Devoir 1

Gustavo Alonso Patino

Abstract— **In this work we present the performance of the SVM and adaboost methods applied to a classification problem of three classes. The data set consists of texts from blogs, so data preprocessing and word embedding was carried out before applying the classification algorithms. The data preprocessing is performed trying to clean the corpus of noise and unnecessary words that are not useful in the classification. Additionally, considering that the texts come from three classes of people with different ages, we tried to identify possible grammatical expressions and structures characteristics of each group in order to help in the classification. After preprocessing, we transform the data set into a vector form using two methods: the TF-IDF and the doc2vec. From the results it is concluded that the most efficient method for classification was the SVM with doc2vec, while the SVM with TF-IDF had a slightly lower performance. This result is interesting since the SVM with TF-IDF has a lower computational complexity than SVM with doc2vec. Finally, we observe that the algorithm with the lowest performance was surprisingly the adaboost with TF-IDF.**

I. Introduction

The SVM and boost techniques have been the focus of several studies in the literature, being successfully applied in the solution of several problems [3]. In both methods, the margin problem is treated mathematically. In the case of SVM, it is related in maximizing the separation between classes, using a kernel trick that maps the samples from the original space for a larger space where the data is easier separable. The original version of SVM was developed by [4]. Subsequently [5] adapted this version for a non-linear formulation. Originally SVM emerged as a binary classifier, however it was adapted for multiclass classification [6]. The way in which the muticlass problem is solved is to optimize a problem of the form

$$\min \frac{1}{2}\sum_{k} w_k^T w_k + C\sum_{(x_i,y_i)} \zeta_i$$

where (x_i, y_i) represents the training data, C is a penalty parameter and ζ is a constraint that meets

$$w_{y_i}^T x - w_k^T x \leq 1 - \zeta_i$$

In this manner, the SVM method is concerned in finding the optimal hyperplane that separates the classes. On the other hand, the boosting method was initially introduced by [7] as a technique for boosting the performance of a weak classifier. This technique was subsequently developed by [8], [9] . Later, the adaptive boosting method (adaboost) [10] emerged. In this method a strong classifier H whose discriminant function is f is obtained by training various weak classifiers $h_1, h_2 ... h_t$ with a "weak" learning algorithm as

$$f(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$$

Indirectly does something similar to SVM by minimizing a cost function related to the margin [11]. In other words, in this method a collection of weak learners are created by calculating a set of weights on the training samples in each iteration, and adjusting their weights to create a strong classifier from the linear combination of weak classifiers. The weights of the examples missclassified by the learner are increased while the examples that are well classified are decreased [11], improving their performance. The adaboost method also has several limitations especially due to the computational complexity and the loss of interpretability [12] which means that we cannot discover the importance of the features in the method.
In this work we compare the SVM and adaboost method in a classification problem of three classes of a large data set. As the data comes from a blog, the samples are texts that must be properly transformed to real number vectors using word embedding (II-C). To make the word embedding two techniques are used, the TF-IDF considering the occurrence and frequency of a word in a text and the most robust method doc2vec.

II. Data preprocessing

A. Data

We worked with a corpus obtained from the site *blogger.com.* This corpus consists of 512628 comments labeled with the numbers $0, 1, 2$ which correspond to the age range of the person who wrote the comment. 0

Class	Comments	Related to total
0	180158	35.14 %
1	240715	46.95 %
2	91755	17.91 %

TABLE I

CORPUS DIVIDED BY CLASSES

is related to younger people, 2 to middle-aged people and 3 to adults of more than 30 years old. Initially, the corpus has the labels mixed, so in the first step a computer code was written with the aim of analyzing the distribution of the comments. Table.I shows that the comments are not well balanced especially in the case of class 3 which has just 17.91% of the comments in relation to the total number of the corpus

B. Cleaning the data

In natural language processing NLP many texts contain words that are redundant for a classification task such as: mis-spellings words, slangs, stopwords etc. In this section we explain the techniques used to do the preprocessing of the data, or in other words the process of cleaning the data. This cleaning is extremely important for the classification process and will be discussed with some detail. The steps implemented in the preprocessing are:

