COMP9318 Project Report

Team ID: Tom_and_Jerry
Team members: z5119536 Jinlong ZHANG
z5122763 Yirong CHEN

Abstract

In this Project, we are required to devise an algorithm/technique to fool a binary classifier named 'target-classifier'. The target-classifier is a binary classifier classifying data to two categories. The target-classifier belong to the SVM family. Based on characteristics of SVM and the sample data in 'class_0.txt' and 'class_1.txt', we tried seven different ways to modify the 'test_data.txt'. The result shows that using 'TFIDF' with ratio-based modification has the best effect on the output accuracy after modified 'test_data.txt'.

Introduction

An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.

Instances used to feed the SVM model in this project are text data. So words in each instance are features, and these features need to be transformed to vectors then can be used in SVM. There are many different ways to extract features, and in the scikit-learn, 'CountVectorizor', 'DictVectorizor', 'TFIDF' can be used to simply implement that. SVM puts less emphasis on common words coming from both classes, so that before training data, we do not need to preprocessing on those common words in both classes.

Methodology with Justifications

Method 1:

This is a very straightforward way to train a svm then modify the file. Since we know that the 'target-classifier' is in SVM family, so that we can use the training data to train a svm model. By using all the weight of the trained model, specific features(words) can be chosen to be deleted, added or exchanged.

1. We used binomial method to count the word frequency. For example, a sentence ‘today is is is a good day’. If we have a word list [‘today’, ‘is’, ‘a’, ‘good’, ‘day’, ‘me’] then the frequency vector will be [1, 3, 1, 1, 1, 0]. We used all the instances from ‘class_0.txt’ and ‘class_1.txt’ as training data. We extracted all the different words as features, then transformed each paragraph into the frequency vector. And we labeled vectors from ‘class_0.txt’ as 0 and vectors from ‘class_1.txt’ as 1.
2. Secondly, we used ‘train_svm’ function in ‘strategy’ class of the ‘helper.py’ to train the svm model. We used all the frequence vector from step 1 to feed the model. We used grid search to find the optimal parameters to feed into svm model. This model reaches the highest accuracy(0.515) when testing on the ‘text_data.txt’.
3. From the trained svm, we can get the weight vector and we then map all the weights to its corresponding words(features). The outcome is ‘feature_weight_dict’.
4. Paragraphs in ‘test_data.txt’ also need to be transformed to word list form(eg.. [‘today’, ‘is’, ‘a’, ‘good’, ‘day’]). For each word in the list, we will check if it is in the ‘feature_weight_dict’. If it is in the dictionary, the weight will be assigned to that word. Otherwise, the word will be assigned to weight ‘0’. Then we sorted weight-word list from largest weight to lowest. And we chose the first 20 words which has the largest weight as words used to do the deletion(modification).

Method 2:

In this method, we used bernoulli method to count the word frequency instead of binomial way in method 1. For the same example, a sentence ‘today is is is a good day’. If we have a word list [‘today’, ‘is’, ‘a’, ‘good’, ‘day’, ‘me’] then the frequency vector will be [1,1,1,1,1,0]. Except this difference, other aspects are the same as method 1.

Method 3:

There is also only one difference in this method compared to method 1. When we do the deletion in method 1, we chose 20 words with the largest weigh but there is not a good reason for that. ‘Class_0.txt’ has 360 instances while ‘class_1.txt’ only has 180 instances, so that there may have a potential unbalanced data problem. From this point of view, the modification can be done in a more reasonable way. We got a ‘ratio’ by using the following formula:

$$ratio = \frac{\#\, paragraph\ in\ class\ 0}{\#\, paragraph\ in\ class\ 0 + \#\, paragraph\ in\ class\ 1}$$

For example, the ratio of 360 class_0 files and 180 class_1 files is $\frac{2}{3}$. Then after $ceil(20 * \frac{2}{3}) = 14$ deletion, there will have 6 addition. These 6 words are those have the smallest weight.

Method 4:

Instead of simply using word frequency to generate vectors, TF-IDF is a way especially effective for word features. TF-IDF intends to reflect how important a word is to a document in a collection or corpus. The tf-idf value increases proportionally to the number of times a word appears in the document and is offset by the frequency of the word in the corpus, which helps to adjust for the fact that some words appear more frequently in general.

$$tfidf(t,d) = tf(t,d) \times idf(t)$$
$$idf(t) = log\frac{1+n_d}{1+df(d,t)} + 1$$

And in this method, we also naively delete 20 words with the largest weight.

Method 5:

Like method 4, method 5 also used TF-IDF, but in this method, we used ‘ratio’ to delete and add words to do the modification.

Method 6:

