CS918 NLP

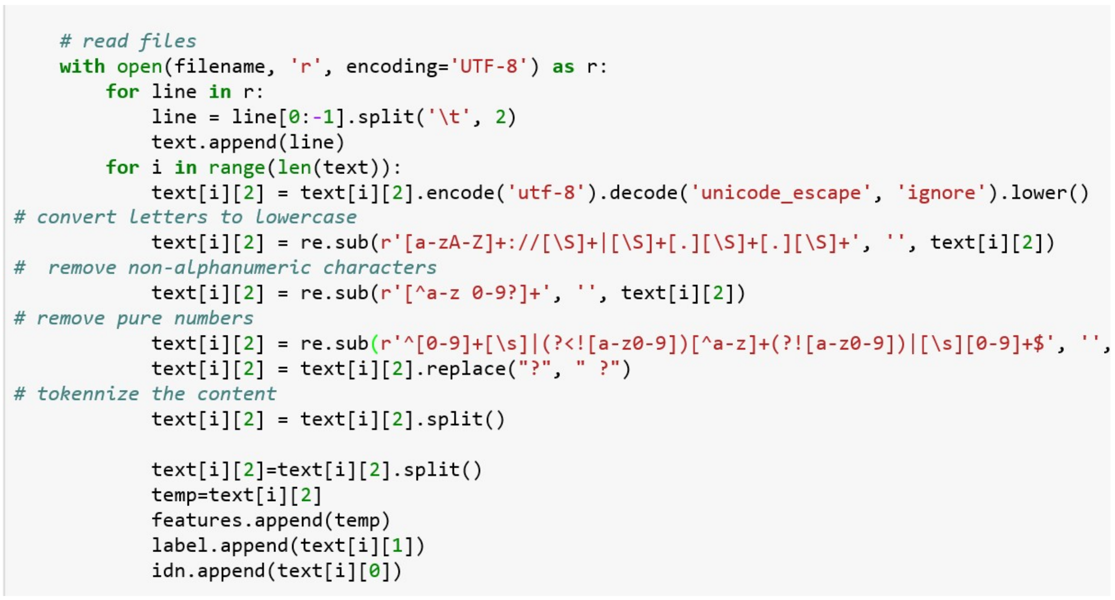
Introduction

This report will present a solution of sentiment classification for social media. There are three methods in this assignment which focus on three different feature extraction techniques. The processed data are applied to Logistic Regression Classifier respectively.

The next section presents the preprocessing methods, section III is feature selection, section IV will introduce the classifier used in this paper, section V is the evaluation of the performance and conclusion.

Preprocessing

As the original file is obtained from Tweeter, the form is not appropriate to be used in classifiers directly. The first step of this paper is preprocessing.



```
    # read files
    with open(filename, 'r', encoding='UTF-8') as r:
        for line in r:
            line = line[0:-1].split('\t', 2)
            text.append(line)
        for i in range(len(text)):
            text[i][2] = text[i][2].encode('utf-8').decode('unicode_escape', 'ignore').lower()
# convert letters to lowercase
            text[i][2] = re.sub(r'[a-zA-Z]+://[\S]+|[\S]+[.][\S]+[.][\S]+', '', text[i][2])
#  remove non-alphanumeric characters
            text[i][2] = re.sub(r'[^a-z 0-9?]+', '', text[i][2])
# remove pure numbers
            text[i][2] = re.sub(r'^[0-9]+[\s]|(?<![a-z0-9])[^a-z]+(?![a-z0-9])|[\s][0-9]+$', '',
            text[i][2] = text[i][2].replace("?", " ?")
# tokennize the content
            text[i][2] = text[i][2].split()

            text[i][2]=text[i][2].split()
            temp=text[i][2]
            features.append(temp)
            label.append(text[i][1])
            idn.append(text[i][0])
```

Each line of this data is divided into three parts which means that the ID, label, and test content can be stored separately, and then the content are tokenized. The rest preprocessing will focus on the main text, the URL addresses, meaningless indications are deleted, and pure numbers are deleted. As stop words are irrelated with the sentiment trend, through an imported dictionary these words are found and removed.

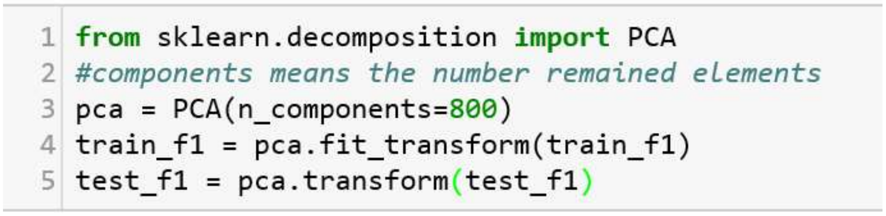
Features extraction

This paper exploit mainly two approaches to extract useful features from these Tweets data: N-gram method, word embedding.