- Identification of characters {"!!", "..." } and replaced by {STRONG!, DOTS}. This part is important as we saw in TP2 people from class 0 and 1 have the tendency to use these type of expressions frequently than class 2, helping to the classification
- Identification of expressions of time in the formats {digit:digit pm, digit:digit am, digit pm, digit am } and replaced by {TIME}
- Identification of expressions {hahah, heheh } and replaced by {RIRE } this helps a lot with the classification between the three classes.
- Tokenization: Breaking down a stream of text in words and symbols.
- Capitalization: Change the words exclusively to lowercase.
- Alpha words: Check if the tokens are alphabetic. In this process we select only the words of the text.
- Lemmatization by adverb, verb, noun: Eliminating redundant prefix and suffix of a word and extract the base word
- Remove the stepwords using the dictionary of *NLTK*

C. Word embedding

The word embedding processing allows the unigrams obtained from cleaning process to be translated into numerical values (vectors) that are understandable to the computer. The first approach we can make is to use the TF method, where each word is mapped to a real value that corresponds to the ratio between number of occurrences of that word compared to the total number of words in that document

$$Tf_{ij} = \frac{n_{ij}}{\sum_k n_{ij}} \tag{1}$$

Although this approach seems quite intuitive it has the problem of giving enormous value to words (which are not stopwords) quite used in the literature that do not help in the classification process. An alternative to outline this problem is to use the IDF model, in this method, we rate the words according to the number of times it is seen in different documents:

$$Idf = \log \frac{N}{df_i} \tag{2}$$

where df_i are the number of documents containing the word i and N the total number of documents. According to Eq.2 a word that rarely appears in the corpus has a high value, while a very common word has a low value because it must appear in many documents (being less interesting in the classification). Finally we can define the numerical matrix of words TF-IDF as

$$w_{ij} = Tf_{ij} * \log \frac{N}{df_i} \tag{3}$$

In the results section we can see the performance of the classifiers using this approach. The TF-IDF model provides, in many cases, good results when applied to classification problems, although it is a simplistic method since it can not model the contexts of the words in a text. The word2Vec [1] and doc2vec [2] methods are much more sophisticated techniques that try to model word contexts. In word2vec a vector representation (context vectors) of a word is made allowing to capture its relationship with other vectors in the space, for this purpose word2vec uses two algorithms: the continuous bag of words (CBOW) and the skip-gram model. CBOW creates a sliding window around a current word to predict it from context (the surrounding words). In skip-gram a different approach is made to CBOW, instead of predicting each word, a word is used to predict all the words that surround it (context words).

In the doc2vec method a numerical representation of a document is created, however, unlike the words the texts do not come in logical structures, then [2] uses the representation of the word2vec model. In the results section, we can also see the behavior of the classifiers using the doc2vec model already implemented in the *gensim* python library.

III. Experiments

A. Tuning doc2vec

In the doc2vec method, we try to predict a word distribution of a paragraph from a word taken randomly from the paragraph, for this purpose the vectors are obtained by training a neural network. The hyperparameters varied in the method are:

- dm: We can choose between 0 (distributed bag of words method) or 1 (distributed memory). The value of 0 was chosen
- Dimension of the vector space: In doc2vec a representation of the texts is made through vectors whose dimension can be chosen. The dimension was varied with values in the set {100,200,300,400}. It was found that from 300 to 400 there was no significant gain in classification accuracy using SVM. So we choose the value of 300.
- min count: In this parameter it is decided to ignore words that have a frequency less than the set value. A value of 3 was chosen.
- alpha=0.07: Value given to the stochastic gradient descent.
- number of epochs: Loop values were chosen in the range {10,20,30,50}. According to the experiments after 30 no significant improvement in accuracy was found using SVM classification.

Given that the doc2vec method has many hyperparameters, it was not possible to make a parameter optimization grid since the number of possible combinations is huge. Therefore, the values obtained do not necessarily represent the optimal values, but the values with the best performance in the chosen experiments.

