Machine Learning Engineer Nanodegree

Capstone Project

Peng Wu

April 2nd, 2018

I. Definition

Project Overview

As personal computers and mobile computing platforms have become virtually ubiquitous over the past decade, people have largely adopted the use of online documents as a way of expressing their feelings and thoughts on the world around them. As this online method of expression has continued to gain traction, it has become a valuable task to attempt to categorize these submissions for the purpose of gaining information about public sentiment. There are many reasons why an individual or organization would have an interest in public sentiment, but one important motivation is commercial: by being able to gain an understanding of how members of the public are reacting to a product or announcement, companies and retailers can adapt their behaviors and business strategies to increase positive sentiment surrounding their products, while minimizing sources of negative sentiment.

Problem Statement

The primary challenge inherent to processing online submissions is at the very thing that makes these submissions so attractive as a representation of public sentiment: the amount of data involved is often extremely large. In order to process text information in these quantities, it has become necessary to turn to the use of automated processing systems. Although the task of automatically extracting information from text, broadly referred to as natural language processing, has resulted in the establishment of procedures for classifying text based on objective markers such as subject matter,

procedures for classifying text based on subjective markers of sentiment are yet unestablished. While humans are well conditioned to interpret emotion from speech and writing, there are no obvious and well established markers of sentiment that can easily be identified by a computer. Several heuristic methods have been suggested for identifying features or markers of a particular sentiment, and various papers have been published detailing attempts at performing automated sentiment analysis on large bodies of online text.

In this project, I will explore some of those feature extraction methods as well as machine learning methods that might be applied to them to successfully perform automated textual sentiment analysis.

Our final goal is to create a model that can do Twitter sentimental analysis and the tasks involved are the following:

- Process the short movie data
- Train the traditional classifier that can determine the sentiment of a sentence
- Train a model using a deep learning method that can determine the sentiment of a sentence
- Make a Twitter application that extract the twitters and analyse its sentiment by a specific word.

Metrics

Accuracy is the common metric for binary classifiers[1].

We can clearly see that the accuracy means from the figure: the proportion of the total number of predictions that were correct. In general, we consider one of the defined metrics in the figure. For example, in a pharmaceutical company, they will be more concerned with minimal wrong positive diagnosis. So, they will be more use the metrics of Specificity. In our binary classifier problem, we need to predict if the sentences if classified in the correct sentimental value. Hence, we only concerned the metrics of Accacy.

Confusion Matrix		Target			
		Positive	Negative		
Model	Positive	a	b	*Positive Predictive Value*	a/(a+b)
	Negative	c	d	*Negative Predictive Value*	d/(c+d)
		Sensitivity	*Specificity*	**Accuracy** = (a+d)/(a+b+c+d)	
		a/(a+c)	d/(b+d)		

Figure 1

II. Analysis

(approx. 2-4 pages)

Data Exploration

The first dataset we use is from nltk corpus “movie reviews”. This corpus is from paper of Bo Pang and Lillian Lee[2]. It contains 2k movie reviews with sentiment polarity classification. The reason that I used it in this project because it is processed and easy to used as in nltk corpus. It has 1K positive reviews and 1K negative reviews. Most the reviews are long, have more than 1000 words.

We also explore what words they most contain. Shown below figure(2), we can see that the most appeared words are the comma symbol and other symbols, also have some words like the “a”, “and” and so on. The words and symbols usually have little effect on the meaning of a sentence. So, after we remove this symbols and words, we can see the most frequent words are shown in fiugre(3)

```
[(',', 77717), ('the', 76529), ('.', 65876), ('a', 38106), ('and', 35576), ('o
f', 34123), ('to', 31937), ("'", 30585), ('is', 25195), ('in', 21822), ('s', 18
513), ('"', 17612), ('it', 16107), ('that', 15924), ('-', 15595)]
```

Figure 2

```
[('film', 9517), ('one', 5852), ('movie', 5771), ('like', 3690), ('even', 256
5), ('good', 2411), ('time', 2411), ('story', 2169), ('would', 2109), ('much',
2049), ('character', 2020), ('also', 1967), ('get', 1949), ('two', 1911), ('wel
l', 1906)]
```

Figure 3

Another corpus used is from this website[3]. Just as it described in the website, our main goal is to do Twitter sentiment. Usually the twitter is very short, if we use the corpus in nltk, the result is not what we will expect. Hence, short movie reviews maybe better to fit for the twitters from Twitter.

