Multi-class Multi-tag Classifier System for StackOverflow Questions

José R. Cedeño González*, Juan J. Flores Romero†, Mario Graff Guerrero‡ and Felix Calderón§

Departamento de Estudios de Posgrado de la Facultad de Ingeniería Eléctrica
Universidad Michoacana de San Nicolás de Hidalgo
*jcgonzalez@dep.fie.umich.mx, †juanf@umich.mx, §calderon@umich.mx
INFOTEC
‡mario.graff@infotec.com.mx

***Abstract*—This work approaches the text document classification problem derived from the contest "Identify Keywords and Tags from Millions of Text Questions", published on the website Kaggle. Using data from the StackOverflow website, the problem is to predict the tags assigned to questions. This categorization is multi-class and multi-tag, which means, a question can be assigned to different topics and can also have several tags. To solve this problem, we propose a 5-way multi-class classifier system. The results obtained by this classification scheme are discussed, by analysing certain score metrics of the classifier system. Competitive results were obtained by the 5-way classifier system, obtaining F1 scores ranging from 0.59 to 0.76. The main contribution of this paper lies on the preprocessing (which implements the feature extraction phase) and the multi-tag multi-class classification scheme.**

I. Introduction

The system proposed in this paper is based on the contest Identify Keywords and Tags from Millions of Text Questions, which was published in the web site Kaggle [1]. This contest consisted on predicting the tags assigned to a set of questions from the StackOverflow [2] website. StackOverflow is a collaborative social network where users are invited to pose and answer questions about computer science topics; those questions include usage of computer programs, algorithms, programming languages, etc. Each question can have at most five tags, these tags are keywords that serve to distinguish the questions and organize them in topics. These tags are assigned manually by the user (with an auto-fill feature), or by selecting them from a list of most popular tags.

For this contest, StackOverflow provided a training set of more than three million questions; each question consists of an identifier, the question title or subject, the question body (which is a free format text field) and a list of the tags assigned to that question. They also provided a test set which has around 20 thousand questions in the same format, but without the tags. The question body comes as the user captured it, including HTML tags to format the question. Although the language of the web site is English, some questions contain non ASCII characters. Also, some questions contain code segments in different programming and mark-up languages, or even alphanumeric tables (with or without format).

The solution to this problem provides other researches ideas, schemes, and algorithms for preprocessing and classification of large data sets. The preprocessing ideas lie in the area of text processing, based on statistical properties of the distribution of words, not on the recognition and exploitation of semantic structures.

This paper is organized in the following manner: first, we discuss about previous and related work, specifically work done over the same contest. Then, we illustrate the design and implementation of the classification scheme developed, by describing the algorithms and models used. We show the results obtained by the classification system and compare it to other works. Finally, we analyse the obtained results.

II. Background and Related Work

The data is delivered in the exact format it was written by the users, so it is necessary to transform it to be processed. Hong and Fang [3] generate a bag of words with the frequency of each word and then applies *TF-IDF* (Term Frequency-Inverse Document Frequency, a numerical statistic that serves to reflect how important a word is to a document in a collection or corpus [4]) with the document frequency calculated from the training set and generate centroids for each tag using these vectors. Then, during the evaluation for each question, it is converted to a TF-IDF vector and the distance between the evaluated question and each tag centroid is computed using the cosine similarity.

Stanley, Clayton and Byrne [5] employ a cognitively-inspired Bayesian probabilistic model based on ACT-R's declarative memory retrieval mechanisms (an ACT-R is a cognitive architecture that aims to define the basic and irreducible cognitive and perceptual operations that enable the human mind). This model uses the following methodology: for each title and question body, the model returns a series of activations for each possible tag, where the tag with the highest activation is the one associated with the evaluated question. The association strength between words used in questions and their tags was determined using two thirds of the data set. The model predictions are based on the number of co-occurrences between the words and the tags used in the questions. These tags are clustered using a database of synonym tags.

Parikh [6] employed discriminant classifiers, where a classifier is built for each tag and its predictions are the more likely tags for that question. They employed the 500 most frequent tags and generated a classifier for each one of the tags. To generate a prediction, they run each question in the test set through 500 classifiers and by using a selection scheme they generate their predictions.

