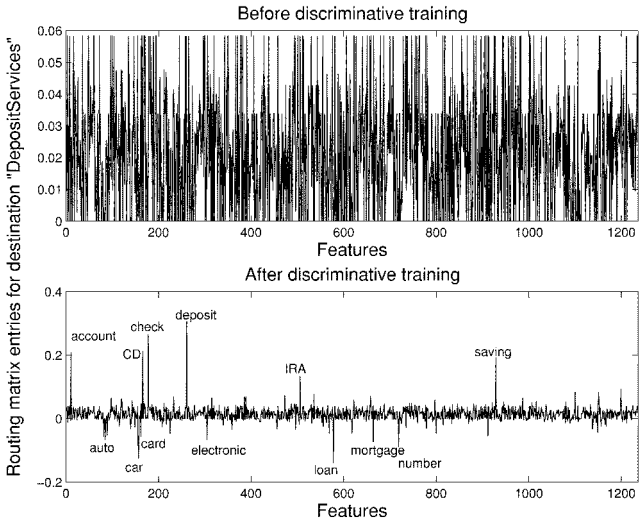

Figure 2.2: Weights of Destination "Deposit-Services" (Figure is taken from [24])

achieved by the GPD algorithm. Another experiment with a different type of classifiers was carried out by Wu *et al.* [44]. They evaluated several classifiers, including the Naive Bayes classifier, a discriminative trained LSI model, a multinomial distribution method with and without boosting algorithm, etc.. Their finding is that the discriminative trained LSI model and multinomial model with boosting are the best in terms of error rate. Liu *et al.* [19] incorporate the discriminative training introduced in [24] into a Naive Bayes Classifier (NBC). They experimented with two different version of NBCs. In the first version, all the NBCs corresponding to different classes were trained by the same feature set while the other version, used a separate dependent feature set for each class. They reported that the discriminative trained Naive Bayes classifiers outperformed the vector based classifier used in [24]. Tyson *et al.* [38] introduced a different approach by integrating confidence scores of a speech recognizer into a Latent Semantic Indexing (LSI)-based language call routing system. The idea is, words with a higher confidence score must have a bigger impact on the destination decision than words with lower confidence scores. They found that the confidence-based LSI classifier outperforms the standard one when the length of input utterance is at least eight seconds. Ullah *et al.* [39] developed a soft computing-based approach to improve the performance of a Naive Bayes classifier for call routing systems. The words in each category are grouped into a more relevant set with high weight, and a less relevant set with low weight. Then, the conditional probabilities of the words are adjusted using these weights and a threshold. The value of weights and threshold are then estimated by utilizing a soft computing technique such as a Genetic Algorithm (GA). This approach has shown remarkable improvement over the base line classifier in terms of accuracy.

Zitouni [46] investigated a method to combine multiple classifiers to improve the performance over any single classifier. He achieved this by using the Constrained

Minimization (CM) technique when the classifiers' errors are uncorrelated. The experiment is carried out using two classifiers: Cosine Similarity Classifier and Beta Classifier. The performance of the constrained minimization technique is then enhanced further by applying the GPD algorithm. Sarikaya *et al.* [33] investigated the use of Deep Belief Networks (DBN) in call routing applications. DBN is a feed-forward, multi-layer neural network whose weights have been initialized by an unsupervised generative probabilistic model called Restricted Boltzmann Machine (RBM) [17]. They compared the performance of a DBN with other classifiers such as Maximum Entropy, Boosting, and Support Vector Machines. The DBN classifier performed equally well or better than the others in this experiment.

2.2 Weighting Schemes

Term weighting has been well studied in the literature. Term weighting has been applied to both Text Categorization (TC) and Information Retrieval (IR). Gerard *et al.* [32] studied different methods of weighting suited for both document and query vectors. They investigated different forms and combinations of term frequency, collection frequency, and normalization components. They examined several combinations of binary weight, raw term frequency, and augmented normalized term frequency with a set of collection frequency components, including inverse document frequency and probabilistic inverse document frequency. They also investigated the impact of applying normalization to these combinations.

Yang *et al.* [45] studied different methods for feature selection. The methods used for feature selection include document frequency threshold, information gain (IG), mutual information, χ^2 statistic (CHI), and term strength. They found that the IG and CHI are the most effective methods for term selection without losing accuracy. Forman *et al.* [14] carried out another extensive study of feature selection methods. In this work, they utilized Bi-Normal Separation (BNS) for the first time as a feature selection metric, as well as other metrics used in the literature at the time. They concluded that BNS, IG, CHI, and Odds Ratio (OR) are the best metrics for feature selection.

Debole *et al.* [12] then introduced supervised term weighting based on the criteria used for feature selection. The idea is to replace the inverse document frequency (idf) part in the *tf-idf* weighting scheme with the criteria used for feature selection, since these criteria take category information into account. They specifically used information gain (*tf-ig*), chi-square (*tf-chi)*, and gain ratio (*tf-gr)* as a replacement for the *idf* part. However, empirical results showed that these new weighting schemes are not superior to *tf-idf*. Jin *et al.* [20] introduced a new scheme to automatically learn term weights based on the correlation between term frequency and document categories. This approach showed good performance in an Information Retrieval task. Soucy *et al.* [37] introduced a confidence weight scheme which estimates the importance of a word based on the statistical confidence intervals. This new scheme also replaces the *idf* part with the new criterion.

Forman [13] then expanded the previous work [14] on feature selection to supervised term weighting, which showed that BNS is an excellent ranking metric. He replaced *idf* with BNS and other ranking metrics and found that the *tf-BNS* combination indeed improves the performance of a SVM classifier, although *tf-idf* in his experiment outperformed other supervised term weighting schemes such as *tf-ig*, *tf-chi*, and *tf-or* (odds ratio), which is consistent with other findings in the literature. Batal *et al.* [3] used a supervised technique to boost the performance of a k-NN classifier. Two schemes, *tf-ig* and *tf-chi*, were selected as the term weighting scheme. They found that the supervised technique does improve the performance of a k-NN classifier by a remarkable margin. Lan *et al.* [25] introduced a new supervised weight factor called *relevance frequency* (rf) to replace the *idf* part. In a nutshell, rf for each term is the ratio of the number of documents in the positive category to the number of documents in the negative category containing the term. The *tf-rf* scheme performs consistently well for both the k-NN and SVM classifier over different data sets. The performance of *tf-idf* in this experiment was also comparably good.

2.3 Conclusion

This chapter has presented call routing applications and several feature weighting approaches used in the information retrieval and text categorization fields. As mentioned earlier, the focus of this work is on the classification part of an automatic call routing system which is a specific case of text categorization. The classification task is distinguished from the general text categorization by dealing with a collection of queries instead of documents due to the nature of incoming calls which are brief and short. This subtle difference makes applying the weight techniques to a call routing application difficult, whereas other techniques such as feature selection, dimensionality reduction, n-gram model, and etc. are still easily available. The reason of this difficulty is explained in detailed in Section 3.1.

To the best of the author's knowledge, none of the mentioned weighting schemes have been applied to the realm of the query classification problem nor specifically to the call routing applications. This work investigates an approach to incorporate weighting methods within the query classification task.

Chapter 3

System Architecture

The proposed approach in this thesis consists of using features' weighting to improve the accuracy of the query classification problem. This technique has been successfully applied before in the context of document classification; however, porting the weighting concept to the query classification problem is not an easy task. This work proposes a tailored algorithm to weight features in the context of query routing [2]. In this chapter, first the system architecture and proposed method is elaborated, and at the end, all the methods and algorithms used in this work are described.

3.1 Query Feature Weight Calculation

Tf-idf is the product of two parts. The Term Frequency (*tf*) part is the number of occurrences of a term in a given document. The Inverse Document Frequency (*idf*) part is obtained by dividing the total number of documents by the number of documents where the term is appeared and then taking the logarithm. (See Section 3.4.1 for more details). Given a set of documents and terms, it is straightforward to compute the *tf-idf* weights. However, the definition of the term frequency-inverse document frequency cannot be applied directly to a set of labeled queries.

To illustrate the problem, consider the query collection example shown in Table 3.1. In this example, there are fifteen queries labeled as either account, accountBalance, or accountNumber. If each query is treated as one document, the *tf* part becomes equal to one for almost all the keywords in the collection. This approach leaves *IDF* as the only factor impacting feature weighting. Table 3.2 displays the *tf-idf* weight values calculated in this way. Since the *tf* factor is one, the weights are global and category-independent. Note that weights for the words *number* and *balance* are relatively small, although these words play a very strong role for distinguishing the accountNumber and accountBalance categories respectively.

In this regard, in order to compute the weights, reorganizing the collection of queries is proposed. The idea is to cluster all queries with the same label (i.e., category) under a single set (i.e., document). In other words, the number of documents

Table 3.1: Query Examples

Query	Category
add money to account	account
can you top up my account balance	accountBalance
account number discussion please	accountNumber
i'd like to pay my account off	account
i wish to pay my account balance	accountBalance
i need my account number	accountNumber
i'd like to top up the account	account
i'd like to pay balance on my bill	accountBalance
what is my account number	accountNumber
adding money to my account	account
i would like to know my balance due	accountBalance
what's my bill customer number	accountNumber
i want to add money into my phone account	account
a balance inquiry	accountBalance
i needed help with my account number please	accountNumber

Table 3.2: *tf-idf* weight values when treating each query as a separate document

	tf	n	N	*idf*	*tf-idf*
account	1	11	15	0.13	0.13
add	1	3	15	0.7	0.7
balance	1	5	15	0.48	0.48
bill	1	2	15	0.88	0.88
money	1	3	15	0.7	0.7
number	1	5	15	0.48	0.48
pay	1	3	15	0.7	0.7
top up	1	2	15	0.88	0.88

Table 3.3: Reorganizing queries into documents

Document	Category
add money to account	
i'd like to pay my account off	
i'd like to top up the account	account
adding money to my account	
i want to add money into my phone account	
can you top up my account balance	
i wish to pay my account balance	
i'd like to pay balance on my bill	accountBalance
i would like to know my balance due	
a balance inquiry	
account number discussion please	
i need my account number	
what is my account number	accountNumber
what's my bill customer number	
i needed help with my account number please	

actually becomes equal to the number of categories. The *tf-idf* weighting scheme can now be applied to any labeled query collection, in which a document is defined as the set of queries labeled under the same category.

Table 3.3 depicts the queries after reorganizing queries into documents. In this case, there are three documents, one for each category, built out of queries under each category. Given these documents, *tf-idf* can easily be calculated for each word in a category. Table 3.4 presents new *tf-idf* values calculated by this approach. Note that the word *account* is now weightless, and the words *number* and *balance* are assigned the highest weight values as expected.

3.2 Query Feature Weight Estimation

Section 3.1 has detailed how to weight features or terms when a collection of labeled queries is available. However, in the case where a single query is given, it is impossible to compute the features' weights. For example, consider a query such as *adding money to my account*. From Table 3.4, one can easily get the *idf* part, however estimating the *tf* part is not an easy task. To estimate the *tf* factor, the approach explained in the previous section is not applicable, since the collection of testing queries is not accessible, and if it were, it would be impossible to cluster them into groups since the labels of the testing queries are not known in advance.