This corpus contains 5332 positive and 5332 negative movies reviews. It roughly has 30 words per sentence. The most frequent words are shown below after remove the punctuation and the stop words.

```
has  5332  postive movies reviews
has  5332  negative movies reviews
all documents is  10664
[("'s", 3537), ('film', 1589), ('movie', 1336), ("n't", 940), ('one', 739), ('like', 720), ('--', 67
0), ('``', 655), ('story', 493), ('much', 386), ('even', 382), ('good', 377), ('comedy', 356), ('tim
e', 341), ('characters', 330)]
```

Figure 4

Exploratory Visualization

The figures below show the visualization of the words' frequence in corpus and the distribution of the sentences' length.

In figure 1,2, we list the first 30 words. We can see that they have many words are both in the two corpus, such as film, good, make, like, bad and so on.

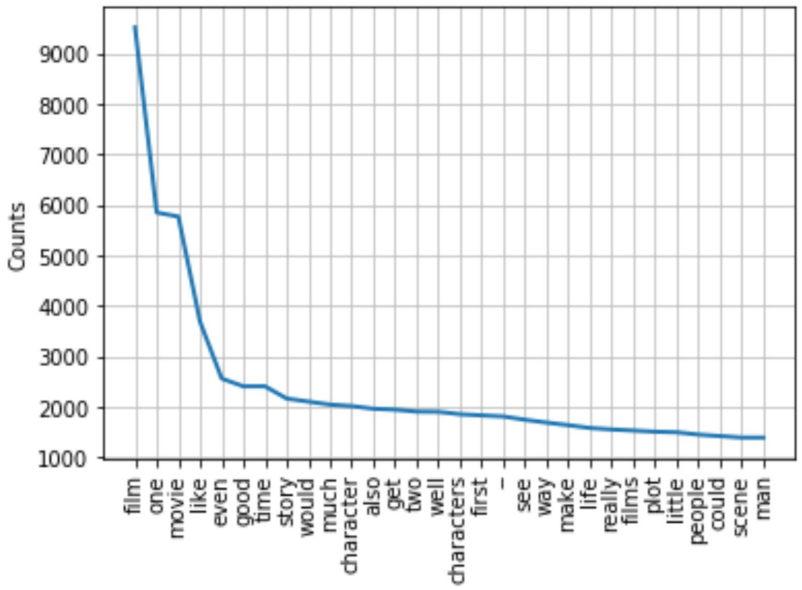

Figure 5

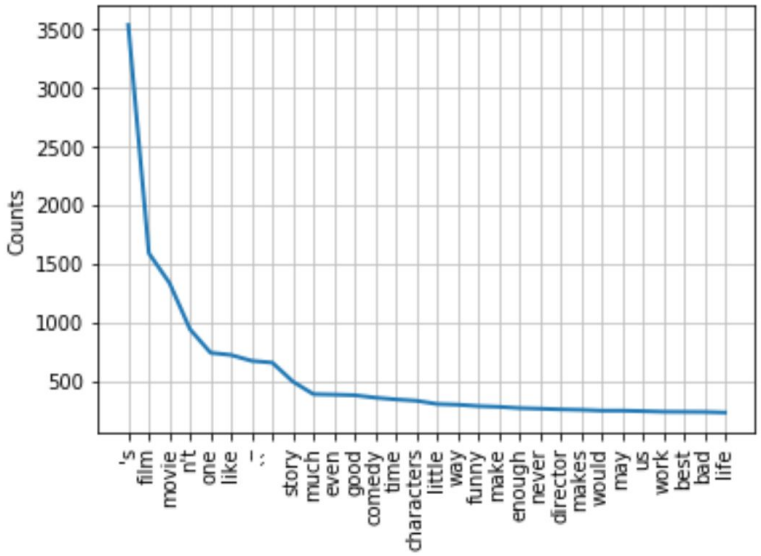

Figure 6

In figure 3 and 4. The first corpus has the longer sentence length. Most sentences length are among 500~1000.

The second corpus has much shorter sentence length, the longest sentence only has 59 words. Most of the them are among 10~40 words which is more likely we write our twitters.

