9318 project report

z5042020 Shen-Chieh Chuang
Z5092923 Jintao Wang

1.Short Abstract

In this project, we need to implement an algorithm to fool the target classifier which means we need to let the target classifier misclassify our modified text from class1 to class0 and only 20 chance of modification are allowed in each case.We are given 2 training data(class-0, class-1) to train our classifier.

2.Introduction

In order to solve the problem, we use bag of word model, TF-IDF (term frequency–inverse document frequency) to reflect the weighting factor in each text and SVM training. Finally we modify the *test_data.txt* and put it in the target classifier to get the accuracy. We also discover the best way of modification for the sample text.

3.Methodology with justifications

In the first step, we count the number of each word from class0 and class1 independently and save it as our bag of word model.

```
    # count words
    for i in range(len_of_class0):
        for word in class0[i]:
            word_index = word_list.index(word)
            class0_word_count[i][word_index] += 1

    for i in range(len_of_class1):
        for word in class1[i]:
            word_index = word_list.index(word)
            class1_word_count[i][word_index] += 1
        y_train[i+len_of_class0] = 1
```

(part of code)

Then we import TfidfTransformer function from sklearn to calculate the tfidf value of each word in each paragraph. Finally we call the SVM training function in help.py to train the classifier. Here we use 'linear' for the kernel parameter. At last we obtain the weight of the classifier by using the attribute coef_.

```
tfidftransformer = TfidfTransformer()
tfidf = tfidftransformer.fit_transform(x_train)

parameters={'gamma':0.01, 'C':1.0, 'kernel':'linear', 'degree':3, 'coef0':3.0}
clf = strategy_instance.train_svm(parameters, tfidf, y_train)

# get weights of the classification
weights_list = clf.coef_[0].toarray()[0]
weights_dict = {}
for i in range(len_of_feature):
    weights_dict[word_list[i]] = weights_list[i]
```

It is important to save the weight with the key(word) in decreasing order to help us have a high accuracy after the modification. The larger the number, the more easily to be classified to class1. We obey this rule to modify our test_data.txt.

```
# sort weights
sorted_weights = sorted(weights_dict.items(), key = operator.itemgetter(1))
```

In the final step, We found the times of adding and deleting can effect the accuracy and we need to test the best combination to get the highest accuracy.

4.Results and Conclusions

Here is the experiment record.

Add	Delete	Accuracy
15	5	65%
17	4	42%
5	15	92.5%
3	17	89.5%

modified_data.txt submission.py	2018-05-21 13:30:56	Success = 84.000 % (Plz make sure that you do not use test data for inference)
modified_data.txt submission.py	2018-05-21 14:42:53	Success = 65.000 % (Plz make sure that you do not use test data for inference)
modified_data.txt submission.py	2018-05-21 14:50:45	Success = 42.500 %
modified_data.txt submission.py	2018-05-21 14:59:37	Success = 92.500 % (Plz make sure that you do not use test data for inference)
modified_data.txt submission.py	2018-05-21 15:11:34	Success = 89.500 % (Plz make sure that you do not use test data for inference)

After the experiment we find that the best way is adding 5 words and deleting 15 words in each sample. Finally, We can get the highest accuracy(92.5%).

In conclusion, we implement bag of word model, TF-IDF, SVM training and different times of modification to achieve the target.