B. Tuning hyper-parameters SVM, adaboost

The methods used in this work to make the classification task are: support vector machines (SVM) and adaboost. In general terms, SVM is an algorithm that maximizes the distance between sample points and a hyperplane. In order to optimize this distance, the SVM algorithm requires several hyper-parameters to be optimized, in our specific case we decided to use a linear kernel, so we were focus in controlling just the penalty of the error C, avoiding underfitting or overfitting in the model . In order to optimize the value of C we execute several training task with values in the array $C = [0.01, 0.05, 0.08, 1, 10]$, arriving to the result that $C = 0.08$ provided the highest training accuracy.
In the case of the adaboost classifier, the most important hyper-parameters to be controlled are: the base estimator, the number of estimators, and the learning rate. The base estimator was not controlled and the method of DecisionTreeClassifier was chosen, this is the default parameter suggested in python *sklearn*. The number of estimators represents the amount of weak learners that must be trained iteratively and the learning rate represents the weights given to the weak classifiers. To choose suitable values for these two parameters, we put in a grid several values for the hyper-parameters and combine them in an object called *grid*:

$$grid = \{'n\ \ estimators' : [100, 200, 500], \tag{4}$$
$$'learning\ \ rate' : [.001, 0.01, .1]\} \tag{5}$$

then the *GridSearchCV* function of *sklearn* was used to execute a classification task for each cross-value. After running the program we get the following result: $\{'learning\ \ rate' : 0.01, 'n\ \ estimators' : 300\}$ for the best score.

IV. Results

We show the performance of the SVM algorithm and adaboost with the preprocessed data (II) and with word embedding (II-C), the metrics used to evaluate the performance are: precision, recall, and F1. Considering Y_i as the true label of the example i, and $y(x_i)$ as the predicted label done by the classifier, and n the number of examples in our data base, the metrics in the multiclass classification are defined as:

- Precision:
$$\frac{\sum_{i=1}^{n} TP_i}{\sum_{i=1}^{n} TP_i + \sum_{i=1}^{n} FP_i} \tag{6}$$
- Recall:
$$\frac{\sum_{i=1}^{n} TP_i}{\sum_{i=1}^{n} FN_i + \sum_{i=1}^{n} TP_i} \tag{7}$$
- F1:
$$F1 = 2 * \frac{precision * recall}{precision + recall} \tag{8}$$

With the definitions TP: true positive, TN: true negative, FP:false positive, FN: false negative. For the SVM method we use two types of word embedding,

Class	Precision	Recall	F1
0	0.00	0.00	0.00
1	0.47	1.00	0.64
2	0.00	0.00	0.00

TABLE II
PERFORMANCE DUMMY ALGORITHM CLASSIFIER CONSIDERED AS BASE LINE

the TF-IDF method and the doc2vec method with hyperparameters tuning as defined in III-B, III-A.

In Table.II we can see the dummy classifier. In this algorithm we always predicts the label of the most frequent class (class 1), for this reason it has high values of precision recall and F1 in this class and null in the others, and will be considered as the base line for our classifiers.

Class	Precision	Recall	F1
0	0.63	0.65	0.64
1	0.50	0.77	0.61
2	0.51	0.47	0.48

TABLE III
PERFORMANCE SVM ALGORITHM USING TF-IDF

The SVM with TF-IDF (see Table.III) has a better performance than the dummy classifier. The table shows that the SVM has a fairly high performance in predicting classes 0 and 1 and lower in class 2, this is probably because the classes are not well balanced and class 2 has much less examples than the classes 0 and 1 as shown in Table.I.

Class	Precision	Recall	F1
0	0.64	0.68	0.66
1	0.58	0.86	0.69
2	0.51	0.48	0.49

TABLE IV
PERFORMANCE SVM ALGORITHM USING DOC2VEC.