To overcome this issue, learning features' weight from a pre-weighted training

Table 3.4: *tf-idf* weight values after reorganizing queries into documents

		Category					
		account		accountBalance		accountNumber	
	idf	*tf*	*tf-idf*	*tf*	*tf-idf*	*tf*	*tf-idf*
account	0	5	0	2	0	4	0
add	0.477	3	1.43	0	0	0	0
balance	0.477	0	0	5	2.38	0	0
bill	0.18	0	0	1	0.18	1	0.18
money	0.477	3	1.43	0	0	0	0
number	0.477	0	0	0	0	5	2.38
pay	0.18	1	0.18	2	0.36	0	0
top up	0.18	1	0.18	1	0.18	0	0

query set is proposed. The idea is to learn a feature's weight based on the context from which the feature comes. To elaborate this procedure, consider the sample queries provided in Table 3.5, which displays certain keywords within different contexts. One can see that the word *copy* in items 1 to 4 has a higher weight value where it co-occurs with the word *bill*. The same observation is valid for the words *final*, *one*, and *charge,* in items 5 to 18. The word *assistance* has also been weighted higher, where the word *directory* happens to be around for items 18 to 24. Based on this observation, it must be feasible to estimate the weight of a word according to its context in a query.

Now the question is how to estimate the weight of a word based on the context. The assumption is that there is a generative model for each word which generates the weight according to some input. Here, the context where a word comes from, is considered as the input to this model. Thus, the input to each model is a couple of words which are represented in a vector space model and the output is the weight which is a real number.

This description of the problem leads to use regression modeling to estimate the weight generator model. This model will interpolate the computed weights during the training to the testing queries. In other words, the relation between the weights in a query and their features is going to be learned. Algorithm 3.1 explains the steps needed to prepare data for training the regression models. For each word or feature f_j, there is a regression model r_j estimating the weight with regards to the input query. The training input for model r_j would be a set of queries Q_j containing word or feature f_j and the corresponding output would be the pre-calculated *tf-idf* weight $\alpha_{j,q}$ where $q \in Q_j$. To explain how to prepare the input and output data for each regression model, consider the sample queries introduced in Table 3.5. Table 3.6 shows the input and output training data for the regression model estimating weight in the case of the word *charge*. In this case, all the queries in Table 3.5 containing the word *charge* and the corresponding weight values are taken as training data. An RBF Support Vector Regression(SVR) machine from

Table 3.5: Sample of 2511 queries where words in different context are assigned different weights. Note that stopwords have been removed and all words are shown in the root form. The *tf-idf* weights are written inside the parentheses.

1. bill(0.13) **copy(0.47)**
2. need(0.05) **copy(0.47)** past(0.09) monthly(0.05) bill(0.13)
3. need(0.05) **copy(0.47)** last(0.07) bill(0.13)
4. need(0.05) get(0.04) **copy(0.22)** call(0.04) electronic(0.44)
5. **final(0.93)** bill(0.09)
6. know(0.04) **final(0.93)** bill(0.09)
7. **final(0.1)** payment(0.26)
8. **one(0.03)** payment(0.26)
9. **one(0.03)** time(0.04) credit(0.05) top(0.1)
10. **one(0.12)** bill(0.13)
11. find(0.04) **one(0.12)** bill(0.13)
12. **charge(0.51)** bill(0.08)
13. current(0.1) **charge(0.51)**
14. understand(0.11) current(0.1) **charge(0.51)** know(0.04)
15. **charge(0.51)** last(0.06) bill(0.08) understand(0.11) speak(0.05)
16. phone(0.05) get(0.03) **charge(0.08)** call(0.03) one(0.17)
17. help(0.03) payment(0.26) **charge(0.05)**
18. directory(0.41) **assistance(0.27) charge(0.08)**
19. account(0.1) balance(0.46) **assistance(0.08)**
20. need(0.01) **assistance(0.08)** bill(0.13)
21. **assistance(0.08)** message(0.16)
22. directory(0.41) **assistance(0.27)**
23. fee(0.08) pay(0.02) directory(0.41) **assistance(0.27)**
24. need(0.04) directory(0.41) **assistance(0.27)**

LIBSVM [9] has been used for regression modeling.

Algorithm 3.1: Data preparation and training regression models

Input:
$Q = \{q_1, \ldots, q_N\}$: Collection of N training queries
$F = \{f_1, \ldots, f_K\}$: Feature space where each word in a training query can be a feature
$q = \{f_{1_q}, \ldots, f_{M_q}\}$: Each query q consists of M_q features where $M_q \leq K$
$w_q = \{\alpha_{1_q}, \ldots, \alpha_{M_q}\}$: Weights corresponding to the features of query q
(x, y): A pair of training data where x is the input and y is the output
Output:
$R = \{r_1, \ldots, r_K\}$: Set of regression models. Each regression model r_j correspons with feature f_j where $j = 1, \ldots, K$.
begin
    `// preparing training data for regression models`
    foreach $q \in Q$ **do**
        foreach $f_{j_q} \in q$ **do**
            add (q, α_{j_q}) to the training set of r_{j_q}
        end
    end
    `// training regression models`
    foreach $r_j \in R$ **do**
        train r_j using a regression algorithm such as SVR
    end
end

Table 3.6: Input and output data to train the regression model for estimating weight for word *charge*.

Input data	Target Weight
charge bill	0.51
current charge	0.51
understand current charge know	0.51
charge last bill understand speak	0.51
phone get charge call one	0.08
help payment charge	0.05
directory assistance charge	0.08

Once all the models are trained, they can be used for estimating feature weights. Algorithm 3.2 describes how to use the regression models to estimate the weights. Given a query q containing M features, there is a regression model r_j for each feature f_j which estimates the weight for the jth feature of q. As an example, consider the

input query *need copy bill*. This query will be fed to the three regression models responsible for *need*, *copy*, and *bill*. The generated weight values will be something like 0.05, 0.47, and 0.13, respectively, for *need*, *copy*, and *bill*.

Algorithm 3.2: Estimating weights using regression models

Input:
$F = \{f_1, \ldots, f_K\}$: Feature space
$q = \{f_{1_q}, \ldots, f_{M_q}\}$: Input query q consists of M_q features where $M_q \leq K$
$R = \{r_1, \ldots, r_K\}$: Set of trained regression models. Each regression model r_j corresponds to a feature f_j where $j = 1, \ldots, K$.
Output:
$w_q = \{\alpha_{1_q}, \ldots, \alpha_{M_q}\}$: Weights corresponding to the features of query q
begin
  foreach $f_{j_q} \in q$ **do**
    $\alpha_{j_q} = r_{j_q}(q)$
  end
end

3.3 Query Classification

After weighting the training query collection, and building the regression models (to weight single queries), the query classification procedure is carried out. The training procedure is detailed in Figure 3.1. Given a collection of queries, the first step is to extract the most important words in the collection (features extraction). This step includes the elimination of all stop words and stemming. Once the set of features has been identified, the features' weighting procedure is carried out, as described in Section 3.2. Each query is fed to a set of regression models corresponding with the query features. This process results in a weighted set of feature vectors similar to the one depicted in Table 3.5. The final training step consists of feeding these updated feature vectors to a classifier for training. Note that the classifier is trained with the weighted feature vectors, which are estimated by the regression models, not the weights calculated from the approach described in Section 3.1.

Once a classifier is trained using the weighted query collection set, the testing is carried out as shown in Figure 3.2. Note that weightings are now estimated by the regression models. Given a query, similar steps are carried out for pre-processing including stopwords removal and stemming. The next step is to select the features. Features will then be weighted using the regression models trained earlier. The weighted feature vector is then fed to the already trained classifier for routing purposes.

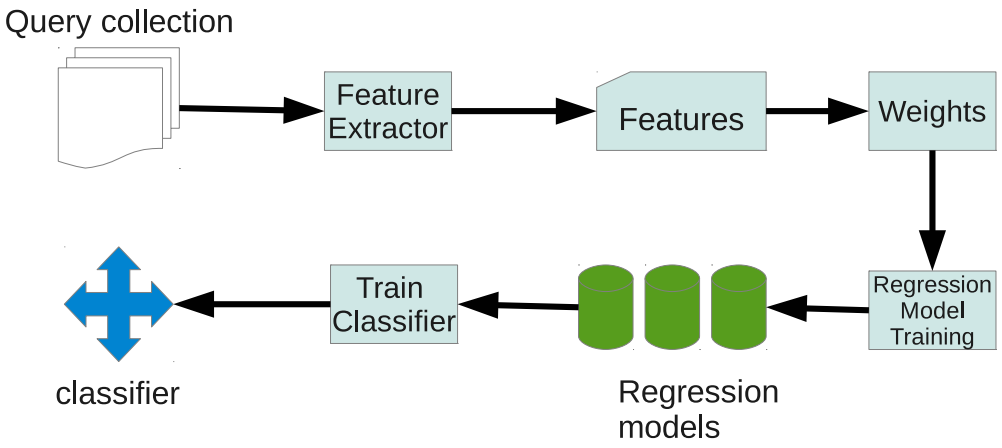

Figure 3.1: Training procedure for query classification

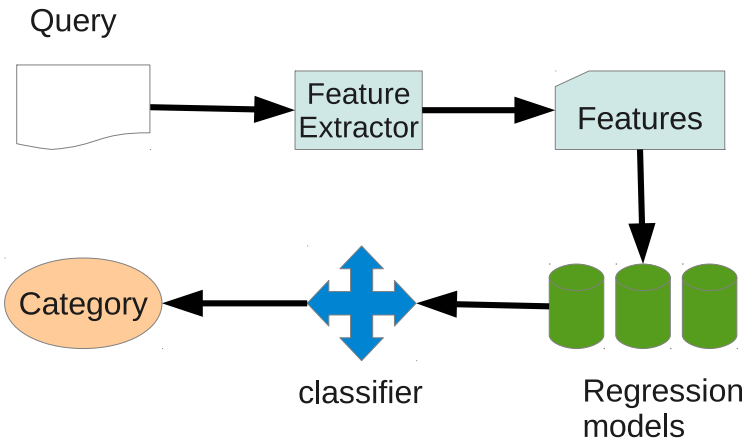

Figure 3.2: Testing procedure for query classification

3.4 Methods and Algorithms

This section presents a brief description of the methods and algorithms used in this work. *Tf-idf* is the weighting scheme of choice. Information Gain (IG) is used for feature selection. For query classification task, four different classifiers are chosen including Support Vector Machines (SVM), Naive Bays, Maximum Entropy, and k-Nearest Neighbor (k-NN). To learn the weights, Support Vector Regression is selected as the regression algorithm and Precision, Recall, and F1-measure are discussed as performance measures.

3.4.1 TF-IDF Weighting Scheme

The term frequency-inverse document frequency, *tf-idf*, is the weighting scheme considered in this work. It is worth mentioning that the proposed algorithm is generic and independent of the weighting mechanism.

The *tf-idf* is a commonly used numerical statistic in the field of information retrieval. In a nutshell, this metric reflects how important a word is to a document in a collection of documents. It is a product of the term frequency and the inverse document frequency. The term frequency is simply the number of occurrences of a term in a given document. The inverse document frequency of the term t in the corpus D is given by equation

$$idf(t, D) = log\frac{N}{n} \tag{3.1}$$

where N is the total number of documents in the collection and n is the number of documents to which term t belongs.

