PCML Project 2 Report - Text Classification

TeamNX – *Jiacheng Xu, Shiyue Nie*

I. Introduction

The task of this project is to predict if a tweet message used to contain a positive or negative smiley, by considering only the remaining text. Therefore, the whole pipeline of our algorithm divides into two main stages: It can be classified by word embeddings (matrix factorization) with data pre-processing at first, and then applies to different training methods we have learned.

II. Word Vector

We first used the given evaluation pipeline to generate the word embedding. After running the *build_vocab.sh*, we got separated vocabularies by running *cut_vocab.sh*. Following the guidance, we then used *pickle_vocab.py* and *cooc.py* to generate the words co-occurrence matrix. To compute the embedding vector for it, GloVe is implemented as in *glove_solution.py*.

However, if we did nothing else to improve these steps, the training error of GloVe is too big that it converges very slow even with full training data. Two main improvement we make in these steps: Prepare the words pre-processing before generate the words co-occurrence matrix, and also pre-processing all training and test data for matching the matrix; implement SGD updates to train the matrix factorization.

A. Data Pre-processing

As the data collected from tweet, there are many vocabularies in wrong formats or useless, such as, some meaningless words shorter than 3 letters (a, an, to, in, on etc.), number and punctuation, format tags like $< user >$ and $< url >$, spelling mistake and informal words for social communication etc. Additionally, words with the same stem actually should be considered as the same word, so that the amount of vocabulary is reduced reasonably with stronger connection.

We write another python file for all these pre-processing steps, from words tokenizing (first seperates words and punctuation) to lemmatizing (after stemming, the stemmers can be translated into normal words by lemmatizer). Then step should be executed after running *cut_vocab.sh*. Once we adapt this pre-processing method to words co-occurrence matrix, we have to adapt it to all training data and test data so that vocabularies can match.

Unfortunately, when we use it to train our model, the result is worse than without pre-processing. Detailed checking, there are many short 'same' words after stemming, like 're', which is reduced by stop words but it occurs again with high frequency. This problem should be over-stemming. In addition, some short words after removing stop words are emotional, so we do not need to remove them. Informal words for social communication cannot be translated by lemmatizer easily. They may be changed into other words. Hence, we finally keep 3 processing steps:

- Tokenizing
- Keeping Words Starting with Letter: This step can clean punctuation, number and unusual vocabulary.
- Removing Stop Words: Despite of existing stop words, we add local stop words 'user' and 'url' as they are the label tags of the original text.

Specially, for the dictionary construction, we want to make sure that all vocabularies are unique and without blank, so after pre-processing, the dictionary is passed through *build_vocab.sh* and *cut_vocab.sh*. For training and test data, we just need to pass it once through pre-processing.

B. Matrix Factorization

After executing *cooc.py*, we get the original co-occurrence matrix. In order to get the approximate vector for every word in vocabulary, we train GloVe matrix factorization model. The GloVe model actually is a variant of of word2vec. It adds a weight f_{dn}:

$$f_{dn} := min\{1, (n_{dn}/n_{max})^{\alpha}\}$$

to each entry, representing importance of this word in the whole dictionary vector.

1) Matrix Factorization with SGD: For improving the accuracy of our word vector, we implement SGD update method with measuring MSE for each iteration round. To accelerate the training speed, we still keep the parameter $\eta = 0.001$ and the scale of MSE also drastically decreases. The formula of MSE is:

$$MSE := \frac{1}{2} \cdot \sum_{(d,n)} f_{dn} \cdot \eta \cdot [x_{dn} - (WZ^T)_{dn}]^2$$

Nevertheless, the biggest problem is how to choose the step length parameter γ: If it is too large, it would cause the weights to be divergent after a few times of iterations, and if it is too small, the convergence of the algorithm takes too many iterations, even not reaching convergence before stopping iteration.

2) Gamma Decreasing: To balance the trade-off of training speed and convergence, the γ in our implement algorithm is dynamic as following rule:

We choose a reasonable large γ at the beginning of the iterations and keep it as constant, which would not cause divergence in the certain times of iterations (what we set is 10 times), but still help us to decrease the MSE fast. After that, we hope to use a smaller γ to reach the convergence as precisely as possible, so the value of γ lowers in this way:

$$\gamma(t+1) := \frac{1}{1 + times \cdot \delta} \cdot \gamma(t)$$

in which $times$ is the current iteration round and δ is the controlled scalar. By doing so our process of matrix factorization can convergence faster and more precisely.

