NLP Homework 2: Sentiment Analysis

Deng Wen 17307130171

Fudan University

2019/11/14

I. Introduction

In this homework, I need to use feature engineering and word2vec based models for sentiment analysis.

For feature engineering model, I need to implement feature extraction which is to the bag of words feature and use naive bayes classifier to do sentiment analysis.

For word2vec model, I am given a list of python files which already contains relatively complete codes for the model. All I need to do is to fill in several blocks of python code based on the properties of the model.

II. Fearture Engineering Model Based

i. Data

The data of training set, development set and test set I used are directly from the given file *data_utils.py* by coding *-from data_utils import ** . Then use the functions defined in the file to get the training set, development set and test set. (Details are in the code file.)

ii. Construct A Bag of Words Model

In the construction, except the training data, I added two more parameters: *negation_dealing* and *boolean*, which are all bool variables to decide wheter to deal negation(Add "Not_" prefix from negation to next punctuation) and using boolean Naive Bayes(Removing duplicates in each document) or not.

Then I defined a set to store all the words in the training data and calculated its number. Moreover, for each label(which is the document class in this problem), I defined a dictionary to store the count of each label, the words and their frequencis and the number of tokens in each dictionary.

iii. Use Naive Bayes Classifier to Analysis

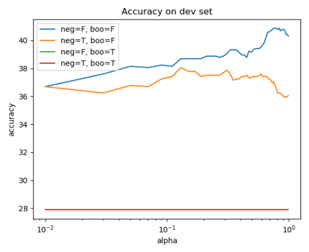
- For probability, I decided to use the log(probability) to avoid overflow and used the negative number to convert the target to a minimize problem. The total probability contains 2 parts: the prior probability and the likelihood probability, which can all be easily calculated using the Bag of Wrods Model.
- In addition, when calculate the probability of the words,we may calculate a word's probability many times which causes a waste of time, we can bulid a dictionary to store the probability each time it is computed, then the next time we call it directly instead of calculate it again.
- To deal with the words that doesn't appear in vocabulary, I use the add-k smoothing, and let alpha = k, later we will train it on development set to get a best alpha.
- When given a sentence and to classify it, I compute the -log(probabilities) under each label and then return the label which minimize the negative log(probability).

iv. Result

- Set the negation and boolean equal to *[(False, False), (True, False), (False, True), (True, True)]* respectively.
- For each negation and boolean values, set alpha(add-k parameters) equal to a range of values between 0 and 1, to train a best alpha.
- Finally run each model with best alpha under the test set.

Here shows the results:

1. Accuracy on the development set



2. Accuracy on the test set

Negation	Boolean	Best Alpha	Accuracy
False	False	0.83	40.2262%
True	False	0.13	40.1810%
False	True	0.99	29.7738%
True	True	0.99	29.7738%

v. Comment and Future Improvement

- We can see from the results that the general Naive Bayes outperforms the boolean Naive Bayes mtehod, and can reach an accuracy of about 40%.
- Whether to deal negation has no apparent differences in the accuracy of the classifiers.
- I think the reason that leads to the above two outcomes is that, the training set given is not so large. Then if we remove the duplicate words, the corpus will be smaller. And since the training data is small, doing a negation reduces some frequencis of words which may make classifier less accurate.

- For the improvement, I think (a). Add some rules (based on the common sense) to the classifier (b). Use a online lexicon resources to help classifier judge whether a word is positive or negative. will be helpful. This work may be done in the future.

III. Part II: Word2vec Model Based

i. Word2vec Training

i.1 normalizeRows()

It is easy to fill by divid each element with the norm of each row.

$$x = \frac{x}{norm(x)}$$

i.2 softmaxCostAndGradient()

$$J_{softmax_CE}(o, v_c, U) = -\sum_i y_i \log \hat{y}_i$$

Due to $y_{target} = 1$, $y_{i \neq target} = 0$, $\hat{y} = \frac{exp(u_o^T v_c)}{\sum_w^V exp(u_w^T v_c)}$
So

$$cost = -log\hat{y}_{target},\ gradPred = U(\hat{y} - y),\ grad = v_c(\hat{y} - y)^T$$

i.3 negSamplingCostAndGradient()