The classification of questions requires special attention since every one of them can have from one to five different tags, this makes this issue a multi-tag classification problem.

III. Design and Implementation

Since the questions are in the format provided by the StackOverflow database, it was necessary to perform certain preprocessing to convert it to usable data for a classifying method. Also, because of the length of the questions and the vocabulary used in them which was extensive and variate. Two preprocessing methods were designed for this purpose: one performing a series text cleaning tasks and a “lazy” preprocessing method that makes use of certain parameters of the software library. In both methods the resulting data is vectorized by a Bag of Words [7] (BoW) vectorizer.

A. Traditional Preprocessing

With this preprocessing, certain text elements are identified and reduced or eliminated by querying each question contained in the training set and individually processed for each kind of element to remove. Basically, this preprocessing removes: punctuation signs; unusual words i.e. everything that is considered specific for the question, like variable names, proper names, idioms, numbers, etc; and HTML tags to format the questions.

Nonetheless, some exceptions were done in the case of unusual words, specifically words used in computer science. To preserve certain terms, a computer related words lexicon was made to be considered usual words.

In order to reduce the number of features, a stemmer was incorporated to the preprocessing task, which is a process that removes morphological affixes from words, leaving only the word stem [8].

The complete integration of these element elimination steps can be appreciated in Algorithm 1.

Algorithm 1 Preprocess question algorithm

```
1: function PREPROCESS_QUESTION($quest, punct, voc, stemmer, stop$)
2:     text ← remove_html($quest$)
3:     for $sign$ in $punct$ do
4:         text ← text.replace($sign$, “”)
5:     end for
6:     text ← tolower($text$)
7:     text ← remove_unusual($text, voc$)
8:     useless ← difference($text, stop$)
9:     for $word$ in useless do
10:        text ← text.replace($word$, “”)
11:    end for
12:    text ← keep_stems($text, stemmer$)
13:    return text
14: end function
```

To preprocess the questions we used Algorithm 2, where for each question contained in the training set (contained in a csv file), Algorithm 1 is called, then for every 1,000 processed questions they are stored in a partial file to avoid memory saturation (Line 12).

Algorithm 2 Preprocessing algorithm

```
1: function PREPROCESSING($csv\_file$)
2:     vocabulary ← load_vocabulary(vocabulary_path)
3:     punctuation ← punctuation()
4:     stemmer ← PorterStemmer()
5:     stopwords ← stopwords()
6:     n ← 0
7:     questions ← list()
8:     for question in csv_file do
       questions.append(preprocess_question(question,
9:     punctuation, vocabulary,
10:    stemmer, stopwords))
11:        n ← n + 1
12:        if n $mod$ 1000 == 0 then
13:            write_to_disk(questions)
14:        end if
15:    end for
16:    write_to_disk(questions)
17: end function
```

This new file with the preprocessed questions is then passed to a vectorizer object which implements the BoW method. This object accepts any iterable data structure, which can be file handlers or lists. This process is described in Algorithm 3 (specifically in line 2).

Although this series of tasks extract most of the features necessary to perform classification, still many features were present in only one question. To remove them, the vectorizer uses a *minimum document frequency* (i.e. a parameter to control if a feature is present in less of a certain amount of documents) to remove every feature that has a document frequency smaller than 0.01 (Algorithm 3 line 3).

Algorithm 3 Traditional question vectorization

```
1: function VECTORIZATION($partial\_file$)
2:     vectorizer ← CountVectorizer()
3:     vectors ← vectorizer.fit_transform(partial_file,
       min_df=0.01)
4:     return vectors
5: end function
```

B. Lazy preprocessing

Considering that the vectorizer object can receive any iterable object a simpler method was devised, where by using an intermediate function that retrieves directly from the training set file the question text and stores it on a temporary list that gets iterated like a file. This method is described in Algorithm 4.

The lazy reader function simply reads the selected columns of the csv file, accumulates them with the yield operator and return it when the end of file is reached (Algorithm 5).

