student id: c1954633 name: Rongjian Huang

Notice: Some of the code in lab were used in project, and the dictionary: "A list of English positive and negative opinion words or sentiment words (Hu and Liu, KDD-2004)" was used in this project.

Part2:

Sentiment analysis is the NLP technic to distinguish positive review and negative review. In order to do sentiment analysis on IMDb, I trained and tested a SVM model and several steps were taken.

(1) and (2)

1. Choice of features

I chose three things as the features of the SVM model which are:

-The total count of positive word and negative word based on sentiment lexicon vocabulary from Pros.Hu and Pros.Liu

-The frequency of single adjective and verb words

-The length of reviews

Using count of negative and positive word can be quite useful, we can easily classify a negative reviews with lots of negative words or a positive reviews with plenty of positive words. It do have a good prediction when I used only this feature on development set with 72% of accuracy. However, this dictionary is not from the dataset and I think some of the informational pattern my appear only in the dataset. So I decide to use training set to create a dictionary to make it more "localize". And I choose the frequency of adjective and verb in training set. In the lab of the lecture, lecturer create a dictionary for all words. But I think the most representative and straightforward words in a sentence must be adjective and verb while adverb and other words are relatively less important. Once I have my original model and new model with two features run on development set, I find out the accuracy have increased to 82%. Finally, I think the distribution with length of reviews might be slightly different between positive and negative reviews. Not surprisingly, adding this features to model only increase 1-2% of accuracy.

2. Preprocessing

In this part, I first load the dataset and sentiment lexicon vocabulary from local files which are train set, test set, development set, "positive-words.txt" and "negative-words.txt". Then I lemmatize the word and transform them to lower case, and ignore the word in "stop-words". After that, with the help of Pos Tagging, I generated an adjective and verb dictionary with size of 1000 according to their frequency in 1000 shuffled reviews in training set. Once I have the dictionary, I can transform the reviews into frequency vector. Then I counted the total frequency of positive word and negative word respectively for every review based on the sentiment lexicon vocabulary. At the same time, I counted the length of reviews. Finally, I combine these three vectors and normalize them, so the features are all between [0,1]. Then I return the vector to do feature selection.

I have two reason for why I chose 1000 word in only 1000 reviews. First, it is time-consuming as the program read the whole training set to create a dictionary. It take 13s to create dictionary based on 1000 reviews and 68s for 4000 reviews, shorten the time is one of my objective. Second, there are only slightly different between reading 1000 reviews and 4000 reviews to create dictionary. Only some sequences of dictionary and some words with less frequency will change slightly which are not important to result.

3. Feature selection

After 2nd part, I have my preprocessed features vectors. However, some of the features may not be relevant to the labels. For example, "see" is third most frequent verb but it can be in both

positive and negative reviews. So, I create an selectKBest object to select features with 503 best ANOVA F-Values using f_classif in sklearn. And the selectKBest object can be used to transform any preprocessed vectors (including training set, development set and test set) to the vectors after feature selection. So my feature dimension decrease from 1003 to 503(2:count of pos&neg word, 1:review length, 500: adj&verb frequency).

In the lab, lecturer recommended us to use chi-square testing to select features. In my not rigorous experiment (since I run program many times instead of writing some compare function in development set), chi-square testing takes more time than f_classif testing and have a lower performance that is approximately 5%. So I choose f_classif testing rather than chi-square testing.

4. Training data and testing data

Before I start training my model, first I am going to find out the best parameter of my model. I trained some SVM models with different parameters including Kernel, C value, and gamma in the development set which returns the best parameter. With these optimized parameters, I start training my final SVM model. Once I have my trained model, I go testing my model with test set and generate a classification report with precision, recall, f-measure, and accuracy.

In development set, I split it to dev_test and dev_train set in 0.8 ratio, for all parameters combination I trained a SVM model to find out the parameter with higher performance on accuracy.

(3). Overall performance

	precision	recall	f1-score	support
0	0.85	0.82	0.83	2502
1	0.82	0.86	0.84	2500
accuracy			0.84	5002
macro avg	0.84	0.84	0.84	5002
weighted avg	0.84	0.84	0.84	5002

(4). Critical reflection

Here are the time of the program to use:

initialize time: 0.16392993927001953
generating vocabulary: 12.701534032821655
vectorize time: 294.58524107933044
feature selection time: 0.2778339385986328
select parameter time: 160.0605571269989
training time: 71.73239088058472
testing time: 115.89063096046448
total time: 655.4121179580688

As it take 10 mins and sometimes even more time to run the program, debugging has became a nightmare of this project. So I would like to improve the algorithm in terms of time. As we can see, it take too much time to vectorize which is about 5 mins. Maybe it's because list in python is less efficient, using Numpy and pandas library to read data may be faster. Also making it a parallel program can be a good way to improve the speeds. For example, using distributed computing framework like Map-Reduce, Spark and storm. So that I can generate a bigger dictionary which means more features and higher accuracy.(In testing, dictionary with size of 3000 and feature size of 1500 can increase the accuracy to 87%).

People tend to think machine learning algorithms are objective, because it's only math and functions rather than opinion. However, the truth is learning algorithm are designed by developers and scientist. They are human and because of culture and environment differences, bias can be invisible and inevitable. With the help of this SVM model, social media company like Twitter can easily detect hateful tweets and punish the author. If the model misclassify the result, it doesn't matter. But when it comes to judging crime and moral, can it maintain justice and useful as usual? What if it misclassify the result? This can be crucial. So balancing the usage and the negative consequences can be difficult.

Extra:
A GitHub repository: https://github.com/Alphver/ml_coursework.git