Similar to above:

$$J_{neg-sample}(o, v_c, U) = -log(\sigma(u_o^T v_c)) - \sum_k^K log(\sigma(-u_k^T u_c))$$

then $cost = J_{neg-sample}$,

$$gradPred = (\sigma(u_o^T v_c) - 1)u_o - \sum_k^K (\sigma(-u_k^T v_c) - 1)u_k$$

For grad, $grad_{u_o} = (\sigma(u_o^T v_c) - 1)v_c$,

$$gard_{u_k} = -(\sigma(-u_k^T v_c) - 1)v_c$$

i.4 skipgram()

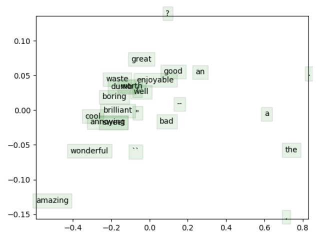
To fill in this block, do the following steps:

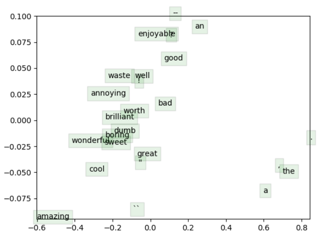
- Get index based on *tokens[currentWord]*, then I can get input vector.
- For each word in context words, use the given function to calculate the cost and gardient, and combine them to return.

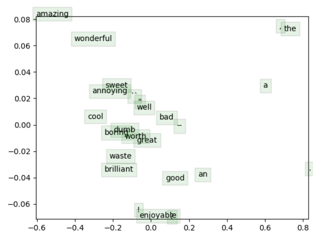
i.5 The plot of word vectors that we trained

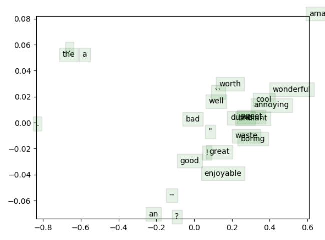
Use Negative Sampling method and softmaxRegression to train my word vectors, and set the iterations and stepsize equal to 40000 and 0.3 respectively.

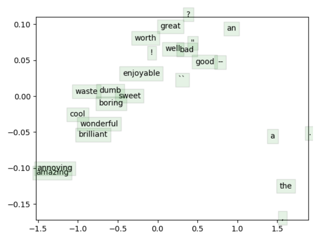
To obtain a more accurate answer, I trained several word vectors with different hyperparameter.(dim and number of negative samples)

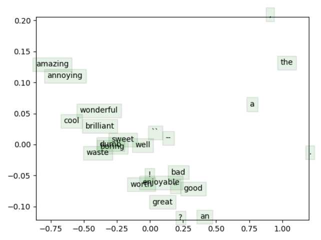

(a) *dim=10,K=10*

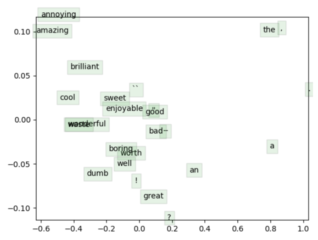

(b) *dim=15,K=10*

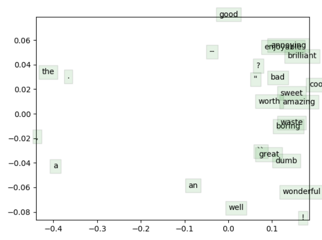

(c) *dim=20,K=10*


(d) *dim=25,K=10*


(e) *dim=10,K=4*


(f) *dim=10,K=6*


(g) *dim=10,K=8*


(h) *dim=10, softmaxRegression*

ii. Sentiment Analysis

ii.1 Get a sentence feature

Use the average of all the word vectors in that sentence as its feature.

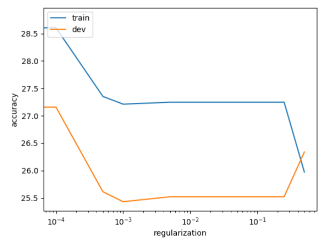
$$feature = \frac{1}{n}\sum_{i=1}^{n} wordVectors[w_i]$$

ii.2 Get best Regularization

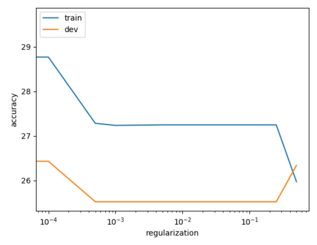
- Set $REGULARIZATION$ equal to one of the values $[0, 0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.25, 0.5]$ respectively.
- For each REGULARIZATION, run the sentiment analysis on the training set and the development set, then

choose the best REGULARIZATION which has the highest accuracy on development set.
- I do the above things on each word vectors I trained in the last part.