With the methods described above, it was possible to generate two preprocessing schemes, which we compared later with the classification schemes devised.

Algorithm 4 Lazy question vectorization

```
1: function LAZY_VECTORIZATION($train\_file$)
2:     vectorizer ←CountVectorizer()
3:     vectors ← vectorizer.fit_transform(lazy_reader(train_file),
       min_df=0.01, max_df=0.85)
4:     return vectors
5: end function
```

Algorithm 5 Reading of training file through a lazy reader

```
1: function LAZY_READER($train\_file$)
2:     csv_reader ← read_csv(train_file)
3:     while csv_reader.next() do
4:         yield(csv_reader[1] + ' ' + csv_reader[2])
5:     end while
6: end function
```

C. Feature Selection

Even though the number of features extracted was manageable by the classifying system, a feature selection method was included to ensure that only the most relevant features were used by the classifiers. For this work, the Linear Support Vector Classifier (LSVC) feature selection method was used. It has been observed [9] that, by making a slight modification to the SVM algorithm, it is possible to train an SVM classifier using the L_1 norm as penalty function. The L vector norm family, $L_p : \mathbb{R}^d \rightarrow \mathbb{R}$ is defined in Equation 1.

$$L_p(x) = \|x\|_p = \left(\sum_{i=1}^{d} |x_i|^p \right)^{\frac{1}{p}} \tag{1}$$

where x is a vector and d are its dimensions.

By using this vector norm it is possible to generate optimization functions. For this case, the optimization function, based on the norm L_1, is described in Equation 2.

$$\begin{aligned} &\min_w (Y - Xw)^T (Y - Xw) + \lambda \|w\|_1 \\ &= \min_w (Y - Xw)^T (Y - Xw) + \lambda \sum_{i=1}^{d} |w_i| \end{aligned} \tag{2}$$

where w is a weight vector and λ was set to 0.0005. This value was selected empirically by repeating this process with different values. Using this value caused a reduction of 50-70% of total features.

The obtained space is disperse i.e., many of the coefficients are zero, so we select the non-zero coefficients. By keeping only the features that correspond to non-zero coefficients, we are effectively reducing the number of features of the data [10].

D. Classification System

For this work a classification system composed of five multi-class classifiers was developed. Each classifier is focused on predicting a single tag of the five possible tags a question can have from a pool of several classes. To test the viability of this approach, we used two classification models: Naive Bayes (NB) [11] [12] and Support Vector Machines (SVM) [13].

1) Data preparation: Certain data preparation is needed to train the classifiers. Firstly, we need to indicate the tag position that the classifier is going to be trained, i.e. the first, second, third, fourth or fifth tag that a question can have. Once this is determined, the method retrieves the tags for that position for all questions.

Algorithm 6 does the following actions. It receives $file\_path$ which contains the questions in the form of feature vectors (variable X), its tags (variable y) and the classes selected to train (variable $tags$); tag is an integer that will serve to the algorithm to select the tag position that will be used to train the classifier. For every question (line 5) we check if the question has a tag that exist in the list $tags$ (line 6), if so, its index is stored in the indices list (line 7) along with its tag (line 8). These lists are then passed to the training algorithm (Algorithm 7) where the vectors indicated by its indices are the ones used to train the classifier.

Algorithm 6 Data preparation for classifier training

```
 1: function DATA_PREP($file\_path, tag$)
 2:     X, y, tags ← read_file(file_path)
 3:     indices ← list()
 4:     y_n ← list()
 5:     for row in y do
 6:         if row[tag] in tags then
 7:             indices.append(row.index)
 8:             y_n.append(row[tag])
 9:         end if
10:     end for
11:     indices, y_n ← shuffle(indices, y_n)
12:     clf ← train(indices, X, y_n)
13:     return clf
14: end function
```

2) Classifier training: Algorithm 7 makes use of a batching approach to train the classifiers to reduce memory consumption. Firstly on the algorithm, we determine the type of the classifier to train with the $classifier\_type$ variable (lines 4 through 8). After this, we calculate the number of training batches in line 9. Then, we slice the lists that contains the labels of the questions and the question vectors in lines 12 and 13. Notice that for the question vectors, we use the indices list to slice the list instead of doing it directly, this is done to avoid handling the actual data and preserve memory. We invoke the $partial\_fit$ method with the selected data.

