Spam SMS Filtering Using a Multiclass Linear Classifier

Ahmet Gazi Cifci, Zabihullah Siddiqullah
Department of Computer Engineering, Eskişehir Technical University, Eskişehir
{ahmetgazicifci,zabihullahsiddiqullah}@eskisehir.edu.tr

***Abstract*—Unsolicited text messages, otherwise known as spam messages, have become a nuisance in all aspects of our lives. From marketing to phishing attacks, spam can be anywhere on the scale from a minor inconvenience to a major security and privacy concern, depending on the type of spam, the content, and the context in which it takes place. In this paper, we set out to tackle one of the many forms spam takes; Short Message Service (SMS). Many spam detection and filtering models have been proposed to solve the problem, but many of the existing models fall short when applied to languages different than their target ones. In this study, we implement a spam SMS filter for Turkish language messages. To classify spam from legitimate messages, we propose a multiclass linear classifier in MATLAB software. The results show significant increase in accuracy of the classifier when proper preprocessing techniques are applied.**

Keywords; Spam, text analysis, sms filtering, linear classifiers

I. INTRODUCTION

Texting remains as one of the most popular ways to keep in touch. It is also one of the top platforms for companies to target for marketing. Spam messages are generated and sent to mobile phones to market products or services. Spam text messages can be a nuisance when they create unwanted notifications for phone users. Spam may also be more than a nuisance in the form of phishing attacks on unsuspecting persons to steal their credentials or personally identifying information. Therefore, spam filtering text messages can be an essential feature on many devices.

Spam filtering falls under the category of text analysis and classification. Because languages can differ widely in their structures and grammars, creating a one-size-fits-all model for filtering spam messages in all languages has not been developed yet. While much study has been done in text analysis for the English language, the same cannot be said for Turkish. In this paper we try to mitigate this problem by implementing a spam SMS filtering system for messages that are in the Turkish language.

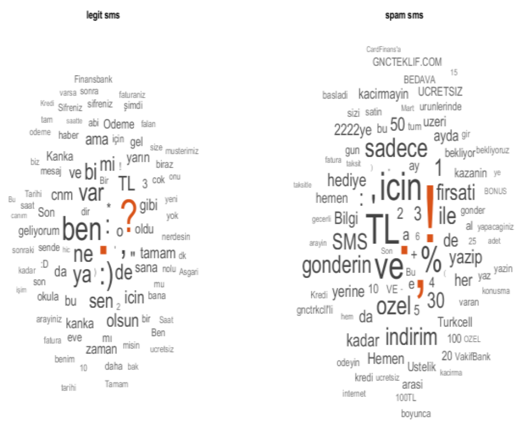
II. DATASET

The dataset used in this paper has two text documents containing SMS messages; one with 420 spam messages, the other with 430 legitimate messages. All messages are real and have been collected from our peers and academic staff. Since the messages are real and mostly between in non-formal settings, a lot of grammar rules and spelling conventions can be assumed to be circumvented. This may affect the classifier's performance if, for instance, all the legitimate messages are in informal settings, while the spam messages are formal. The classifier can be lead to find patterns that are not always reflecting the ground truths, such as classifying all messages with a formal and correct grammar as spam.

III. METHODOLOGY

A. Preprocessing

Preprocessing is usually carried out on datasets to make the data more relevant to the model, remove noise, and isolate patterns of interest from the background. Preprocessing provides a more compact representation of the data. In text classification tasks, like spam filtering, preprocessing usually translates to finding meaningful units of text using methods such as finding typical choices of words, sets of words, bags of words. It could also mean analyzing phrases in context of syntactical phrases such as noun phrases, statistical phrases such as frequent pairs of words. The preprocessing steps also include tasks such as stop-word removal, which is the process of removing very common words like articles, prepositions, conjunctions, adverbs, etc. from the terms to be considered for the classification. Other preprocessing steps often used include stemming, that is considering only the roots of terms and not their derivatives. The feature selection method employed can greatly affect accuracy and precision of classifiers. Fig.1 below show the word-cloud data representation of both the text documents before any preprocessing is applied.



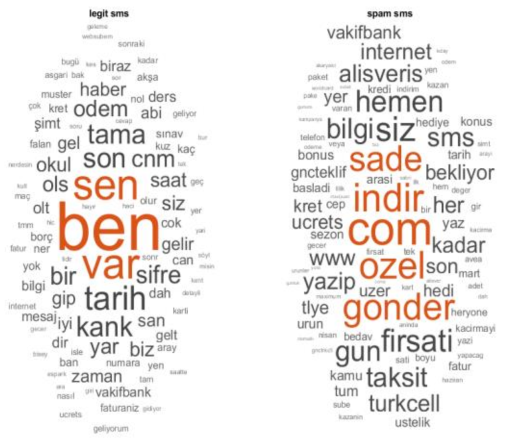


Figure 2. Word-cloud representation of the classes after preprocessing

As seen, there are a lot of punctuation marks, many numbers, and other insignificant data that can add to the classifier's complexity and lead to poor performance issues. To avoid this, we process the text using the previously mentioned techniques.

In this work, we make use of [1] to stem the SMS text. While MATLAB provides a built-in stemmer function, it does not support Turkish language. The stemmer application takes an input word and gives the stem as the output. Giving as it only functions on single words, we had to make a Python application that runs the stemmer on every term in the input file, which contains all the SMS text and provides a "cleaned" version of the file. After the stemming is carried out, the cleaned file is read by the MATLAB for further preprocessing steps to be carried out.

