COMP-9318 Final Project Report

Student 1: FUDI ZHENG z5126586

Student 2: QIAN ZHANG z5005027

Part 1: Short Abstract

In this project, we are required to devise an algorithm to fool a binary classifier, which can help to classify data two categories, calss-1 and class-0. Firstly, there are two types of txt file named class-1.txt and class-0.txt respectively. Secondly, SVM (support vector machines) are a set of supervised learning methods used for classification, which can be trained to classify calss-1 and class-0. Thirdly, test data which belongs to class-1 need to be modified to generate new data which should belong to class-0. Finally, trained SVM is used to test the success rate (how much chance does the modified data belong to class-0).

Part 2: Introduction

In order to solve the problem in this project, there are two important parts should be implanted including training SVM and designing an classification method.
To begin with, we will introduce that how we train the SVM. Next, the detailed algorithm which is used to fool a binary classifier will be explained.

Part 3: Methodology with justifications

1. the process of training SVM

Support vector machines (SVMs) are a set of supervised learning methods used for classification, regression and outliers detection.
However, it is difficult to for us to achieve SVM model. Fortunately, there is a SVM library in scikit-learn in python.
From helper.py file, we know that parameters, train x and train y need to be passed to train SVM.
There are five parameters needed to passed to train SVM. The first parameter is 'gamma' Kernel coefficient for 'rbf', 'poly' and 'sigmoid'. If gamma is 'auto' then 1/n_ features will be used instead. The second parameter is 'C' which is penalty parameter C of the error term. The third parameter is 'kernel'. It must be one of 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed' or a callable. If none is given, 'rbf' will be used. The fourth parameter is degree which is degree of the polynomial kernel function ('poly'). Ignored by all other kernels. The fifth parameter is 'coef' which is only significant in 'poly' and 'sigmoid'.
Train x includes 540 examples in the class-0 and class-1. Because class-0.txt and class-1.txt are provided files, we know there are 360 examples in class-0.txt and 180 example in class-1.txt respectively. Every line in the class-0 and class-1 file is regarded as a example. The first 360 examples are marked 0 and last 180 examples are marked 1.
Since every example has different length, same length of every example needed to be implemented to meet SVM library requirements.
The good method to address this issue is regulation which can make every example includes all unique words.
Example:
train_x_class0=[1,2,3,4,1,1,2,3],   train_x_class1=[2,4,5,6,2,4,6,8]
If we build a new list which include all train x data, we can get all unique words list.
train_x_all=[1,2,3,4,5,6,8]
In order to represent the same information , we need to count the number of specific word in class-0 or class-1 by using index. For example, the number of 1 is three in class-0, so the value of index '1' is three.

new_ train_x_class0=[3,2,2,0,0,0,0] new_train_x_class1=[0,2,0,2,1,2,1]

From the lecture note, we know that linear or polynomial kernels are usually used to do text classification. First, we use linear kernel to train SVM. Since linear kernel is relatively easy, most parameter are default value. Trained SVM model is used to predict test_data.txt. Although different parameters are used to train SVM, we found it has poor success rate about 0.5. As the test_data.txt belongs to class-1, the performance of linear kernel is not good. Second, we use poly kernel to train SVM. Many parameters can be changed to train SVM such as 'C', 'degree', 'gamma' and 'coef'. We use different parameters to train SVM many times and we get the good result. The prediction of test_data.txt is correct and it has a very high success rate. As a result, the performance of poly kernel is better than linear kernel.

2. an modification algorithm

There are four steps in the modification algorithm.

Step 1:

Processing class-0.txt and generating class-0 dictionary.

Firstly, building empty dict includes all unique words and value equals 0 named class0_dict_unsorted.

Secondly, counting the number of every unique word in class-0.

Thirdly, filling the index with corresponding frequents and generating new dict

Example:

class_0=[i love you very very much], class_1=[i hate you very much much]

class0_dict_unsorted={ i:0,love:0,you:0,very:0,much:0,hate:0}

new_ class0_dict_unsorted={i:1,love:1,you:1,very:2,much:0,hate:0}

Processing class-1.txt and generating class-1 dictionary

It is same as processing class-0.txt

Step 2:

Processing nonsense words e.g. 'a', 'the', 'an', ','…

Firstly, calculating the ratio of length of class-0.txt and class-1.txt

Secondly, traverse the all word and find the number of every word in class0 and class1

Thirdly, rate = (num_class0(specific_word)/ratio) / num_class1(specific_word). If 1<rate<1.25 and record the specific_word in rubbish_words(nonsense) dict

The rate upper bond is from tests many times and 1.25 is best upper bond so far.

Example:

Length of class-1.txt is 180 and length of class-0.txt is 360. Ratio is 360/180=2

'the' is 2168 in class0 and 'the' is 1039 in class1

rate=(2168/2)/1039= 1.0433108758421559

since 1<rate<1.25, rubbish_words={the:0}

Step 3:

Generating preferred class-0 dictionary:

Firstly, sorted class0_dict, sorted class1_dict

Secondly, for every word, the value in sorted class0_dict minus the value in sorted class1_dict and store this word and its minus value in class0_ preferred dict.

Thirdly, sorted the class0_ preferred according to its value high to low.

Example:

class0_dict={the:2168,in:957}, class1_dict={the:1039,in:490}

the:2168-1039*2=91, in:957-490*2=-23

class0_ preferred={the:91,in:-23}

Generating preferred class-1 dictionary:

It is same as generating preferred class-0 dictionary.

Step 4:

Modifying the test_data.txt according to preferred class-0 dictionary and preferred class-1 dictionary.

Firstly, process class0_preferred dict by deleting nonsense words in rubbish_words dict(nonsense words dictionary).

Secondly, traverse test_data.txt and replace top 10 frequents word in class1_preferred

Thirdly, generating new test_data.txt named modified_data.txt by replacing 10 words in every example.

Finally, trained SVM is used to predict modified_data.txt.

Part 4: Results and Conclusions

In summary, after testing modified_data.txt many times, 89.5% is the most high successful rate so far. The result can be vary due to the change of parameters and the optimization of modification algorithm. This project can not only help us build a better understanding of SVM classifier but also advance our problem-solving skills.