3) Making predictions: To make predictions, the questions are passed to the five classifiers. Each classifier predicts a tag for its corresponding position. This will make a final prediction in the form of a list of integers, where each integer represents the associated tag.

Algorithm 8 receives a list $clfs$ that contains the trained classifiers, $threshold$ who is the length of the batches to process and X_test which is a randomly selected set of questions that were separated from the original data set. Then, we slice the test data set in batches (line 5). For every batch, we obtain the predictions of the five classifiers and arrange them by using the $column\_stack$ function on the all list (line 8). In occasions, the classifiers will make the same prediction so we devised a method to deal with these repeats and called it $filter$ (Algorithm 9), the result of the filtering process is

Algorithm 7 Classifier training

```
function TRAIN($X, y, classifier\_type$)
    start ← 0
    clf ← Null
    if classifier_type == 1 then
        clf ← NaiveBayes()
    else
        clf ← SVMClassifier()
    end if
    much ← ⌊ length(indices) / 1000 ⌋
    for i in range(much) do
        end ← (i + 1) * length(indices) / much
        y_train ← y[start : end]
        x_train ← X[indices[start : end]]
        clf.partial_fit(x_train, y_train)
        start ← end
    end for
    x_train ← X[indices[end : ]]
    y_train ← y[end : ]
    clf.partial_fit(x_train, y_train)
    return clf
end function
```

stored in a list called z (line 9). Then, the list z is rearranged and its contents stored in the y_pred list (line 11). Once all the batches are processed y_pred is returned to be compared with the true tags.

Algorithm 8 5-tag Prediction Method

```
function PREDICTOR($clfs, threshold, X\_test$)
    y_pred ← list()
    start ← 0
    end ← 0
    much ← length(X_test) / threshold
    for i in range(much) do
        end ← (i + 1) * length(x_test) / much
        all ← column_stack((clfs[0].predict(X_test[start : end]), clfs[1].predict(X_test[start : end]), clfs[2].predict(X_test[start : end]), clfs[3].predict(X_test[start : end]), clfs[4].predict(X_test[start : end])))
        z ← [filter(x) for x in all]
        for j in range(5) do
            y_pred[j] ← hstack(([x[j] for x in z]))
        end for
        end ← start
    end for
    all ← column_stack((clfs[0].predict(x_test[end : ]), clfs[1].predict(x_test[end :]), clfs[2].predict(x_test[end : ]), clfs[3].predict(x_test[end : ]), clfs[4].predict(x_test[end : ])))
    z ← [filter(x) for x in all]
    for j in range(5) do
        y_pred[j] ← hstack(([x[j] for x in z]))
    end for
    return y_pred
end function
```

In certain cases, two or more classifiers make the same prediction, so a filter was included to deal with repetitions. Algorithm 9 shows the filtering process. Basically, it generates a list f with as many x elements contained in z if they are not stored in the set $seen$ or if they are not have been added to $seen$ (line 4). This way we traverse z in order and skip any repeated element. This process will make f shorter than z if repeats were present, so we fill f with as many $empty-tag$ tags needed (line 6) and return f.

Algorithm 9 Repeated predictions filter

```
function FILTER($z$)
    seen ← set()
    seen_add ← seen.add
    f ← [x for x in z if not (x in seen or seen_add(x))]
    for i in range(5 - length(filtered)) do
        f.append(empty-tag)
    end for
    return f
end function
```

With the developed methods it was possible to process the questions and convert them to characteristic vectors, which were used to train the proposed classifier scheme. As we are going to present in the next section, it was possible to attain a series of results that show the capacity of the classifier method.