Then *tf-idf* calculation of term t within document d is

$$\textit{tf-idf}(t, d, D) = tf(t, d) * idf(t, D). \tag{3.2}$$

3.4.2 Feature Selection

A text categorization task could easily consist of tens or even hundreds of thousands of features. Feature selection is one of the means to reduce the number of features, since most of the learning algorithms in text classifications are not able to deal with all features effectively. Automatic feature selection is done by selecting the most informative features, according to the relation between features and the categories in the training data. In this work, Information Gain has been selected as the feature selection method of choice, since it has been reported in machine learning as one of the best examples of a "term-goodness criterion" [45, 13, 14].

Information Gain measures how many bits on average would be saved, if a category were going to be transmitted by knowing the presence or absence of a term in a document. In other words, the term with the highest number of saved bits is the most important term.

Assuming there are m categories $\{c_i\}_{i=1}^{m}$, the information gain of term t is defined as [45]:

$$\begin{aligned} IG(t) &= -\sum_{i=1}^{m} P(c_i)\log_2 P(c_i) \\ &\quad +P(t)\sum_{i=1}^{m} P(c_i \mid t)\log_2 P(c_i \mid t) \\ &\quad +P(\bar{t})\sum_{i=1}^{m} P(c_i \mid \bar{t})\log_2 P(c_i \mid \bar{t}) \end{aligned} \tag{3.3}$$

where $\bar{t}$ means the absence of term t, $P(c_i)$ is the probability of having ith category, $P(c_i \mid t)$ is the conditional probability of having ith category given term t, and $P(c_i \mid \bar{t})$ is the conditional probability of having ith category given the absence of term t. Equation (3.3) measures the information gain of term t globally with respect to all categories.

3.4.3 Naive Bayes (NB)

A Naive Bayes (NB) classifier is a simple probabilistic classifier. It is based on Bayes' theorem. All features are assumed to be independent. In other words, given the class variable, the presence (or absence) of a particular feature of a class has no relation to the presence (or absence) of any other feature. Despite that, these assumptions are rarely true in the real world, naive Bayes classifiers have proved to be competitive in solving complex problems.

In abstract, the probability of having class c given document d is

$$P(c \mid d) \propto P(c)\prod_{i=1}^{n_d} P(w_i \mid c) \tag{3.4}$$

where $P(w_i|c)$ is the conditional probability of occurring word w_i in a document of class c. $P(c)$ is the prior probability of having a document in class c. The best class representing document d is then the most likely or *maximum a posteriori (MAP)* class c_{MAP}:

$$c_{MAP} = \arg\max_c \hat{P}(c \mid d) = \arg\max_c \hat{P}(c)\prod_{i=1}^{n_d} \hat{P}(w_i \mid c) \tag{3.5}$$

where $\hat{P}$ is the estimation of P from training data. In Equation (3.5), many probabilities are multiplied; this can result in calculation instability. Thus, it is a common practice to take the logarithm of Equation (3.5). Since logarithm is a monotonic function, it results:

$$c_{MAP} = \arg\max_c \left[\ln \hat{P}(c) + \sum_{i=1}^{n_d} \ln \hat{P}(w_i \mid c)\right] \tag{3.6}$$

The prior is estimated as:

$$\hat{P}(c) = \frac{N_c}{N} \tag{3.7}$$

where N_c is the total number of documents in class c and N is the total number of documents. The estimation for conditional probabilities is

$$\hat{P}(w_i \mid c) = \frac{T_{cw_i}}{\sum\limits_{w_j \in V} T_{cw_j}} \tag{3.8}$$

where $T_{cw_i}$is the number of occurrences of word w_i in class c and V is the vocabulary. One problem with the estimation of conditional probabilities in Equation (3.8) is that it becomes zero if the frequency of word w_i in class c is zero. Since the conditional probabilities are being multiplied, $\hat{P}(c \mid d)$ in Equation (3.4) becomes zero. To avoid this zero problem, smoothing techniques such as add-one or Laplace smoothing, which increases the frequency of each word by one, can be used:

$$\hat{P}(w_i \mid c) = \frac{T_{cw_i} + 1}{\sum\limits_{w_j \in V} (T_{cw_j} + 1)} = \frac{T_{cw_i} + 1}{\sum\limits_{w_j \in V} (T_{cw_j}) + B} \tag{3.9}$$

where $B = |V|$ is the number of words in the vocabulary.

3.4.4 Maximum Entropy or Multinomial Logistic Regression (MaxEnt)

Maximum Entropy (MaxEnt) is a general-purpose machine learning technique that produces the least biased estimate possible based on the given information. It assumes no conditional independence between features, unlike the Naive Bayes classifier. The idea behind this classifier is to "model all that is known and assume nothing about that which is unknown"[5]. To achieve this, only those classifiers are taken into consideration which are empirically consistent with a set of training data, and maximizing entropy. The estimation of how the decision of a classifier is predictable is provided by the classifier's entropy. A classifier with a higher entropy behaves more randomly. For example, a classifier with zero entropy always classifies inputs into the same label. On the other hand, a classifier with a very high entropy, classifies inputs randomly.

Maximum entropy's estimate of $P(y)$, where there are $m + 1$ categories with category 0 being the reference category, takes the following exponential form:

$$\begin{aligned} P(y = j) &= \frac{\exp(\boldsymbol{\beta}_j.\mathbf{x})}{1 + \sum\limits_j \exp(\boldsymbol{\beta}_j.\mathbf{x})} \qquad j = 1, 2, \ldots, m \\ P(y = 0) &= \frac{1}{1 + \sum\limits_j \exp(\boldsymbol{\beta}_j.\mathbf{x})} \end{aligned} \tag{3.10}$$

where $\mathbf{x}$ is the input vector, $\boldsymbol{\beta}$ is the weight vector and $P(y = j)$ is the probability of having jth category. The solution for $\boldsymbol{\beta}$ is usually an iterative procedure such as Iteratively Re-weighted Least Squares (IRLS), or a quasi-Newton method such as the L-BFGS method. The explanation of these methods is outside the scope of this work. To read more see *Regression Models for Categorical and Limited Dependent Variables* [26].

A single layer neural network with logistic function as the activation function is identical to a maximum entropy or logistic regression model. This relation is more obvious if Equation (3.10) is rewritten as

$$\ln\left(\frac{P(y=j)}{P(y=0)}\right) = \boldsymbol{\beta}_j.\mathbf{x} \tag{3.11}$$

There are other aliases to multinomial logistic regression in the literature, which is mentioned here for the sake of clarity:

- Polytomous Logistic Regression.

  Multinomial logistic regression is also known as polytomous, polychotomous, or multi-class logistic regression, or just multilogit regression.

- Ridge Regression and the Lasso.

- Shrinkage and Regularized Regression.

- Generalized Linear Model and Softmax.

3.4.5 *k*-Nearest Neighbor (*k*-NN)

is one of the simplest and most popular classification techniques. It can be used for both classification and regression purposes. It is a non-parametric lazy learning algorithm that makes no assumptions about the underlying distribution of the training data. Laziness means there is no training phase, and it does no generalization base on the training data. For classification, k-NN assigns a new point to the class which has the majority among its k nearest neighbors. This means that all the training data are needed in the testing phase, and this makes k-NN computationally expensive. This naive method takes a time complexity of $O(dn)$ where d and n are the dimension and the number of data respectively. To reduce the time complexity, some methods have been proposed such as KD-tree indexing [4]. There are different distance metrics to measure the distance between a point and their neighbors, such as Euclidean, Manhattan, and Cosine distance, although Euclidean distance is the most commonly used metric. The selection of parameter k is critical. A small value of k makes the result very sensitive to noise, while on the other hand, a large value of k makes it computationally expensive, and classes with a large number of data values tend to dominate the result. The best value for k is usually selected by techniques like cross-validation [15]. To enhance the classification result, it can be useful to weight the contribution made by each neighbor, so

that the nearer neighbors have bigger weights and contribute more than the distant ones. One of the common weighting schemes is to assign weight to each neighbor according to its similarity, or the inverse of its distance.

In this work, the weighted k-NN with a cosine similarity metric [35] is used:

$$\text{sim}(\mathbf{x}, \mathbf{y}) = \cos(\theta) = \frac{\mathbf{x}.\mathbf{y}}{\| \mathbf{x} \| \| \mathbf{y} \|}. \tag{3.12}$$

$$f(\mathbf{x}_i, c_k) = \begin{cases} 0 & \text{if } \mathbf{x}_i \notin c_k \\ 1 & \text{if } \mathbf{x}_i \in c_k \end{cases} \tag{3.13}$$

$$y(\mathbf{t}_j) = \arg\max_k \sum_{x_i \in k\text{nn}} \text{sim}(\mathbf{t}_j, \mathbf{x}_i) f(\mathbf{x}_i, c_k) \tag{3.14}$$

where $\mathbf{x}_i$ and c_k are training data and classes respectively, and $\mathbf{t}_j$ is the test data.

3.4.6 Support Vector Machine (SVM)[1]

Support Vector Machine (SVM) is a new generation of learning systems that has been developed and introduced by V. Vapnik [40] for classification and pattern recognition based on supervised learning and statistical theory. Like neural networks and fuzzy systems, SVM is a typical non-parametric classifier, meaning that no primary knowledge is assumed for tackling the pattern classification problem [1]. SVM theory was originally developed to tackle the separation of two series of data points (binary separation).

3.4.6.1 Linear SVM

Suppose that one wants to classify some data points into two classes. For N training data points:

$$\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N) \mid \mathbf{x}_i \in \mathbb{R}^n, y_i \in \{-1, 1\}\}_{i=1}^N \tag{3.15}$$

the task is to find a linearly separable hyperplane

$$H_0 : \mathbf{w}^T \cdot \mathbf{x} + b = 0 \tag{3.16}$$

where $\mathbf{w}$ is the weight vector and b is the bias. Hyperplane H_0 should have the maximum separating margin (see Figure 3.3). Since the aim is maximizing the margin, and the data are linearly separable, $\mathbf{w}$ and b can be chosen to maximize the distance between two hyperplanes parallel to H. These parallel hyperplanes can be described as

$$H_1 : \mathbf{w}^T \cdot \mathbf{x} + b = 1 \tag{3.17}$$

$$H_2 : \mathbf{w}^T \cdot \mathbf{x} + b = -1 \tag{3.18}$$

[1] To see the full explanation and complete details, see Burgs tutorial [7].