3) Matrix Factorization using External Library: Actually, the result of matrix factorization with SGD is not satisfied enough, as the calculation speed is slow and the MSE is too big (lessen with η, the minimal MSE is about 50). Consequently, we try to implement matrix factorization using *sklearn* library, which calculates the matrix much more quickly, and surprisingly, improves our prediction. So far, in this way, we generate word vector with 100, 200 and 500 features.

III. Learning Methods

The aim of our project is to predict the emotion of every tweet sentence, but what we have calculated so far is the word vector of all sensible vocabularies of all messages. In order to find the represented vector of each sentence, every word matched in the sentence is recorded its word vector. Then, the sentence vector is the average of all words' vectors of this sentence, which represents its features.

According to the project's suggestion, we train linear classifiers on our constructed features. Some typical linear regression methods we adopt, Least Square (LS) and Linear Regression (LR). Some other methods like Support Vector Machine (SVM) and Naive Bayes (NB) are traditionally suggestive to text classification problem. Moreover, we once want to implement Convolutional Neural Network (CNN), but time is too limited to learn new methods.

A. Principal Component Analysis

The scale of word vector we get is $D \times K$. D is the amount of unique vocabularies found trough all training data, and K is the amount of features as we set. To get a more accurate word vector, we typically set K as 50, which we once think is large enough. The larger K is, the heavier computational burden of each training methods suffer. Therefore, we wonder if Principal Component Analysis (PCA) is helpful to reduce the dimension and speeds up model training.

Oppositely, when we use cross-validation to check how many components are needed for the best prediction natively, the result shows that higher dimension improves our model training. For $K = \{20, 50, 100, 200, 500\}$, we change the kept components from 1 to K, the tendency of model score is that the best result occurs when kept components is nearly equal to K. This phenomenon alarms us that K is still too small!

As the figure 1 and 2 show, we can found a positive correlation between local predict score in cross-validation and the number of features, so we choose all features we get from matrix factorization.

B. Least Square

Least square (LS) method is one of the most simple models to train the linear classifier. The reason why we choose it as the first implementation is because its fast calculation and fast checking the accuracy of matrix factorization. What's more, we also regard LS as the control group of other methods we have implemented.

C. Support Vector Machine

Support Vector Machine (SVM) is a representation of the examples as points mapped in space, so that the examples of the separate categories are divided by a clear gap that is as wide as possible. In our project, we match all word vectors corresponding to words in each sentence, which means we read every word in each sentence to record its corresponding vector. Then, we average all word vectors in each sentence to get the new vector of this sentence. Calculating like this, every tweet message can be regarded as a point in space to add a gap to divided the positive and negative tweet.

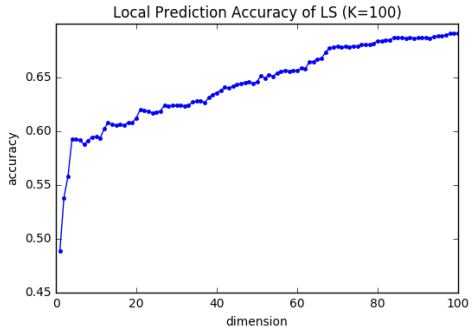

Fig. 1. The Cross-Validation Result of Least Square (K=100)

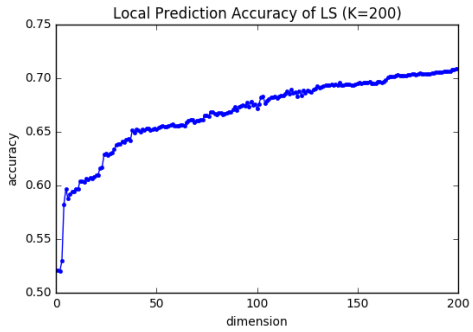

Fig. 2. The Cross-Validation Result of Least Square (K=200)

D. Naive Bayes

As it is suggested to solve text classification problem with Naive Bayes (NB), we try to apply Multinomial Naive Bayes (MNB) and Bernoulli Naive Bayes (BNB) and even Gaussian Naive Bayes (GNB) using parts of our data (e.g. data samples are randomly chosen 25,000 entries of the whole data set). Although they are proved to be suitable to text classification problem, they cannot fit our model well. Multinomial Naive Bayes requires no negative data, so we cannot apply it directly to our data. Overall, training on low dimension feature, the result of Naive Bayes is not good enough, even worse than least square.