IV. Results

The metrics employed to measure the effectiveness of the classifier systems are the precision, recall and F1 value for multi-class and multi-tag classification. We chose to not use ROC (Receiver Operating Characteristic) curves because the evaluation metric employed for the competition was not ROC curves, and the representation of ROC curves for multi-class classifiers is not as intuitive as their binary classification counterpart.

Precision is defined as the proportion of positive results that are truly positive. It is expressed as:

$$p = \frac{\text{\# of true positives}}{\text{\# of true positives} + \text{\# of false positives}} \tag{3}$$

Recall measures the proportion of existing positives that are actually identified as positives.

$$p = \frac{\text{\# of true positives}}{\text{\# of true positives} + \text{\# of false negatives}} \tag{4}$$

The F1 value is given by Equation 5.

$$F1 = 2\frac{p \times r}{p + r} \tag{5}$$

where p is the precision and r is the recall. The F1 value can be interpreted as the harmonic mean of the precision and recall where the best value is when $F1 = 1$ and worse when tends to 0 [14]. This metric is employed in the classification problem because it weights equally both the precision and recall, which a good classifier will try to maximize simultaneously, which means that the classifier is, besides of being accurate, it also limits accordingly its predictions.

TABLE I. ACRONYMS

Acronym	Description
TP	Traditional Preprocessing
LP	Lazy Preprocessing
TP LSVC	Traditional Preprocessing with Linear SVC Feature Selection
LP LSVC	Lazy Preprocessing with Linear SVC Feature Selection
NB	Naive Bayes Classifier
SVC	Support Vector Machine Classifier

As defined on section III-D, two versions of the same system were tested, one using Naive Bayes classifiers and the other using SVM classifiers with Stochastic Gradient Descent penalty. For both systems LSVC was used as a feature selection mechanism.

Table I describes the acronyms that describe the variants used in this work.

A. Execution parameters

Since there is little more than 40 thousand different tags in the data set, it is extremely difficult to train a classifier with this number of classes. Also, to compare the behavior of the generated classifiers against other systems, the classifiers were trained with the 100, 200, and 500 most frequent tags, plus the empty tag.

The classifier systems were trained using the number of data indicated in Table II and by using 365,590 questions as test set. These training and test set lengths represent the 90% and 10% of the total data. The questions of each set were ordered randomly.

TABLE II. TRAINING SIZE PER TAG

tags	First Tag	Second Tag	Third Tag	Fourth Tag	Fifth Tag
100	2,758,527	1,721,079	1,843,326	2,633,436	3,284,595
200	3,013,218	2,101,518	2,042,352	2,692,251	3,294,720
500	3,290,310	2,600,415	2,414,520	2,838,330	3,326,076

We present the results as a dispersion map, where the abscissa represents the recall and the ordinate represent the precision. Every classification scheme is represented as a point over this space in which its coordinates are its recall and precision scores. The coordinate $(1,1)$ represents the best classification possible and the coordinate $(0,0)$ represents the worst classification possible.

From Figure 1 can be noticed that the 5-tag classifier system paired with SVC's perform better than the NB classifiers, but most curiously, the use of any feature selection method did not impacted in the classification performance significantly. The preprocessing method gave mixed results for both types of classifiers.

Figure 2 shows a recession for every classifier, mainly in their recall values. This indicates that, when the number of classes to predict increases, the classifiers tend to be more liberal in their predictions, this is, the number of false negatives increases. Just like it is shown on Figure 1, the SVC's have better performance than the rest of the classifiers.

In Figure 3 we can see that the tendency showed on Figure 2 increases, which means that, for the preprocessing and classification schemes presented on this work, incrementing the number of training classes affects them negatively.