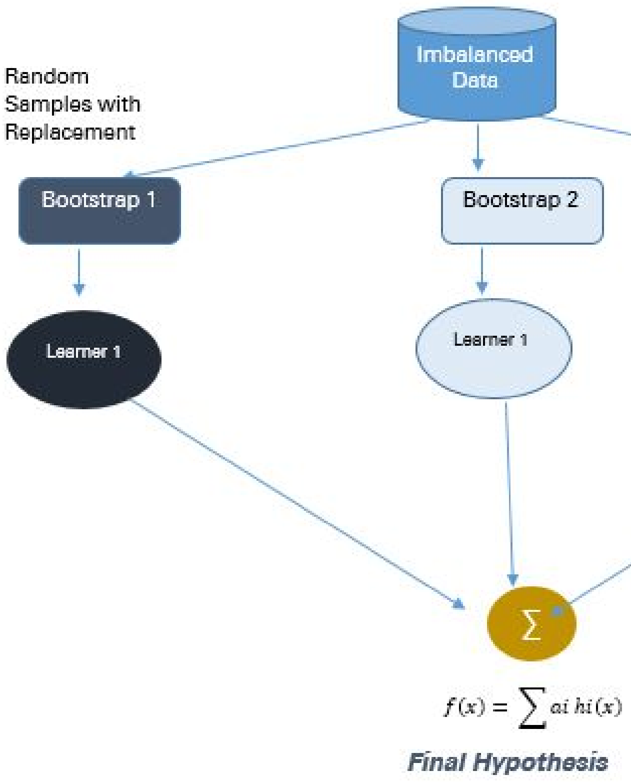
By doing some research, there is an improved TF-IDF keyword extraction algorithm called TF-IWF. Here is the formula for calculation TF-IWF. This algorithm assigns corresponding position weight to the words located in different position and calculates the words similarities with the same parts of speech which have a high counter in the result of the word segmentation, then merge the words with a higher similarity.

$$TF-IWF_{i,j} \rightarrow TF_{i,j} \times IWF_i = \frac{n_{i,j}}{\sum_k n_{k,j}} \times \log\frac{\sum_{i=1}^{m} nt_i}{nt_i}$$

The modification still uses delete 20 words with the largest weight.

Method 7:

Since ‘class_0.txt’ and ‘class_1.txt’ are imbalanced data, so we used Bagging whose full name is abbreviation of Bootstrap Aggregating. The conventional bagging algorithm involves generating ‘n’ different bootstrap training samples with replacement. And training the algorithm on each bootstrapped algorithm separately and then aggregating the predictions at the end. In our particular method 7, there are 6 different bootstrap training samples. Five of them are balanced data(eg.180 class_0 and 180 class_1, both are randomly selected) and one used all the data(360 class_0 and 180 class_1). And we used TF-IWF to deal with the features. Basically, 5 out of 6 training data are balanced data, so the 'ratio' = 0.5, and it means deleting 10 words with the highest weight then adding 10 words with the lowest weight.



(picture1: an example of bagging)

Method 8:

The result of using method 7 is not that satisfied, so we think maybe 6 bootstrap training samples is not enough and maybe using 'ratio' is not a reasonable way to do the modification. So we added 6 more bootstrap training samples, and deleted 20 words with the highest weight.

Method 9:

After tried many different methods, the result of using 'delete 20' and 'ratio' to do the modification is not stable. Since all the weights in different paragraphs are different, so it is better to try a dynamic way to do the modification. Here is how we did this method. Imaging the paragraph after transformation is [['i', 3.6], [['am', 3.4], [['happy', 1.6], [['finally', 0.9]], and the weight belongs to class_0(these words are used to be added) is [['less', -2.4], [['crazy', -1.3], [['blah', -1.2]]. For example, ['i', 3.6], the weight of word 'i' is 3.6. Firstly, we compare the absolute weight of 'i' and 'less' where $|3.6| > |-2,4|$, so we delete the word 'i'. Then, we compare 'am' and 'less' where $|3.4| > |-2.4|$, so we continue doing deleting on word 'am'. When we compare 'happy' and 'less', we found that $|1.6| < |-2.4|$, so we add the word 'less' to the modified file. Then we compare 'finally' and 'crazy', and so on. This method used TF-IDF to extract the features, and did not use bagging technique.

Result and Conclusions

(table 1: final result of using different methods)

Method	Description	Accuracy(submit system)
1	Binomial+DictVectorizor+Delete 20	84.5%
2	Bernoulli+DictVectorizor+Delete 20	84.5%
3	Binomial+DictVectorizor+'ratio'	72.5%
4	Binomial+TF-IDF+Delete 20	92%
5	Binomial+TF-IDF+'ratio'	92.5%
6	Binomial+TF-IWF+Delete 20	90%
7	Binomial+TF-IWF+Bagging 6+'ratio'	63.5%
8	Binomial+TF-IWF+Bagging 12+Delete 20	90.5%
9	Binomial+TF-IDF+Dynamic Modification	53.5%

For this project, we tried together 9 different methods. From the result we can tell that using binomial and bernoulli does not have a big different, but using TF-IDF to handle the feature extraction is much better than normal DictVectorizor. Using bagging technique or TF-IWF did no show a very high improvement. Using dynamic modification shown an dramatic decrease so that this method is excluded, but there is some uncertainties between 'delete 20' and 'ratio'. Considering that 'ratio' only shown better than 'delete 20' once which is when using method 5, 'delete 20' is better in other cases. Although that when using 'ratio' at method 5 has the highest accuracy, we still think using 'delete 20' is a more stable and secure way. Finally, we decided to use method 4 which is using binomial with TF-IDF and delete 20 words with the highest weight.

Reference:

[1] Wang, X., Yang, L., Wang, D. and Zhen, L., 2013. Improved TF-IDF keyword extraction algorithm. *Computer Science & Application*, *3*(1), pp.64-68.
[2] Martineau, J. and Finin, T., 2009. Delta TFIDF: An Improved Feature Space for Sentiment Analysis. *Icwsm, 9*, p.106.
[3] He, H. and Garcia, E.A., 2009. Learning from imbalanced data. *IEEE Transactions on knowledge and data engineering, 21*(9), pp.1263-1284.
[4] Shmilovici, A., 2009. Support vector machines. In *Data mining and knowledge discovery handbook* (pp. 231-247). Springer, Boston, MA.