N-gram

N-gram models are widely used as a feature selection method, which is normally used to predict the next item. In this paper we just utilize it to build the sentence vectors. With the help of NLTK tool package the bigram bag can be created which contains the all possible combinations of two neighboring words. Considering the efficiency only the most top 1000 bigrams are collected.

After obtained a bigram list, the next step is to convert each sentence to a vector with the same dimension. Simply, these vectors are created with a dimension equals to the size of bigram bag (1000). If the element in bigrams list is existed in the sentence the value in this position will be written as “1”, otherwise the value will be “0”. However, when using such a high dimension vectors to describe a sentence, these vectors can be very sparse, which means there must be many uncorrelated data included. To tackle this Principal component analysis (PCA) method are applied. PCA can convert a set of possibly correlated variables into a linearly irrelated variables.



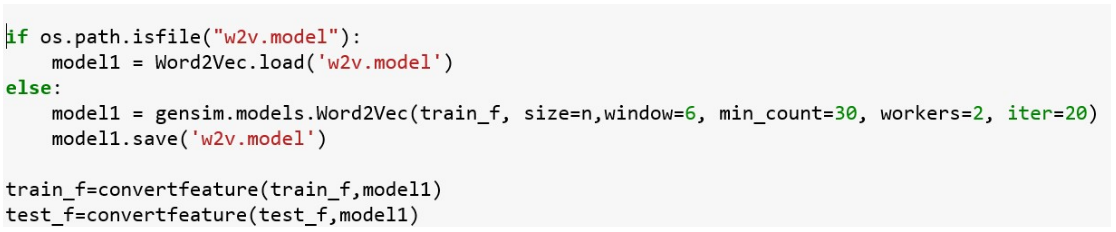
```
from sklearn.decomposition import PCA
#components means the number remained elements
pca = PCA(n_components=800)
train_f1 = pca.fit_transform(train_f1)
test_f1 = pca.transform(test_f1)
```

To keep consistency, the features in test set should also be filtered in this way too. Therefore, he dimensionality reduction pattern are stored as “pca”, which are used to test set too. In this method the number of bigrams are set as 800.

Word embedding

Word2vec

Word embedding is a technique to convert words or phrases from the vocabulary to a group vectors of real numbers, which involves a mathematical concept of embedding a word from one dimension to continuous vector space. In this paper Word2vec technique are first implied. Word2vec was developed by Tomas Mikolov and his coworkers when he worded at Google. This algorithm is a shallow, 2-layer neural network which are trained to evaluate the similarity of different words based on their neighboring words, the size of words group is called window. In this paper the value of window is set as 5. And the words appear less than 30 times are removed.



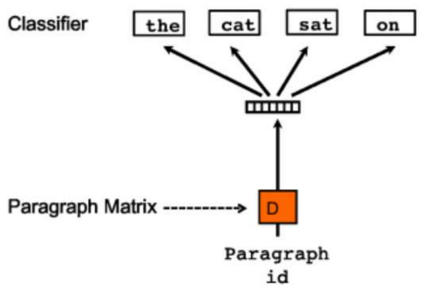
```
if os.path.isfile("w2v.model"):
    model1 = Word2Vec.load('w2v.model')
else:
    model1 = gensim.models.Word2Vec(train_f, size=n,window=6, min_count=30, workers=2, iter=20)
    model1.save('w2v.model')

train_f=convertfeature(train_f,model1)
test_f=convertfeature(test_f,model1)
```

$$vs_i = \frac{\sum vw_i}{\text{number of words}}$$

Even though this algorithm has converted each word to a vector, they cannot be utilized to classifier directly. Sentence vectors should be created based on word vectors. With the limitation of time a simple function is implied. For every single sentence, the vector are the mean vectors of its whole word vectors which means the value of a particular dimension of sentence vector are the mean value of all word vectors of this dimension.

Doc2vec

This algorithm is developed from word2vec, by Mikolov, in 2013. The main idea of doc2vec is to add a paragraph vector when training the word vector. As a result, the relationship of the context of neighboring sentences are considered. The processing methods of doc2vec are similar with word2vec.



```
import gensim.models.doc2vec
from gensim.models.doc2vec import Doc2Vec,LabeledSentence
from gensim.models.doc2vec import TaggedDocument
# using TaggedDocument to add tags
def doc2v_file(word):
    sen=[]
    for i in range(len(word)):
        T = TaggedDocument(word[i],[i])
        sen.append(T)
    return sen
sen=doc2v_file(train_f)
model = Doc2Vec(sen,size =50, window = 5, min_count=1,workers=2)
model2 = Doc2Vec.load('doc2vec.model')
```

Classifiers

As this paper applied word2vec, doc2vec and Ngram + PCA techniques the values in vectors varies a lot and contain negative values. The naïve Bayes and Decision tree classifiers can not be used directly. The Deep learning methods and SVM are very time consuming. After Compared the performance of KNN and Logistic Regression(LR) algorithm, this paper choses LR as the classifier.