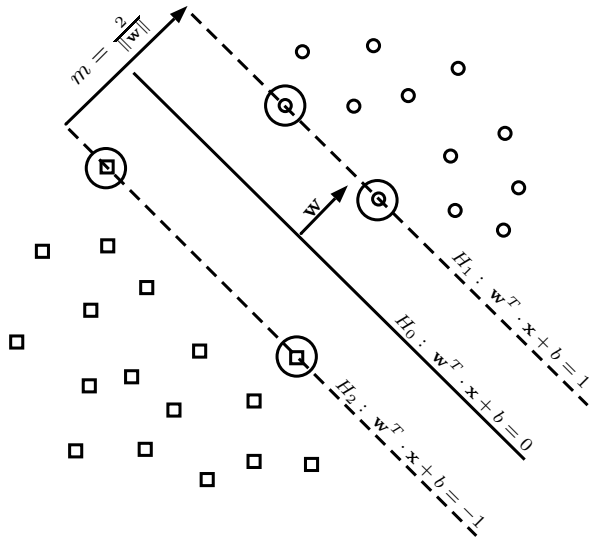

Figure 3.3: Illustration of maximum separation. Theoretically, the best line is the line that maximizes the margin m where there are no data points between H_1 and H_2.

or

$$y_i(\mathbf{w}^T \cdot \mathbf{x}_i + b) = 1. \tag{3.19}$$

The distance between H_1 and H_2 is $\frac{2}{\|\mathbf{w}\|}$. To maximize the distance, one can minimize $\| \mathbf{w} \|^2$. Note the constraint that there should be no data points between H_1 and H_2. Thus the problem can be written as

$$\min \frac{1}{2} \| \mathbf{w} \|^2 \tag{3.20}$$

subject to

$$y_i(\mathbf{w}^T \cdot \mathbf{x}_i + b) \geq 1 \text{ for all data points.} \tag{3.21}$$

This is a convex, quadratic programming problem, in a convex set. Using Lagrange multipliers $\alpha_i \geq 0$, results the following:

$$L(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2} \| \mathbf{w} \|^2 - \sum_{i=1}^{N} \alpha_i y_i(\mathbf{w}^T \cdot \mathbf{x}_i + b) + \sum_{i=1}^{N} \alpha_i. \tag{3.22}$$

where L is the . This problem can be solved by standard quadratic programming. The superiority of SVM comes from this specific formulation of a convex objective function with constraints. Since the function is solved using Lagrange multipliers, it guarantees the following:

1. A global optimal solution exists that will be found.

2. The result is a general solution avoiding over-training.

3. The solution is sparse and only a limited set of training points contribute to this solution.

4. A non-linear solution can be calculated efficiently due to the using of inner products.

3.4.6.2 The Dual Problem

Equation (3.22) could be solved as a Wolfe dual problem as well:

$$\frac{\partial L}{\partial \mathbf{w}} = 0 \tag{3.23}$$

$$\frac{\partial L}{\partial b} = 0 \tag{3.24}$$

$$\alpha_i \geq 0 \tag{3.25}$$

Here, $\mathbf{w}$ and b are primal variables and the gradient of L with respect to primal variables need to be eliminated. Equations (3.23) and (3.24) result:

$$\mathbf{w} = \sum \alpha_i x_i y_i \tag{3.26}$$

and

$$\sum_{i=1}^{N} \alpha_i y_i = 0. \tag{3.27}$$

Only data points whose α_i is greater than zero contribute to the solution. These data points are *support vectors*. The dual form is then obtained by substituting (3.26) and (3.27) into (3.22):

$$L_D \equiv \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j) \tag{3.28}$$

in which the primal variables are vanished. Note that threshold b is not determined by training procedure, but can be easily calculated by using Equation (3.19) for any i which $\alpha_i \neq 0$. One can observe that for each $\alpha_i \neq 0$, there will be a different threshold, thus in practice, it is more accurate to take the average of all possible b over all support vectors:

$$b = \frac{1}{N_{SV}} \sum_{i=1}^{N_{SV}} (\mathbf{w} \cdot \mathbf{x}_i - y_i) \tag{3.29}$$

where N_{SV} is the number of support vectors. Replacing the newly evaluated value for $\mathbf{w}$ in the initial linear separating hyperplane Equation (3.16) results in the following:

$$H_0 : \sum_{i=1}^{N} \alpha_i y_i (\mathbf{x}_i \cdot \mathbf{x}) + b \tag{3.30}$$

Note the training vectors $\mathbf{x}_i$ occur only in the form of dot product, both in the objective function, Equation (3.28), and the solution, Equation (3.30).

3.4.6.3 Non-linear SVM

In many cases the surface separating the two classes is not linear. In 1992, B. Boser, I. Guyon and Vapnik [6] suggested a way to create non-linear classifiers by applying the kernel trick. The trick for separating these classes is transformation of the data points to another high-dimensional space such that the data points will be linearly separable. Assuming the transformation function is $\boldsymbol{\phi}$, the Lagrange equation can be rewritten for such a case as:

$$L_D \equiv \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j (\boldsymbol{\phi}(\mathbf{x}_i) \cdot \boldsymbol{\phi}(\mathbf{x}_j)) \tag{3.31}$$

where

$$\boldsymbol{\phi}(\mathbf{x}_i).\boldsymbol{\phi}(\mathbf{x}_j) = k(\mathbf{x}_i, \mathbf{x}_j). \tag{3.32}$$

$k(\mathbf{x}_i, \mathbf{x}_j)$ is a kernel function of the input space and equivalent to the dot product of that high-dimensional space. This implies that transformation function $\boldsymbol{\phi}$ does not need to be explicit. There are many kernel function formats that can be used in the above equation, such as linear, Radial Basis Function (Gaussian Kernel), and polynomial:

$$k_{Linear}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i.\mathbf{x}_j \tag{3.33}$$

$$k_{RBF}(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\| \mathbf{x}_i - \mathbf{x}_j \|^2}{2\sigma^2}\right) \tag{3.34}$$

$$k_{Poly}(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i.\mathbf{x}_j + c)^d \tag{3.35}$$

3.4.6.4 Imperfect separation

Note that so far a linearly separable case is investigated. In practice, there is a large overlap of the classes due to noise or bad labeling, so there are two solutions. One is to use a powerful kernel, which is not suitable because it ends up over-fitting. The other solution is to introduce positive slack variables ξ_i , $i = 1, \ldots, l$ in the constraints to relax the hard margin constraints [10]. The slack variables permit some examples to violate the constraints. One possible form for incorporating slack variables in the objective function is

$$\min \frac{1}{2} \| \mathbf{w} \|^2 + C \sum_i \xi_i \tag{3.36}$$

subject to

$$y_i(\mathbf{w}^T \cdot \mathbf{x}_i + b) \geq 1 - \xi_i \text{ for all data points,} \tag{3.37}$$

$$\xi_i \geq 0. \tag{3.38}$$

The parameter C is a term to penalize the misclassified points. The parameter C can also be considered a trade-off between the complexity term and the empirical

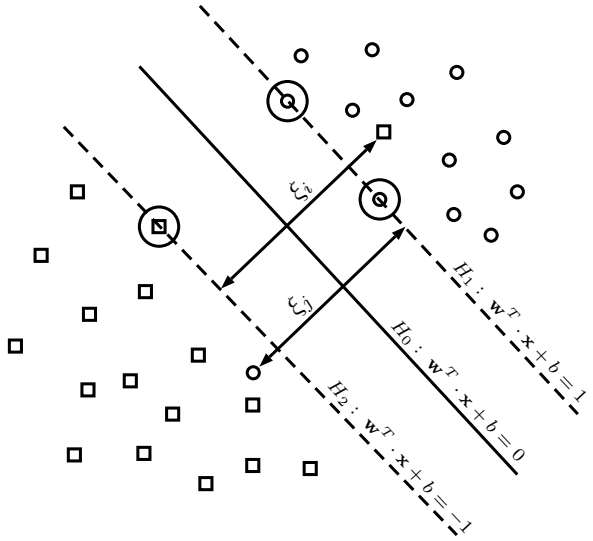

Figure 3.4: Slack variables for non-separable case

error [28]. The dual form becomes

$$L_D \equiv \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j k(\mathbf{x}_i \cdot \mathbf{x}_j) \tag{3.39}$$

subject to

$$0 \leq \alpha_i \leq C \tag{3.40}$$

$$\sum_i \alpha_i y_i \geq 0. \tag{3.41}$$

The difference from the perfectly separable case is that α_i is now bounded by C. The linear separation for the non-separable case is shown in Figure 3.4.

3.4.6.5 Multiclass SVM

The SVM classifier is a binary classifier. Due to various complexities, a direct solution of multiclass problems using a single SVM is usually avoided. To solve a multiclass problem, a combination of several binary SVM classifiers is used. Here, some popular methods for the combination of binary classifiers are briefly described:

1. **One-against-all or Winner-Takes-All**: Considering an M class problem, one can construct M classifiers which separate one class from the others. In this method, there are some areas which remain unassigned or assigned by several classes.

2. **Pairwise or Max-Wins-Voting**: This method builds one binary classifier for every pair of distinct classes. Therefore, M(M-1)/2 binary classifiers will

finally be constructed. Each of those M(M-1)/2 classifiers makes its own vote to a new example. Pairwise strategy assigns the example to the class with the largest number of votes.

A fuzzy version of both the above methods has been also developed. For more discussion see [1].

3.4.7 Support Vector Regression (SVR)[2]

There exist different variations of Support Vector Regression (SVR). Here ε-SVR [41] is explained which is the most commonly used version. Given a set of data

$$\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N) \mid \mathbf{x}_i \in \mathbb{R}^n, y_i \in \mathbb{R}\}_{i=1}^N, \tag{3.42}$$

the goal is to find a function $f(\mathbf{x})$ which should meet two criteria. First, the maximum deviation of $f(\mathbf{x})$ from actual training data must be at most equal to ε for all training data. Second, the solution must be as flat as possible. In other words, the maximum error tolerated is ε.

3.4.7.1 Linear SVR

Consider the case where $f(\mathbf{x})$ takes a linear function

$$f(\mathbf{x}_i) = \mathbf{w}^T.\mathbf{x}_i + b. \tag{3.43}$$

To make sure that the flatness condition is met, $\mathbf{w}$ needs to be small. One way to achieve this is to minimize $\| \mathbf{w} \|^2$ obtaining a convex optimization problem:

$$\min \frac{1}{2} \| \mathbf{w} \|^2 \tag{3.44}$$

subject to

$$\begin{aligned} y_i - \mathbf{w}^T.\mathbf{x}_i - b &\leq \varepsilon \\ \mathbf{w}^T.\mathbf{x}_i + b - y_i &\leq \varepsilon. \end{aligned} \tag{3.45}$$

Equations (3.44) and (3.45) imply that such a function f which can approximate all data with ε precision exists. However, this is not always the case. In practice, one need to relax the first criterion. Similar to SVM (Section 3.4.6.4), one can introduce slack variables ξ_i and ξ_i^* to deal with an infeasible approximation problem. So Equation (3.44) can be rewritten as

$$\min \frac{1}{2} \| w \|^2 + C \sum_{i=1}^{N} (\xi_i + \xi_i^*) \tag{3.46}$$

[2]To see more discussion about SVR, see Smola *et al.* tutorial [36].