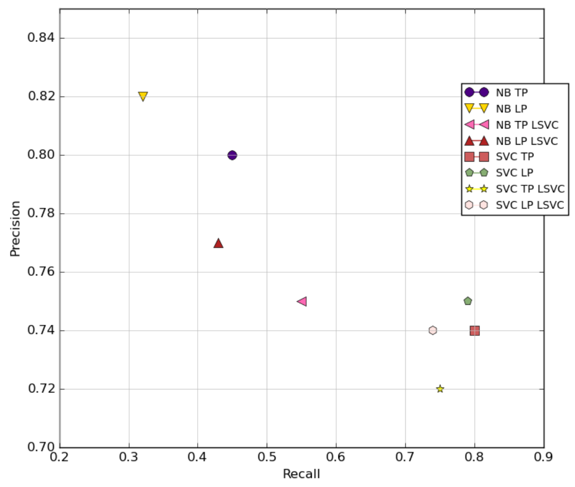

Fig. 1. Dispersion map for the 100 most frequent classes

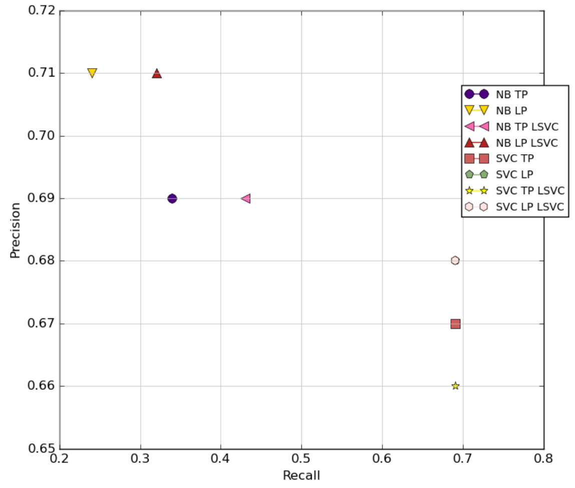

Fig. 2. Dispersion map for the 200 most frequent classes

These figures expose that, even though the Naive Bayes classifiers with the 5-tag classifier system show good precision values, their recall is very low, which leads to poor F1 values; on the contrary, the SVC's have better recall, although their precision is a bit lower compared to the Naive Bayes classifiers. For all of the classifiers their precision and recall values decrease when the number of training tags is increased, but this behaviour could be happening because of the low support present for certain classes in the data set.

Figure 4 shows the F1 values of the NB and SVC's. As previously stated, the NB based classifier has a lower F1 value than the SVC based scheme. In both cases, their F1 score decreases as the number of training tags increases. Interestingly, the SVC's do not show any improvement by using the LSVC feature selection method.

Table III shows the results of the developed system against the systems made by Parikh [6] and Schuster [15]. Both of

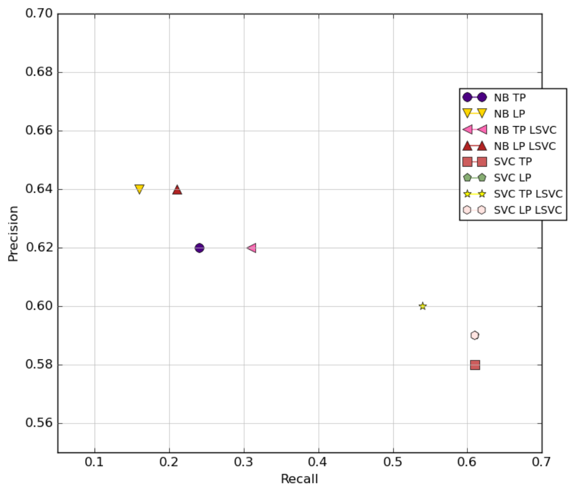

Fig. 3. Dispersion map for the 500 most frequent classes

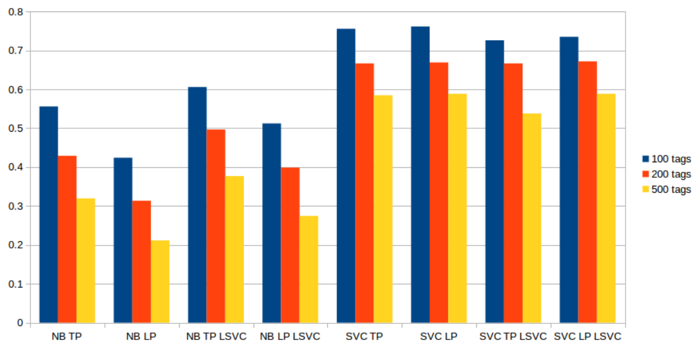

Fig. 4. F1 values of the 5-tag classifier system

TABLE III. COMPARISON AGAINST OTHER SYSTEMS

metrics	Parikh	Schuster	5-tag-NB	5-tag-SVC
precision	**0.64**	0.35	0.57	0.59
recall	0.25	0.58	0.31	**0.61**
F1 value	0.36	0.43	0.39	**0.59**