For any further preprocessing to take place on the data, it needs to be tokenized. Tokenization is the process of breaking the text data into its components, i.e. words. Text can be tokenized using the *tokenizeDocument()* function.

Stop-word removal is the next step applied to the data. Again, since MATLAB does not support Turkish Language, the stop-word removal step needed to be carried out in a custom fashion. First, we defined a set of stop-words using [2]. Then using MATLAB's *removeWords()* function, we clean the document from the stop-words. Next, we use the function *erasePunctuation()* and remove all punctuation from the text. This built-in function works, because punctuation is almost universal, especially among languages with the Latin script, which includes Turkish. Finally, we use the functions *removeShortWords(x, m)* and *removeLongWords(x, n)*, where x is the document with the messages, m and n are upper and lower limits, respectively, on the lengths of the terms allowed to pass through. Fig.2 shows the result of the preprocessing on the two classes. As can be seen, messages are better represented than before.

B. Feature Selection

After preprocessing is complete, comes feature selection which is another important task before classification can begin. Feature selection can affect the training time and computing power required significantly. It can also help avoid overfitting or underfitting the model. In text classification, two methods of feature selection used include Information Gain (IG) and Document Frequency (DF). The IG feature selection method is based on a notion of the amount of information the presence or absence of a term contains about the class of the document [3]. The DF of a term is the number of training documents in which it appears [4]. For the purposes of this paper, we have decided to implement DF, because of its simplicity and MATLAB's built-in functions that provide all the legwork for completing the task. To use DF, we simply create a bag of words from our preprocessed document, and then pass it to the *removeInfrequentWords(bag, 2)*, where all terms occurring in 2 or fewer documents are removed from the bag.

C. Classification

Text classification can be either multi-class or binary. Multi-class classification can be utilized to properly categorize news content, for instance, where each news item may fall into one or several categories. Spam filter is a binary classification problem, since we are only concerned with whether a text message is spam or not. Many classifiers exist that support text classification models, including Naïve Bayes, Decision Tree, and Support Vector Machines (SVM). MATLAB provides a function called *fitcecoc()*, which supports multiclass models for SVM or other classifiers [5]. The function returns an error-correcting output codes (ECOC) model using any specified learner. In our work, we have decided to go with a linear classifier since our data are linearly separable. By specifying the linear learner, by default, MATLAB uses SVM as a binary classifier.

D. Evaluation

Evaluation is a crucial segment for creating effective classifiers. Two main methods of evaluation include the k-fold cross validation and the holdout method. In our work, we use the holdout method of evaluation. The holdout method partitions data using set ratios for a training and testing set. A common approach is to use 70% of the data for training and the remaining 30% for testing the model. We have done similar in our work, using MATLAB's *cvpartition()* function.

IV. Results

After the classifier is trained, the testing set is used for evaluation of the model. Four different metrics are used to measure the performance; accuracy, precision, recall, and f-score. All the metrics can be calculated using a confusion matrix which is provided by the function *confusionchart()*. Fig.3 provides the confusion matrix for the model created using the test dataset.

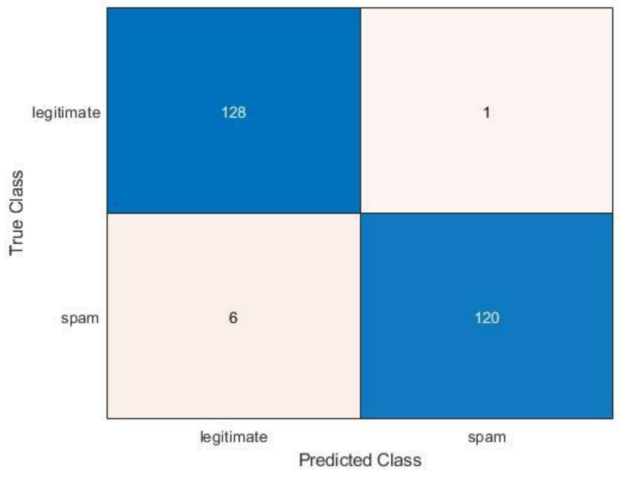


Figure 3. Confusion matrix for the model.

From the above matrix, we can determine the performance metrics using the following observations:

True Positive (TP) = 120

True Negative (TN) = 128

False Positive (FP) = 1

False Negative (FN) = 6

And then we calculate metrics using the above values, based on the definitions of the terms.

Precision = TP/(TP+FP) = 0.99%

Recall = TP/(TP+FN) = 0.95%

Accuracy = (TP+TN)/(TP+FP+TN+FN) = 0.97%

F-score = 2*(precision*recall)/(precision+recall) = 0.97%

V. Conclusion

In our previous tests of trying different preprocessing methods and gauging their performances, we tried to only use the built-in MATLAB functions that were available for Turkish text and did not make use of the custom stemmer. The results we got were as follows - Precision:98%, Recall:88%, Accuracy:93%, and F-score:92%. This means, stemming, combined with the other preprocessing and feature selection methods can boost the performance of a classifier significantly.

References

[1] *Turkish Stemmer for Python*, https://github.com/otuncelli/turkish-stemmer-python

[2] 'Turk Dili ve Edebiyati', https://www.turkedebiyati.org/Dersnotlari/sozcukturleri.html

[3] Y. Yang and J. O. Pedersen, 'A Comparative Study on Feature Selection in Text Categorization', in *ICML*, 1997.

[4] A. Dasgupta, P. Drineas, B. Harb, V. Josifovski, and M. W. Mahoney, 'Feature selection methods for text classification', in *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '07*, San Jose, California, USA, 2007, p. 230.

[5] *fitcecoc*, https://www.mathworks.com/help/stats/fitcecoc.html