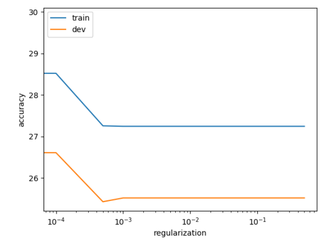
ii.3 Plot the accuracy on trainset and devset.



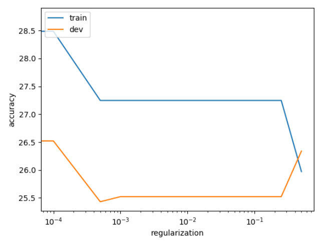
(a) *dim=10,K=10*



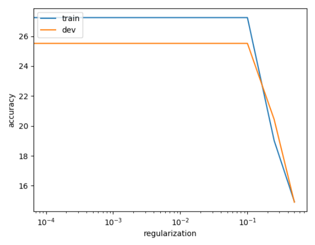
(b) *dim=15,K=10*



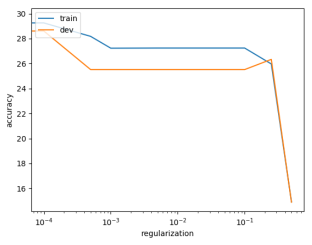
(c) *dim=20,K=10*



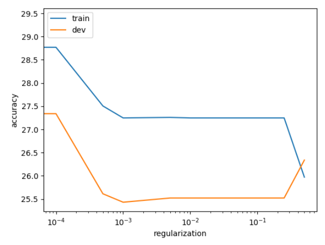
(d) *dim=25,K=10*



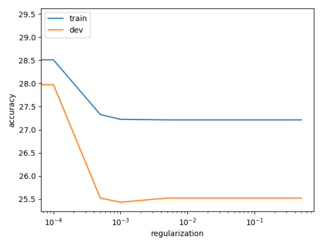
(e) *dim=10,K=4*



(f) *dim=10,K=6*



(g) *dim=10,K=8*



(h) *dim=10, softmaxRegression*

ii.4 The accuracy on the test set

After I choosed the best REGULARIZATION, I run the sentiment analysis program with it on the test set and get the following outcomes.

RegressionMethod	Dim	SampleNum(K)	Accuracy
negSamping	10	10	26.832579%
negSamping	15	10	26.968326%
negSamping	20	10	27.782805%
negSamping	25	10	26.968326%
negSamping	10	4	23.031674%
negSamping	10	6	27.511312%
negSamping	10	8	27.194570%
softmaxRegression	10	/	26.696833%

iii. Comment

- We can see from the plot of the word vectors, increasing the dim of the word vector properly will make the word vectors more loosen and clear. And decreasing the number of negative sample can do the same thing. But not all the position of the words with similar meaning in the plot are closer, and some words with contrary meaning are actually closer.
- With the dim of the vectors increasing, the accuracy on the test set first increase but fluctuate slightly after that.
- The number of negative samples dosen't need to be very large or very small, they will both lead a decrease in accuracy.
- The softmaxRegression method has a similar accuracy with the negative Sampling, but takes a relatively longer time than negativeSampling.
- Although the accuracy may change, but changes are not so apparent. I think it is due to the size of the training data and the specific problem.

IV. Conclusion

1. In this problem, the Model based on the feature engineering obviously outperforms the Model based on the word2vec. As a fact, the word2vec model need a large-scale training data to get good word vectors, so when the training set is relatively small, the feature engineering model behaves better.
2. No matter what models we use, the larger the training set is, the better performance We may get.
3. In word2vec model, we should choose the dim of the vectors properly, because when it is increased to some extent, the accuracy won't increase much while the run time will actually increase much!!
4. When using negativeSampling to train word2vec model, we should also choose a proper K(negative Samples), because on one hand if K is small, the model will converges very slow, on the other hand if K is large it will less effect and even make the model less accurate.