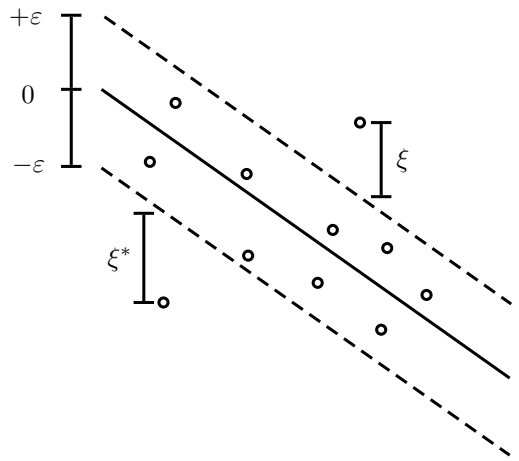

Figure 3.5: Slack variables have been introduced to cope with an impractical approximation problem

subject to

$$\begin{aligned} y_i - \mathbf{w}^T.\mathbf{x}_i - b &\leq \varepsilon + \xi_i \\ \mathbf{w}^T.\mathbf{x}_i + b - y_i &\leq \varepsilon + \xi_i^* \\ \xi_i^{(*)} &\geq 0 \end{aligned} \tag{3.47}$$

Note that notation $A^{(*)}$ refers to both A and A^*. The C parameter is the trade-off between the flatness and the tolerance of having a deviation larger than ε. Figure 3.5 displays the situation. Only points outside the tube contribute to the penalty term. The quadratic problem (Equations (3.46) and (3.47)) can be solved using Lagrange multipliers:

$$\begin{aligned} L &= \frac{1}{2} \parallel \mathbf{w} \parallel^2 + C \sum_{i=1}^{N} (\xi_i + \xi_i^*) \\ &- \sum_{i=1}^{N} (\eta_i \xi_i + \eta_i^* \xi_i^*) \\ &- \sum_{i=1}^{N} \alpha_i (\varepsilon + \xi_i - y_i + \mathbf{w}^T.\mathbf{x}_i + b) \\ &- \sum_{i=1}^{N} \alpha_i^* (\varepsilon + \xi_i^* + y_i - \mathbf{w}^T.\mathbf{x}_i - b) \end{aligned} \tag{3.48}$$

where

$$\alpha_i^{(*)}, \eta_i^{(*)} \geq 0. \tag{3.49}$$

3.4.7.2 The Dual Problem

Equation (3.48) can be rewritten in the form of a dual problem by taking the derivatives of L with respect to the prime variables which are $\mathbf{w}, b, \xi_i^{(*)}$.

$$\frac{\partial L}{\partial \mathbf{w}} = \sum_{i=1}^{N}(\alpha_i^* - \alpha_i) = 0 \tag{3.50}$$

$$\frac{\partial L}{\partial b} = \mathbf{w} - \sum_{i=1}^{N}(\alpha_i - \alpha_i^*)\mathbf{x}_i = 0 \tag{3.51}$$

$$\frac{\partial L}{\partial \xi_i^{(*)}} = C - \alpha_i^{(*)} - \eta_i^{(*)} = 0 \tag{3.52}$$

The dual form is then obtained by substituting (3.50), (3.51), and (3.52) into (3.48):

$$\begin{aligned} \max \quad & -\frac{1}{2}\sum_{i,j=1}^{N}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)\mathbf{x}_i.\mathbf{x}_j \\ & -\varepsilon\sum_{i=1}^{N}(\alpha_i + \alpha_i^*) + y_i\sum_{i=1}^{N}(\alpha_i - \alpha_i^*) \end{aligned} \tag{3.53}$$

subject to

$$\sum_{i=1}^{N}(\alpha_i - \alpha_i^*) = 0$$

$$0 \le \alpha_i^{(*)} \le C. \tag{3.54}$$

Note that $\eta_i^{(*)}$ has disappeared from the dual form. Equation (3.51) results in the following:

$$\mathbf{w} = \sum_{i=1}^{N}(\alpha_i - \alpha_i^*)\mathbf{x}_i \tag{3.55}$$

thus

$$f(\mathbf{x}) = \sum_{i=1}^{N}(\alpha_i - \alpha_i^*)\mathbf{x}_i.\mathbf{x} + b. \tag{3.56}$$

The solution for $\mathbf{w}$ is a linear combination of the training data $\mathbf{x}_i$, and therefore the complexity of the solution depends only on the *Support Vectors*, and is independent of the dimensionality of the input data. One can observe that for evaluating f, there is no need to calculate $\mathbf{w}$ explicitly, since the solution of f is described in terms of the dot product between the data.

So far, equations (3.50), (3.51), and (3.52) do not provide any means to calculate b. Since Karush-Kahn-Trucker conditions [21, 23] have to be satisfied by the dual problem, it is possible to calculate the lower and upper bound of b. It can be shown [36] that b boundaries are:

$$\begin{aligned} &\max\{-\varepsilon + y_i - \mathbf{w}.\mathbf{x}_i \mid \alpha_i < C \text{ or } \alpha_i^* > 0\} \le b \le \\ &\min\{-\varepsilon + y_i - \mathbf{w}.\mathbf{x}_i \mid \alpha_i > 0 \text{ or } \alpha_i^* < C\}. \end{aligned} \tag{3.57}$$

One option for choosing b is to take the midpoint of the preceding range. For alternative ways to choose b, see [22].

3.4.7.3 Non-linear SVR

From Equation (3.56), it is obvious that this solution can only be applied to the linear cases. In other words, this solution is capable of modeling the training data with an acceptable error, if the data has linear behavior. Analogous to the non-linear SVM (Section 3.4.6.3), it is possible to overcome this problem by means of kernel functions. Knowing that there exists such a function $k(\mathbf{x}_i, \mathbf{x}_j) = \boldsymbol{\phi}(\mathbf{x}_i).\boldsymbol{\phi}(\mathbf{x}_j)$ where $\boldsymbol{\phi}$ transfers a point to the feature space with higher dimension, Equation (3.53) can be written as follows:

$$\begin{aligned} \max \quad & -\frac{1}{2}\sum_{i,j=1}^{N}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)k(\mathbf{x}_i.\mathbf{x}_j) \\ & -\varepsilon\sum_{i=1}^{N}(\alpha_i + \alpha_i^*) + y_i\sum_{i=1}^{N}(\alpha_i - \alpha_i^*) \end{aligned} \tag{3.58}$$

subject to the following:

$$\sum_{i=1}^{N}(\alpha_i - \alpha_i^*) = 0 \tag{3.59}$$

$$0 \leq \alpha_i^{(*)} \leq C.$$

The solution for f becomes:

$$\mathbf{w} = \sum_{i=1}^{N}(\alpha_i - \alpha_i^*)\boldsymbol{\phi}(\mathbf{x}_i) \tag{3.60}$$

$$f(\mathbf{x}) = \sum_{i=1}^{N}(\alpha_i - \alpha_i^*)k(\mathbf{x}_i.\mathbf{x}) + b. \tag{3.61}$$

Contrary to the linear case, it is not possible to calculate $\mathbf{w}$ explicitly.

3.4.8 Performance Measures

There are several measures to evaluate the effectiveness of a classifier, such as error, accuracy, recall, and precision [34]. Each measure describes one aspect of a classifier performance. Thus, to have a perfect view of a classifier performance, one needs to consider a few of them, or even a combination of measures. This chapter explains some of these measures and some commonly used methods to combine them.

3.4.8.1 Precision and recall

Precision (π) and recall (ρ) are two standard methods for measuring the effectiveness of a classifier, adapted from Information Retrieval (IR) measures into the text categorization field. Precision(π) for class C is the number of elements correctly labeled as C by the classifier (True Positive), divided by the number of items labeled as C by the classifier. On the other hand, ρ for class C is the number of items

Table 3.7: The contingency table for category c_i

classifier		expert	
c_i		+	-
classifier	+	TP_i	FP_i
	-	FN_i	TN_i

correctly labeled as C by the classifier, divided by the number of elements labeled as C by the expert. If precision (π) is 1.0, it means all the instances labeled as C indeed belong to C, but it says nothing about whether all the documents related to category C are retrieved. Recall (ρ) equal to 1.0 means all documents belonging to C have been successfully retrieved, but says nothing as to whether other documents not belonging to C have been categorized as C. Precision (π) and recall (ρ) for class c_i , $i = 1, \ldots, N$ can be estimated in terms of a contingency table (Table 3.7) where TP_i, TN_i, FP_i, and FN_i are true positive, true negative, false positive, and false negative respectively. The precision and recall then are defined as :

$$\pi_i = \frac{\mathrm{TP}_i}{\mathrm{TP}_i + \mathrm{FP}_i} \tag{3.62}$$

$$\rho_i = \frac{\mathrm{TP}_i}{\mathrm{TP}_i + \mathrm{FN}_i}. \tag{3.63}$$

To obtain π and ρ, two different methods exist:

1. *micro-averaging*: π and ρ are estimated globally by summing over all elements of the contingency table:

$$\pi^{\mu} = \frac{\sum_{i=1}^{N} \mathrm{TP}_i}{\sum_{i=1}^{N} (\mathrm{TP}_i + \mathrm{FP}_i)} \tag{3.64}$$

$$\rho^{\mu} = \frac{\sum_{i=1}^{N} \mathrm{TP}_i}{\sum_{i=1}^{N} (\mathrm{TP}_i + \mathrm{FN}_i)}. \tag{3.65}$$

2. *macro-averaging*: π and ρ are estimated globally by averaging over different precision and recall for each category:

$$\pi^{M} = \frac{\sum_{i=1}^{N} \pi_i}{N} \tag{3.66}$$

$$\rho^{M} = \frac{\sum_{i=1}^{N} \rho_i}{N}. \tag{3.67}$$

3.4.8.2 F_β Measure

Neither precision nor recall are perfect effectiveness measures by themselves. To have an overall performance of a classifier, one needs to take both precision and

recall into account. There are several means to combine these two measures effectively such as break-even, Receiver Operating Characteristic (ROC), and F_β measure. One of the most widely-used combined measures in multiclass classification task is the F_β measure. F_β measure is the harmonic average of precision and recall:

$$F_\beta = \frac{(1+\beta^2)\pi\rho}{\beta^2\pi+\rho}. \tag{3.68}$$

β in Equation (3.68) defines the relative importance degree between precision and recall. With $\beta = 1$, precision and recall contribute equally to F_β which is traditionally called the F-measure or F1-score:

$$F1_i = \frac{2\pi_i\rho_i}{\pi_i+\rho_i}. \tag{3.69}$$

To calculate F-measure globally, similar to precision and recall (Section 3.4.8.1), two methods can be applied:

1. *micro-averaging*: Global F1-score is the harmonic mean of micro precision (Equation 3.64) and micro recall (Equation 3.65):

$$F1^\mu = \frac{2\pi^\mu\rho^\mu}{\pi^\mu+\rho^\mu}. \tag{3.70}$$

2. *macro-averaging*: Global F1-score is the average of F1-score over different categories:

$$F1^M = \frac{\sum_{i=1}^{N} F1_i}{N}. \tag{3.71}$$

It is important to note that micro-averaging and macro-averaging results could be quite different. The micro-averaging methods put more weight on more frequent categories, whereas macro-averaging methods highlight the performance of the classifier on rare categories where there are few positive training data. Thus, choosing one of these methods depends on the type of the application.

Chapter 4

Experimental Results And Interpretations

This chapter describes data set specifications and experimental framework used in this work. It then provides the empirical results and discussions regarding the performance of the proposed approach.

4.1 Data Sets

The proposed approach has been assessed on queries collected from four major telecommunication companies' call centers. Every query is tagged by a human expert to one single category. Table 4.1 reports details about the number of queries, categories, and words for four data sets. Table 4.2 shows the scattering of the queries among the categories. For example, in the case of Telecom 1 data set, 39% and 50% of the categories contain 80% and 90% of the queries respectively. The data set categories have been listed in Appendix A.