IV. Results

A. Methodology

The methodology we follow to obtain our results is an extensive use of the cross-validation while varying all the other parameters. For LS method, we vary the dimension of features and do a lot of local predictions to find the best dim, and we also implemented a grid-search for SVM to find the best degree and penalty parameter C of the error term.

B. General Results and Comments

There are some results that we obtain from each method. Notice that these results come out from the local predictions by using 5-fold cross-validation. For the time limitation, some methods we only tried on the condition of $dimension = 50$ (such as LR, BNB, GNB) as their results are not satisfactory. The result is shown below (Table I):

Method used	LS	SVM		LR	BNB	GNB
Accuracy (%)	66.0	66.3	66.4	65.9	60	62
Degree	1	1	3	\	\	\
C	\	2.0	1.0	\	\	\

TABLE I

THE LOCAL PREDICTION USING 25,000 SAMPLES,50 FEATURE DIMENSIONS RESPECTIVELY LS, SVM, LR, BNB, GNB. THE DEGREE FIELD CORRESPONDS TO THE DEGREE OF THE POLYNOMIAL BASIS WE USED FOR THE PREDICTION, THE C REPRESENTS THE REGULARIZATION LEVEL,SMALLER VALUES SPECIFY STRONGER REGULARIZATION.

C. Change the Dimension

1) High Dimension Performance : For LS and SVM methods, which performed well in the lower dimension,we try to increase the dimension to find if the results could be better.Here we choose $dim = \{100, 200, 500\}$, and using cross-validation to do local prediction,the accuracy was enhanced both in LS and SVM method,but from the figure we can see the improvement of SVM is smaller than LS,and another reason for us to choose LS is that the calculation speed of SVM is really slow (Table II).

Dim	LS	SVM
20	54.4	54.4
50	65.8	66.3
100	68.9	67.8
200	71.0	\
500	73.4	\

TABLE II

THE LOCAL PREDICTION WITH DIFFERENT DIMENSIONS USING 25,000 SAMPLES, RESPECTIVELY LS, SVM.THE RESULTS ARE THE ONES WITH OBTAINED WITH A 5-FOLD CROSS-VALIDATION

2) Best Result: We finally choose LS for our test classifier,and we choose $dimensions = \{100, 200, 500\}$,here are our result on Kaggle (Table III). The final best result of $K = 500$ actually use 250,000 data sample because whole data is too large to train in time.

Dim	LS
20	54.3
50	63.0
100	66.5
200	71.0
500	74.4

TABLE III

THE PREDICTION ON KAGGLE,USING ALL 2500000 SAMPLES AND IMPLEMENTED BY USING LS METHOD IN DIFFERENT DIMENSIONS OF FEATURES.

D. Details of the Results

As shown above, the result of our project is not satisfied. One of the barriers is that we are struggling in matrix factorization. Every time we alter our algorithm or test the selection of γ, we have to run it again for iteration, which takes us hour after hour, even with 10% of all tweets. In addition, we change our data pre-processing criteria for times, which results in all data renew and vector recalculation.

Another problem is to make sure all details are exactly right, both in coding and data pre-processing steps. Actually we still confuse if we should execute lemmatizing to correct tweet's vocabularies or not.

Last but not least, it is a pity to figure out the dimension problem in the last day. We can just prove our assumption of it by doing cross-validation with data sample (10% of whole data set), and we apply high dimension data to least square to prove that higher dimension (so far less as 500) can enhance our prediction.

V. CONCLUSION

The most significant problem of our solution is that the dimension of word vector is not high enough, and it takes over hours and even over day to train the word matrix and classifier again! Unfortunately, we figure out this problem at the very last moment so that we do not have enough time to run through other training algorithms like SVM and Naive Bayes.

One possible improvement is to approximate our matrix factorization using tweet data provided on the GloVe homepage. Also, other public implemented methods may be more suitable to text classification than the ones in existing external library. More advanced methods like Neuron Network (e.g. CNN) could be useful to our project.