As can be noted in Table.IV there is also an improvement in the results with word embedding doc2vec when compared to TF-IDF, since TF-IDF has the problem of not taking into account the context of the words in the text. However, it is worth noting that the less complex TF-IDF method applied to the data set is not very far from the performance achieved with the doc2vec method with the hyperparameters chosen. The SVM with doc2vec is computationally more expensive because it is necessary to build the word vectors by an iterative method, although later this effort is partially compensated in the calculation of the hyperplane at SVM, since the number of features in doc2vec is defined beforehand (300 according to III-A) less than TF-IDF, where the size of the space is defined by the number of most frequent words found in the corpus.

Class	Precision	Recall	F1
0	0.67	0.55	0.60
1	0.49	0.86	0.63
2	0.50	0.45	0.47

TABLE V
PERFORMANCE ADABOOST ALGORITHM USING TF-IDF.

Table.V shows the performance of the adaboost algorithm in the classification problem. The adaboost method beats the performance of the baseline, however it is not better than the results obtained from the SVM classification. Generally, the adaboost algorithm with decision trees behaves better than SVM when its parameters are fully optimized, however its performance is seriously affected by the presence of noise in the training data and outliers, this fact or a problem in the optimization of the hyperparameters may be a possible explanation of the poor performance in our classification problem.

V. CONCLUSION

We have built a number of models to predict the class of a blog comment using two methods, the SVM (with word embedding TF-IDF and doc2vec), and the adaboost method with TF-IDF word embedding. Our best model was SVM with doc2vec. The trained models were characterized by classifying with good performance (according to precision recall metrics and F1) classes 0 and 1 although they were not very efficient in classifying class 2, probably because class 2 is unbalanced in the data set, so that the algorithms could not learn the characteristics of that the class in the corpus. The adaboost method with TF-IDF had the worst performance, which was surprising because this algorithm is quite robust and efficient. The adaboost method is very sensitive to outliers and noise in the dataset, this fact, or a problem in the tuning of hyperparameters could trigger this poor performance.

REFERENCES

[1] Mikolov,T. Chen,K. Corrado, G. Dean,J. Efficient Estimation of Word Representations in Vector Space. arXiv: 1301.3781. 2013.

[2] Mikolov,T. Quoc,V. Distributed Representations of Sentences and Documents arXiv: 1405.4053.

[3] Hearst,M.A. Support vector machines. IEEE Intelligent Systems, pages 18-28, JulyIAugust 1998.

[4] Vapnik,V. Chervonenkis, A.Y. algorithms for pattern recognition learning.Avtomat.Telemekh 1964, 25, 937–945.

[5] Boser,B.E. Guyon, I.M. Vapnik,V.N. A training algorithm for optimal margin classifiers. In Proceedings of the Fifth Annual Workshop on Computational Learning Theory, Pittsburgh, PA, USA, 27–29 July 1992; pp. 144–152.

[6] Bo,G. Xianwu,H. SVM Multi-Class Classification. J. Data Acquis. Process.2006,3,017.

[7] Schapire,R.E. The strength of weak learnability.Mach.Learn. 1990,5, 197–227.

[8] Freund,Y. An improved boosting algorithm and its implications on learning complexity. In Proceedings of the Fifth Annual Workshop on Computational Learning Theory, Pittsburgh, PA, USA, 27–29 July 1992; pp. 391–398.

[9] Bloehdorn,S. Hotho,A. Boosting for text classification with semantic features .In International Workshop on Knowledge Discovery on the Web; Springer: Berlin/Heidelberg, Germany, 2004; pp. 149–166.

[10] Freund,Y. Kearns, M. Mansour,Y. Ron,D. Rubinfeld,R. Schapire, R.E. Efficient algorithms for learning to play repeated games against computationally bounded adversaries. In Proceedings of the 36th Annual Symposium on Foundations of Computer Science, Milwaukee, WI, USA, 23–25 October 1995; pp. 332–341.

[11] Schapire, R.E. A Brief Introduction to Boosting. Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, 1999.

[12] Geurts,P. Some enhancements of decision tree bagging. In European Conference on Principles of DataMining and Knowledge Discovery; Springer: Berlin/Heidelberg, Germany, 2000; pp. 136–147.