4.2 Experimental Setup

Four different classifiers, including a Linear Support Vector Machine (SVM) from LIBSVM [9], a Maximum Entropy (MaxEnt) and Naive Bayes (NB) probabilistic from MALLET [27], and *a* k-Nearest Neighbor (k-NN) classifier, have been used to assess the impact of features' weighting on query classification. A few metrics were used to quantify the classification performance, namely micro F1-measure, macro F1-measure, macro precision, and macro recall[1]. They provide different views of the classifier performance, and may give different results. The micro measure puts more weight on more frequent categories, while macro measures highlight the performance of a classifier on categories with a small number of positive training

[1]It can be proved that for a single-label classification, all micro F1-measure, micro recall, micro precision, and accuracy are equal.

Table 4.1: Data sets Specification

	Total Queries	Number of Categories	Number of Words
Telecom 1	2511	18	1043
Telecom 2	4200	26	1550
Telecom 3	13364	26	3357
Telecom 4	116275	100	3208

Table 4.2: Percentage of the categories containing 80% and 90% of the queries

		Query Percentage	
		80%	90%
Category Percentage	Telecom 1	39%	50%
	Telecom 2	23%	35%
	Telecom 3	42%	58%
	Telecom 4	19%	31%

data. To find out if the improvement or regression provided by the proposed method is statistically significant, the student t-test has been performed on the results. Both P-value ≤ 0.05 and P-value ≤ 0.01 have been reported.

4.3 Results And Interpretations

All the experiments have been carried out using four-fold cross validation. Each fold has been repeated ten times, and the average has been taken as a final result. For the k-NN classifier the numbers are the average taken for different number of neighborhoods in the range of 1 to 10. A pre-processing task including stemming and stop-words removal has been applied for both training and testing data sets. To train the classifiers and regression models, the top 15% most important features, ranked by Information Gain, have been chosen. To compare the performance of the proposed approach, all classifiers have also been trained and tested with unweighted data, which means each feature is assigned a binary weight.

4.3.1 Macro Precision and Recall Performance

Figure 4.1 shows the results for macro precision and macro recall. The performance for the SVM classifier in the case of macro precision is unsatisfactory. The proposed approach was not able to improve the precision performance; however, in the case of macro recall, the proposed approach indeed improved the recall performance for data set 2 and data set 3. In the case of the MaxEnt classifier, the new weighting method increased the performance for both macro precision and macro recall for all data sets but data set 4. For the Naive Bays classifier, the proposed approach

boosted the performance for both macro precision and recall, except for Telecom 4's macro precision. The k-NN classifier also enjoyed a jump in both macro precision and macro recall figures, although the proposed method has not been well suited for data set 3 in the case of k-NN.

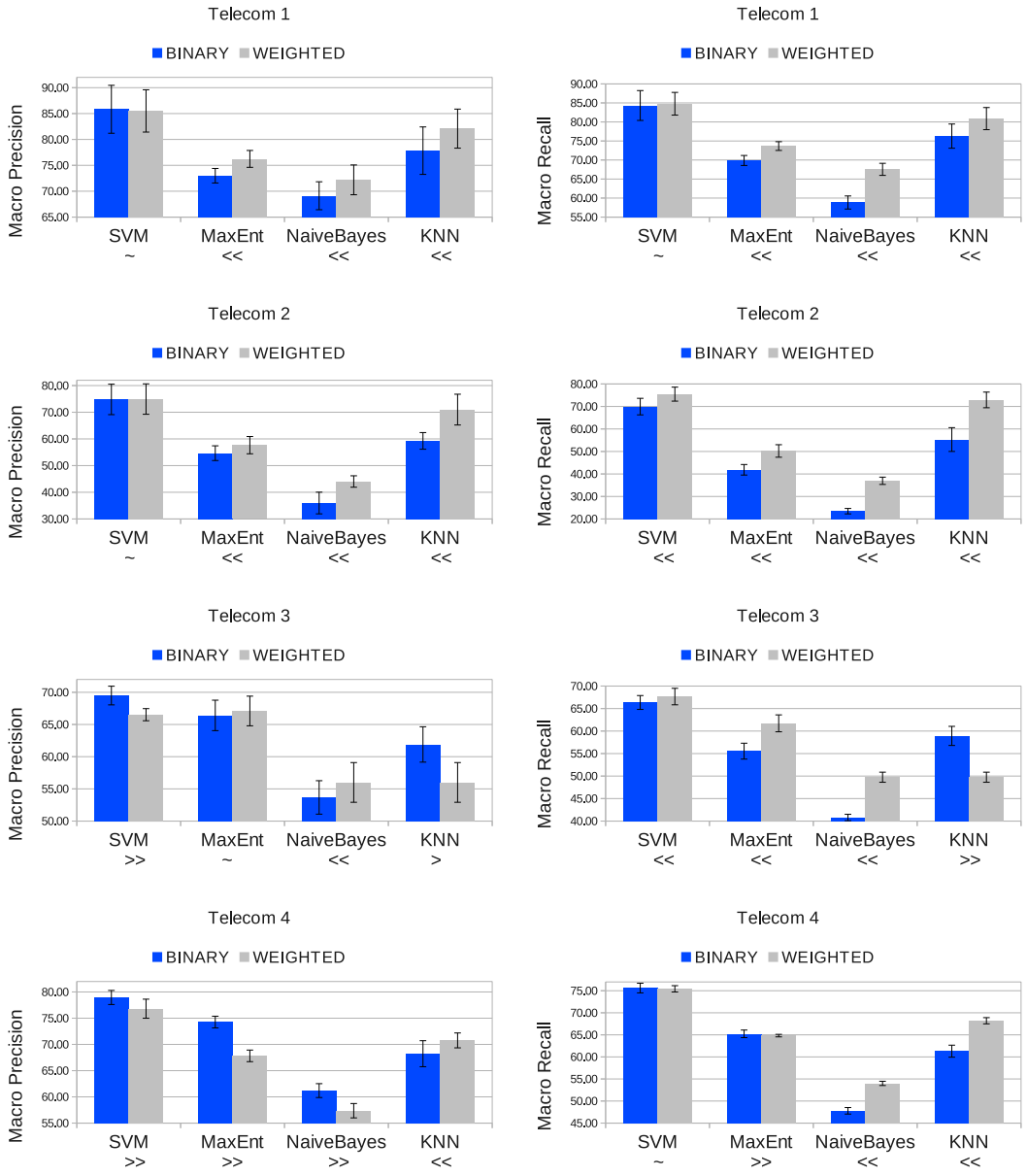

Figure 4.1: macro Precision, macro Recall and significance test results
"$\ll$" or "$\gg$" means P-value ≤ 0.01
"$<$" or "$>$" means $0.01 <$ P-value ≤ 0.05
"$\sim$" means P-value > 0.05

4.3.2 Macro and Micro F1-measure performance

Figure 4.2 depicts the macro F1-measure, micro F1-measure, and the significance test results. The proposed features' weighting approach improved both macro and micro F1-measures in the case of Naive Bayes, and k-NN classifiers for all data sets. One can observe high variance with k-NN compared to other classifiers. The reason is that these numbers are the average of macro and micro F1-measures over different neighborhoods (from 1 to 10). Nevertheless, the weighting approach is able to reduce this variance substantially in most of the cases. The macro and micro F1-measures have been increased by applying the weighting method to the Maximum Entropy classifier, except for data set 4 and the micro F1-measure of data set 1, where some performance degradation is observed. It is interesting to see that there is no improvement in the case of an SVM classifier except for the second testing set, where some improvement has been achieved on the macro F1-measure.

4.3.3 Precision vs. Recall

Although the F1-measure combines precision and recall and shows the overall performance of a classifier, a balance between precision and recall is needed for a high quality F1-measure. It can be beneficial to see what the trade-off is between precision and recall for each classifier with and without applying a weighting scheme. Figure 4.3 shows the precision vs. recall for different classifiers. The dotted curves depict the points with equal F1-measures, showing the direction for getting better F1-measure performance. In the case of SVM classifiers, binary weighting tends to have a better precision, while the proposed method tends to have a better recall. However for data set 2, the approach has boosted the recall without any regression in precision. For the rest of the classifiers, the proposed approach has yielded a better balance between precision and recall, and managed to improve both precision and recall. The exceptions are MaxEnt and Naive Bays for data set 4, and k-NN for data set 3. One possible explanation for these anomalies is that similar concepts in data sets 3 and 4 have been labeled differently due to human error. These errors can yield in contradiction weight values when calculating weights. It can then result in weak regression models and/or classifiers.

4.3.4 Absolute Improvement

Table 4.3 shows the absolute improvement averaged over four data sets, obtained by the proposed weighting method for each classifier. Substantial improvements have been bolded. As observed by other forms of analysis, the proposed method has managed to only improve the macro recall measure in the case of SVM classifier. Overall macro F1-measure and macro recall have also been improved for the MaxEnt classifier. For the Naive Bays and k-NN classifiers, all overall measures have been improved substantially.