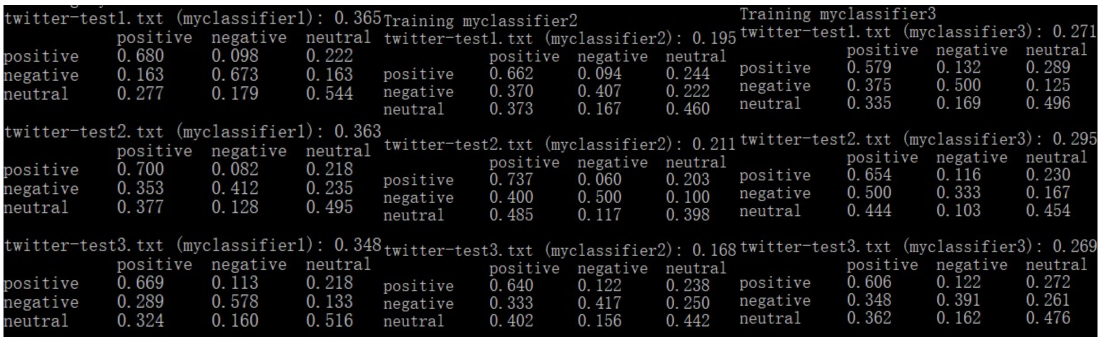
```
def logistic_regression_classifier(train_x, train_y):
    from sklearn.linear_model import LogisticRegression
    model = LogisticRegression(penalty='l2')
    model.fit(train_x, train_y)
    return model
```

LR is a regression model where the dependent variable (DV) is categorical. Logistic regression was developed by statistician David Cox in 1958.[2][3] The binary logistic model is used to estimate the probability of a binary response based on one or more predictor (or independent) variables (features).

To save time the trained models have been stored in “model” file.

Evaluation

The figure below shows the macroaveraged F1 score of these three classifers for the neutral, positive and negative classes over 3 test sets. The results show that the word2vec methods gives the best performance under the given test set.



```
twitter-test1.txt (myclassifier1): 0.365
          positive  negative  neutral
positive  0.680     0.098     0.222
negative  0.163     0.673     0.163
neutral   0.277     0.179     0.544

twitter-test2.txt (myclassifier1): 0.363
          positive  negative  neutral
positive  0.700     0.082     0.218
negative  0.353     0.412     0.235
neutral   0.377     0.128     0.495

twitter-test3.txt (myclassifier1): 0.348
          positive  negative  neutral
positive  0.669     0.113     0.218
negative  0.289     0.578     0.133
neutral   0.324     0.160     0.516
```

```
Training myclassifier2
twitter-test1.txt (myclassifier2): 0.195
          positive  negative  neutral
positive  0.662     0.094     0.244
negative  0.370     0.407     0.222
neutral   0.373     0.167     0.460

twitter-test2.txt (myclassifier2): 0.211
          positive  negative  neutral
positive  0.737     0.060     0.203
negative  0.400     0.500     0.100
neutral   0.485     0.117     0.398

twitter-test3.txt (myclassifier2): 0.168
          positive  negative  neutral
positive  0.640     0.122     0.238
negative  0.333     0.417     0.250
neutral   0.402     0.156     0.442
```

```
Training myclassifier3
twitter-test1.txt (myclassifier3): 0.271
          positive  negative  neutral
positive  0.579     0.132     0.289
negative  0.375     0.500     0.125
neutral   0.335     0.169     0.496

twitter-test2.txt (myclassifier3): 0.295
          positive  negative  neutral
positive  0.654     0.116     0.230
negative  0.500     0.333     0.167
neutral   0.444     0.103     0.454

twitter-test3.txt (myclassifier3): 0.269
          positive  negative  neutral
positive  0.606     0.122     0.272
negative  0.348     0.391     0.261
neutral   0.362     0.162     0.476
```

classifier1:word2vec classifier2:ngram classifier3:doc2vec

Conclusion

All of these three classifiers perform good in predicting the positive tweets while unsatisfied for negative class, which may due to the unbalanced train set. The number of positive tweets is twice of negative tweets. Normally for long text data the doc2vec techniques are mostly have a better performance than word2vec, while for shorts text like Tweets, adding paragraph vectors may have a negative effect. With the limitation of time when processing the original dataset, the influence of negation words are neglected, which may lead a bad performance of N-gram classifier, While since the word2vec algorithms have considered the neighboring words the impact may be released slightly.

With the limitation of time I cannot spend many time in data preprocessing and the parameters modification of classifiers which may also lead a negative. The use of SVM and deep learning methods may help to improve the performance.