these systems used the top 500 tags, and accordingly, we show the results of the 5-tag scheme trained with these tags. Even that Parikh's has the best precision, the recall value obtained by their system is not high enough to allow it a better F1 value; curiously, the opposite happens with Schuster's. On contrast, the system that employs the SVC possess the right balance between precision and recall to achieve a superior F1 value with respect the other systems (even the based on Naive Bayes).

V. Conclusions and Future Work

This work shows that we can achieve good results using a simple classifying system, based on publicly available scientific libraries. This proposal presents certain advantages that are not available in other schemes. One of these advantages is that classifiers can be tuned respect to the tag they try to predict. That is, if it is known that there is high probability of a tag to appear in a position, it is possible to train the classifier with certain bias, which is not possible to attain with the developments discussed in Section I.

It is necessary to state that no syntax nor lexical analysis was included in this project, and yet it was possible to obtain favourable results by employing only the words as characteristics. Which indicates that, although this kind of analysis can result useful, it is not obligatory to include them when dealing with text.

The scheme presented obtained good results, mainly caused by the recall scores (and in lesser way by the precision scores) achieved by the classifiers of the third to the fifth tag, who were extremely good at identifying the empty tag, which has a high presence on this tag positions.

As showed in the results, there it was almost no difference between the use of feature selection methods and without them, which questions the effectiveness of the preprocessing task. More importantly, it shows that a better way to preprocess text is simply to discard the most frequent and less frequent words, rather than a full analysis of the text content.

This work showed that, even though the results obtained are not the best, they are competitive. Also most notably, the classifying systems presented are quite simple compared to other works for the same problem, where more computer power was needed to make and prepare their predictions; and with this approach it did not required more than easily available statistical tools.

References

[1] A. Goldbloom, "Kaggle," http://www.kaggle.com/c/facebook-recruiting-iii-keyword-extraction, 2013, [Online; accessed 13-November-2013].

[2] J. Attwood, "StackOverflow," http://www.stackoverflow.com, 2013, [Online; accessed 13-November-2013].

[3] J. Hong and M. Fang, "Keyword extraction and semantic tag prediction," 2013.

[4] A. Rajaraman and J. D. Ullman, *Mining of massive datasets*. Cambridge University Press, 2011.

[5] C. Stanley and M. D. Byrne, "Predicting tags for stackoverflow posts," in *Proceedings of ICCM*, 2013.

[6] C. Parikh, "Identifying tags from millions of text question," 2013.

[7] Z. S. Harris, "Distributional structure." *Word*, 1954.

[8] J. B. Lovins, *Development of a stemming algorithm*. MIT Information Processing Group, Electronic Systems Laboratory, 1968.

[9] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *The Journal of Machine Learning Research*, vol. 3, pp. 1157–1182, 2003.

[10] J. Zhu, S. Rosset, T. Hastie, and R. Tibshirani, "1-norm support vector machines," *Advances in neural information processing systems*, vol. 16, no. 1, pp. 49–56, 2004.

[11] S. Russell, P. Norvig, and A. Intelligence, "A modern approach," *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs*, vol. 25, 1995.

[12] C. D. Manning, P. Raghavan, and H. Schutze, *Introduction to Information Retrieval*. New York, NY, USA: Cambridge University Press, 2008.

[13] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *Proceedings of the fifth annual workshop on Computational learning theory*. ACM, 1992, pp. 144–152.

[14] C. J. V. Rijsbergen, *Information Retrieval*, 2nd ed. Newton, MA, USA: Butterworth-Heinemann, 1979.

[15] S. Schuster, W. Zhu, and Y. Cheng, "Predicting tags for stackoverflow questions," 2013.