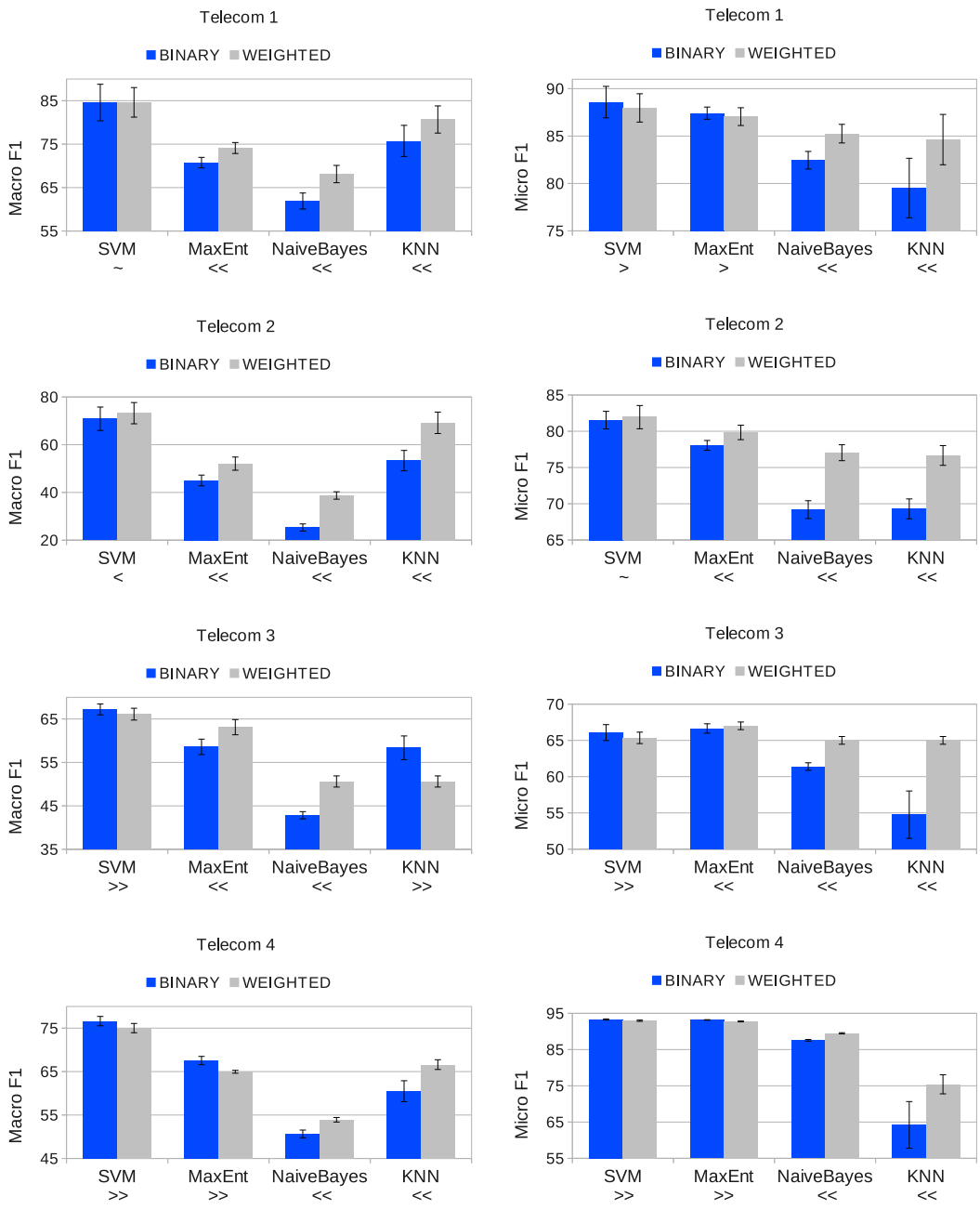

Figure 4.2: macro F1-measure, micro F1-measure and significance test results
"$\ll$" or "$\gg$" means P-value ≤ 0.01
"$<$" or "$>$" means $0.01 <$ P-value ≤ 0.05
"$\sim$" means P-value > 0.05

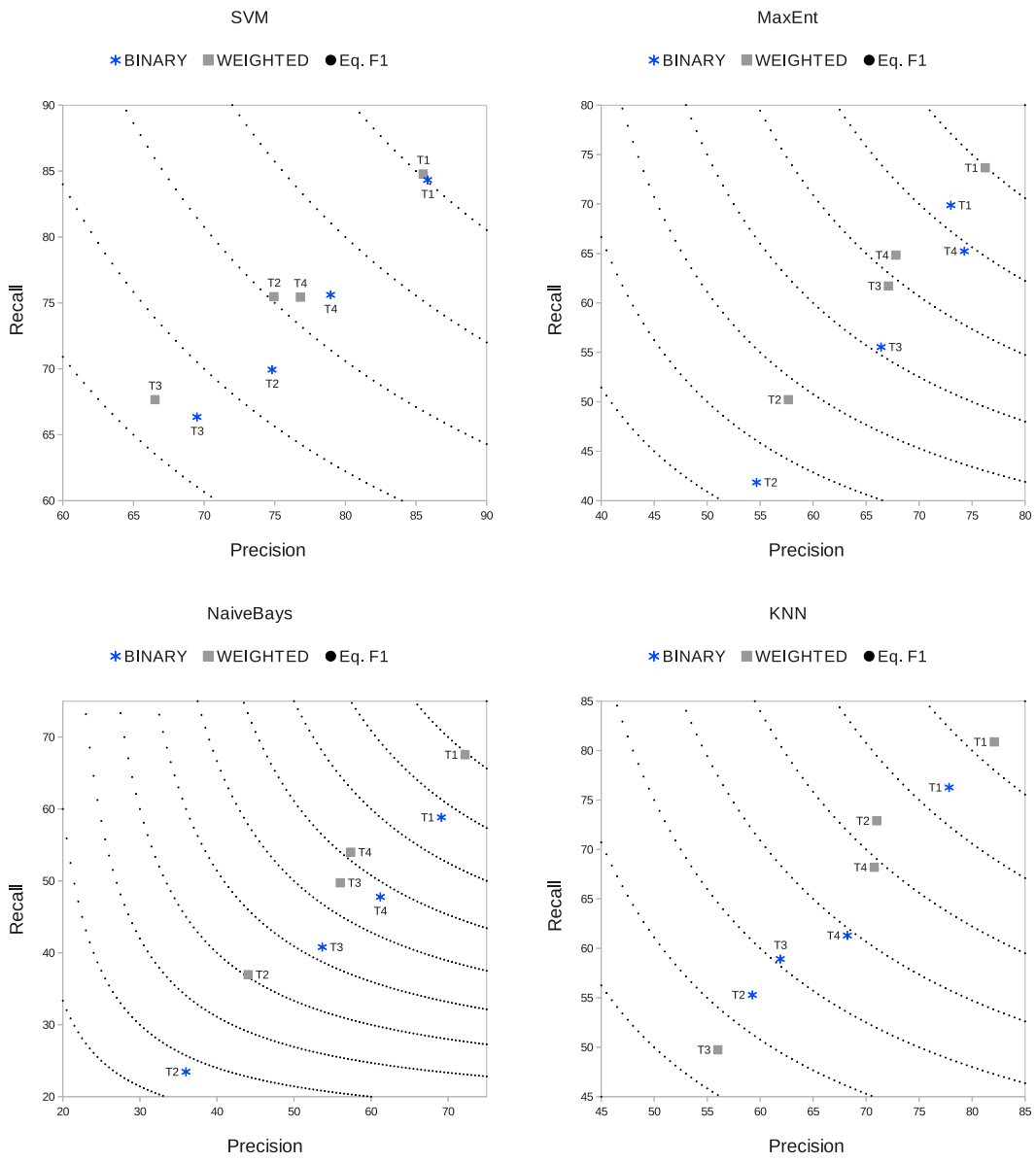

Figure 4.3: Precision vs. Recall

4.3.5 Conclusion

The SVM classifier seems to be somehow weighting-resistant. In other words, SVM classification performance is not affected much by applying weighting schemes. This finding is in concurrence with other works in the literature [12, 13]. Having said this, Forman [13] has shown that Bi-Normal Separation weighting can improve the performance of the SVM classifier. The significance test results show that the improvements obtained by the proposed method are reliable and statistically significant. Overall, the proposed method has done a better job in the case of

Table 4.3: Average Absolute Improvement

	f1-macro	P-macro	R-macro	f1-micro
SVM	-0.08%	-1.32%	**1.78%**	-0.32%
MaxEnt	**3.09%**	0.12%	**4.48%**	0.34%
Naive Bays	**7.66%**	**2.41%**	**9.34%**	**4.05%**
k-NN	**4.79%**	**3.16%**	**4.98%**	**8.48%**

macro recall. Thus, if the intended goal is to increase recall performance, it's highly recommended to adopt the proposed method. However as shown in Table 4.3, there is no substantial overall loss, even in the case of SVM, by adopting the proposed method, thus adopting the method may not result in regression for most cases.

Chapter 5

Summary and Future Work

This work have introduced a novel approach to assign weights to the features in a query classification task. First pseudo documents are created by grouping queries of the same category. Feature weights were then calculated, by applying *tf-idf* weighting scheme to those pseudo documents. Since it is not possible to build those pseudo documents for testing data, calculating weights for testing query is a challenge. A set of regression models is then developed to learn weights from training data, and then applied learned weights to the testing queries. Four different classifiers, including SVM, MaxEnt, Naive Bays, and k-NN, have been trained with and without using feature weights. All micro and macro F1-measures, macro recall, and macro precision metrics can be substantially improved by applying weight to the queries using the proposed method. In the experiments , different classifiers enjoyed absolute improvement up to 9% and the proposed approached showed consistent macro recall improvement. Although experiments have been carried out using the *tf-idf* weighting technique, the proposed approach is weighting-scheme-independent, and therefore can be generalized to any technique.

As a future work, there are three areas which can potentially improve the overall performance of the approach:

1. An approach is outlined in Section 3.1 which elaborates how to calculate feature weights for a given collection of queries. The quality of the weights obtained in this step is a crucial factor for the performance of the whole proposed approach, since it can deeply impact the performance of the next steps. Thus investigating a more advanced and smarter way of clustering queries into groups can be greatly beneficial to the proposed approach. On the other hand, investigating a completely new approach for calculating weights for a given query collection is an open area.

2. The next area is to improve the means of learning weights. In this work, a Support Vector Regression model is used for learning the weights. The experiments in this work can be expanded by investigating the other regression models such as Gaussian Process [30]. Gaussian Process has proved itself as one of the strongest method for doing regressions.

Another completely different approach for learning weights is to utilize the relatively new Stacked Denoising Autoencoders (SDA) [42] method. In a nutshell SDA is a method to reconstruct original data from noisy data. The idea is to train an SDA model with a weighted query collection and use it to reconstruct a new version of a test query which is weighted by SDA.

3. The other area for improving the performance of the approach is to investigate other weighting schemes. In those experiments, SVM classifier was somehow resistance to apply *tf-idf* weights. Bi-Normal Separation is one of the candidate weighting schemes to look into, since it has been reported to work well with SVM classifiers in the text categorization field [13].
   The Generalized Probabilistic Descent (GPD) algorithm [24] also looks very promising to be adopted as a base method for learning weights. It has been shown that GPD is able to calculate very meaningful feature weights (see Figure 2.2), thus could be capable of improving SVM classifier performance, as well as improving the other classifiers performance even more.

References

[1] Shigeo Abe. *Support Vector Machines for Pattern Classification*. Springer, 2005.

[2] Arash Abghari, Kacem Abida, and Fakhri Karray. Features' weight learning towards improved query classification. In *Proceedings of the Third International Conference on Autonomous and Intelligent Systems*, AIS'12, pages 184–191, Berlin, Heidelberg, 2012. Springer-Verlag.

[3] Iyad Batal and Milos Hauskrecht. Boosting KNN text classification accuracy by using supervised term weighting schemes. In *Proceedings of the 18th ACM conference on Information and knowledge management*, pages 2041–2044. ACM, 2009.

[4] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, Sept. 1975.

[5] Adam L. Berger, Vincent J. Della Pietra, and Stephen A. Della Pietra. A maximum entropy approach to natural language processing. *Comput. Linguist.*, 22(1):39–71, March 1996.

[6] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A Training Algorithm for Optimal Margin Classifiers. In *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, pages 144–152. ACM Press, 1992.

[7] Christopher J. C. Burges. A Tutorial on Support Vector Machines for Pattern Recognition. *Data Min. Knowl. Discov.*, 2(2):121–167, June 1998.

[8] Bob Carpenter and Jennifer Chu-Carroll. Natural language call routing: a robust, self-organizing approach. In *ICSLP'98*, 1998.

[9] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:1–27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[10] Corinna Cortes and Vladimir Vapnik. Support-Vector Networks. In *Machine Learning*, pages 273–297, 1995.

[11] S. Cox. Discriminative techniques in call routing. In *Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Processing (ICASSP '03)*, volume 1, 2003.

[12] Franca Debole and Fabrizio Sebastiani. Supervised Term Weighting for Automated Text Categorization. In *In Proceedings of SAC-03, 18th ACM Symposium on Applied Computing*, pages 784–788. ACM Press, 2003.

[13] George Forman. BNS feature scaling: an improved representation over tf-idf for svm text classification. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, pages 263–270, 2008.

[14] George Forman, Isabelle Guyon, and André Elisseeff. An extensive empirical study of feature selection metrics for text classification. *Journal of Machine Learning Research*, 3:1289–1305, 2003.

[15] Seymour Geisser. *Predictive inference: An introduction*. Number 0412034719. Chapman & Hall (New York), 1993.

[16] A.L. Gorin, H. Hanek, R. Rose, and L. Miller. Automated call routing in a telecommunications network. In *Interactive Voice Technology for Telecommunications Applications, 1994., Second IEEE Workshop on*, 1994.

[17] Geoffrey Hinton. Training Products of Experts by Minimizing Contrastive Divergence. *Neural Computation*, 14:1771–1800, 2000.

[18] Qiang Huang and Stephen J. Cox. Automatic call-routing without transcriptions. In *INTERSPEECH'03*, 2003.

[19] Hui Jiang, Pengfei Liu, and Imed Zitouni. Discriminative training of naive Bayes classifiers for natural language call routing. In *INTERSPEECH*, 2004.

[20] Rong Jin, Joyce Y. Chai, and Luo Si. Learn to weight terms in information retrieval using category information. In *Proceedings of the 22nd International Conference on Machine Learning*, pages 353–360. ACM, 2005.

[21] W. Karush. Minima of Functions of Several Variables with Inequalities as Side Constraints. Master's thesis, Dept. of Mathematics, Univ. of Chicago, 1939.

[22] S. S. Keerthi, S. K. Shevade, C. Bhattacharyya, and K. R. K. Murthy. Improvements to Platt's SMO Algorithm for SVM Classifier Design. *Neural Computation*, 13:637–649, 2001.

[23] H. W. Kuhn and A. W. Tucker. Nonlinear Programming. In *Proc. second Berkeley Symp. on Math. Statist. and Prob.*, pages 481–492. Berkeley: University of California Press, 1951.

[24] H.-K. J. Kuo and Chin-Hui Lee. Discriminative training of natural language call routers. 11(1):24–35, 2003.

[25] Man Lan, Chew Lim Tan, Jian Su, and Yue Lu. Supervised and Traditional Term Weighting Methods for Automatic Text Categorization. *IEEE Trans. Pattern Anal. Mach. Intell.*, 31(4):721–735, apr 2009.

[26] J. Scott Long. *Regression Models for Categorical and Limited Dependent Variables*. Number 978-0-8039-7374-9. Sage, 1997.

[27] Andrew Kachites McCallum. MALLET: A Machine Learning for Language Toolkit. http://mallet.cs.umass.edu, 2002.

[28] Klaus-Robert Müller, Sebastian Mika, Gunnar Rätsch, Koji Tsuda, and Bernhard Schölkopf. An introduction to kernel-based learning algorithms. *IEEE TRANSACTIONS ON NEURAL NETWORKS*, 12(2):181–201, 2001.

[29] N.J. Nilsson. *Learning Machines: Foundations of Trainable Pattern-Classifying Systems*. McGraw-Hill, 1965.

[30] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2006.

[31] Gorin Parker Sachs, A. L. Gorin, B. A. Parker, R. M. Sachs, and J. G. Wilpon. How May I Help You? *Speech Communication*, 23:113–127, 1997.

[32] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. In *INFORMATION PROCESSING AND MANAGEMENT*, pages 513–523, 1988.

[33] R. Sarikaya, G. E. Hinton, and B. Ramabhadran. Deep belief nets for natural language call-routing. In *Proc. IEEE Int Acoustics, Speech and Signal Processing (ICASSP) Conf*, pages 5680–5683, 2011.

[34] Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM Comput. Surv.*, 34(1):1–47, March 2002.

[35] Amit Singhal. Modern information retrieval: a brief overview. *BULLETIN OF THE IEEE COMPUTER SOCIETY TECHNICAL COMMITTEE ON DATA ENGINEERING*, 24:2001, 2001.

[36] Alex J. Smola and Bernhard Schölkopf. A tutorial on support vector regression.

[37] Pascal Soucy and Guy W. Mineau. Beyond TFIDF weighting for text categorization in the vector space model. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, pages 1130–1135. Morgan Kaufmann Publishers Inc., 2005.

[38] N. Tyson and V. C. Matula. Improved lsi-based natural language call routing using speech recognition confidence scores. In *Proc. Second IEEE Int. Conf. Computational Cybernetics ICCC 2004*, pages 409–413, 2004.

[39] S. Ullah, F. Karray, A. Abghari, and S. Podder. Soft computing-based approach for natural language call routing systems. In *Proc. 9th Int. Symp. Signal Processing and Its Applications ISSPA 2007*, pages 1–4, 2007.

[40] V. Vapnik. *Estimation of Dependences Based on Empirical Data*. Springer, 1982.

[41] Vladimir N. Vapnik. *The nature of statistical learning theory*. Springer-Verlag New York, Inc., 1995.

[42] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and Composing Robust Features with Denoising Autoencoders. In *Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML'08)*, pages 1096–1103. ACM, 2008.

[43] Andrew R. Webb. *Statistical Pattern Recognition*. John Wiley & Sons Ltd., second edition, 2002.

[44] C. Wu, D. Lubensky, J. Huerta, X. Li, and H.-K. J. Kuo. A framework for large scalable natural language call routing systems. In *Proc. Int Natural Language Processing and Knowledge Engineering Conf*, pages 65–71, 2003.

[45] Yiming Yang and Jan O. Pedersen. A Comparative Study on Feature Selection in Text Categorization. In *Proceedings of the Fourteenth International Conference on Machine Learning*, pages 412–420. Morgan Kaufmann Publishers Inc., 1997.

[46] I. Zitouni. Constrained Minimization and Discriminative Training for Natural Language Call Routing. *Trans. Audio, Speech and Lang. Proc.*, 16(1):208–215, jan 2008.

Appendix A

Data Set Categories

Telecom 1:

- accountBalance
- accountInformationAndChanges
- ADSLDisambig
- ADSLTroubleshooting
- backoff
- billingAndPayments
- billingProblemsDisambig
- billReprint
- bundledServices
- callBlocking
- callDisplayDisambig
- callForwardingDisambig
- callingCardDisambig
- callingCardPassword
- callWaitingDisambig
- cancelInternet
- cancelPhoneFeatures
- cancelService

Telecom 2:

- account
- accountBalance
- accountNumber
- billing
- billingCharges
- billReprint
- collectcall
- collectionAgency
- creditDepartment
- credits
- detailedBilling
- directoryAssistance
- eBill
- finalsDepartment
- firstBill
- genericRequest
- latePaymentCharges
- oneBill
- payment
- paymentNotification
- phone
- preAuthorizedDebit
- refundCheck
- systemAccessFee
- taxes

- webBanking

Telecom 3:

- appointment
- authxfer
- autoxfer
- billing
- budgetplan
- credit
- duplicatebill
- energy
- fortyeighthr
- hazard
- help
- meaningless
- outage
- paybyphone
- paylocs
- payplan
- phupdate
- pilot
- programs
- start
- stop
- stopvague
- svcvague
- tentwenty

- transfersvc
- xfer

Template 4:

- BillingBalance
- BillingCopy
- BillingDetail
- BillingGeneral
- BillingPassword
- BillingProblem
- CancelService
- ChangeAccount
- ChangeAddress
- ChangeNumber
- ChangePhone
- ChangePlan
- ChangeService
- FeatureAdd
- FeatureAddCallBlock
- FeatureAddCallerIDBlock
- FeatureAddCallForwarding
- FeatureAddGetItNow
- FeatureAddMobileWeb
- FeatureAddUnlimitedInMessaging
- FeatureGeneral
- FeatureGeneralCallBlock
- FeatureGeneralCallerIDBlocking

- FeatureGeneralCallForwarding
- FeatureGeneralFnF
- FeatureGeneralGetItNow
- FeatureGeneralMobileWeb
- FeatureGeneralUnlimitedInMessaging
- FeatureHelp
- FeatureHelpCallBlock
- FeatureHelpCallerIDBlock
- FeatureHelpCallForwarding
- FeatureHelpGetItNow
- FeatureHelpMobileWeb
- FeatureHelpUnlimitedInMessaging
- FeatureRemove
- FeatureRemoveCallBlock
- FeatureRemoveCallerIDBlock
- FeatureRemoveCallForwarding
- FeatureRemoveGetItNow
- FeatureRemoveMobileWeb
- FeatureRemoveUnlimitedInMessaging
- GeneralAccount
- GeneralHelp
- GeneralNonWireless
- GeneralPhone
- GeneralPlan
- GeneralService
- International

- LNP
- LostOrStolen
- Minutes
- Operator
- PayAddress
- PayArrangement
- PayAutoPay
- PayCreditCard
- PayMakePayment
- PayOnline
- Prepaid
- RecentAccountActivity
- ReturnCall
- RingbackTones
- RingTones
- SalesActivation
- SalesGeneral
- SalesPhone
- Store
- TechBroadband
- TechCalling
- TechDroid
- TechFeature3WayCalling
- TechFeature411
- TechFeatureCallerID
- TechFeatureCallWaiting

- TechGeneral
- TechLock
- TechNationalAccess
- TechPDA
- TechPDABlackberry
- TechPDATreo
- TechPhone
- TechPictureMessaging
- TechRoaming
- TechTextBlock
- TechTextMessaging
- TechVCast
- TechVCastMusic
- TechVCastVideo
- TechVCastVPack
- TechVZHub
- TechVZNavigator
- TechWebPasswor
- TechWebPassword
- TechWebSite
- TechWebUsername
- VoicemailGeneral
- VoicemailPassword
- VoicemailProblem
- VoicemailSetUp

Glossary

BNS Bi-Normal Separation

CM Constrained Minimization

CT Corrective Training

DBN Deep Belief Networks

FN False Negative

FP False Positive

GA Genetic Algorithm

GPD Generalized Probabilistic Descent

idf Inverse Document Frequency

IG Information Gain

IR Information Retrieval

IRLS Iteratively Re-weighted Least Squares

LDA Linear Discriminant Analysis

LM Language Model

LSI Latent Semantic Indexing

MAP Maximum A Posteriori

MaxEnt Maximum Entropy

MI Mutual Information

NB Naive Bayes

NBC Naive Bayes Classifier

OR Odds Ratio

RBM Restricted Boltzmann Machine

rf Relevance Frequency

ROC Receiver Operating Characteristic

SDA Stacked Denoising Autoencoders

SVD Singular Value Decomposition

SVM Support Vector Machine

SVR Support Vector Regression

TC Text Categorization

tf Term Frequency

TN True Negative

TP True Positive

CHI χ^2

F Feature Space

f_i ith feature

k-NN k-Nearest Neighbor

L Lagrangian function

$\boldsymbol{\alpha}$ Lagrange multipliers

N Total number of documents

$P(x)$ Probability of x

π Precision

$P(x|y)$ Conditional probability of x given y

Q Query Collection

q_i ith query

r_i ith regression model

ρ Recall

ξ Slack Variables

$\boldsymbol{\beta}$	Weight Vector
$\mathbf{w}$	Weight Vector
w_i	Weight corresponding to the ith feature
$\mathbf{x}$	Input Vector