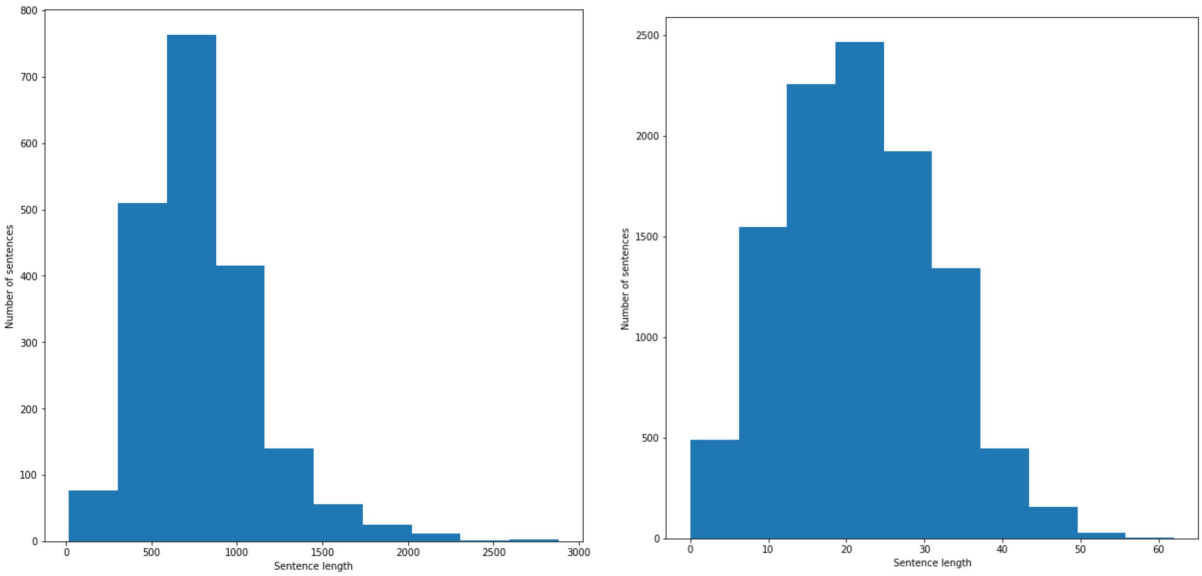

Figure 7 Figure 8

Algorithms and Techniques

Naïve Bayes Classifier

The Naïve Bayes classifier is based off of Bayes rule, which defines the probability of a sample belonging to class ωi, given that the sample has feature vector x. Naïve Bayes simplifies this condition by assuming that each feature in x is statistically independent, although this is clearly not true for real world examples of sentiment text. If we assume ni(x) is a function which represents the number of times that feature fi occurs in x, then we can show Naïve Bayes rule as follows:

$$P_{NB}(\omega_i|\mathbf{x}) = \frac{P(\omega_i) * (\prod_{i=1}^{m} P(f_i\ |\omega_i)^{n_i(d)})}{P(\mathbf{x})}$$

A Naïve Bayes classifier then selects a class for each sample where the probability of that class is maximized. In this way, a Naïve Bayes classifier can select a class for each sample based on its feature vectors. In our specific example, the Naïve Bayes classifier was trained using relative-frequency estimation of the prior and add-one smoothing.

Despite its simplicity, and reliance on the false assumption that the individual features are independent, we see that the Naïve Bayes classifier, in some cases, works quite well as a sentiment analyzer.

Convolutional Neural Networks[4]

In traditional Neural Networks, we connect each input neuron to each output neuron in the next layer and learn the weights and biases of each neuron. However, in Convolutional Neural Networks[5], we use convolutions over the input layer to compute the output. Each layer applies different filters,during the training phase, a CNN automatically learns the values of its filters. It is common to periodically insert a Pooling layer in-between successive Conv layers in a ConvNet architecture. Its function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting.

CNN is widely used in Computer Vision, but how it apply to natural language processing as NLP tasks are sentences and words very different from the pixels. The method we usually used is transform the words into matrix, each row is vector that represents a word. This method is called Word Embedding[6]. There are several ways to do word embedding, such as word2vec[7], GloVe[8], FastText[9]. In this project, we will use the word embedding functional API in Keras.

Benchmark

From this website[10] , I find that it's accuracy is around 67%. Using the same corpus from this website, so I hope get a better accuracy. I have create two models for my Twitter application. One uses traditional classifiers, such as NaiveBayesClassifier, BernoulliNB_classifier, LogisticRegression_classifier and so on. I hope my accuracy will be around 70%. Another way is used by CNN, I wish the accuracy will be around 75%.

III. Methodology

Data Preprocessing

Data preparation can be constitued by those steps：

1. loading and cleaning reviews
    a. Token the text
    b. Remove all punctuation from words
    c. Remove all words that are known stop words
2. Define the feature set
    a. In traditional classifier, we use the most 5k frequent words as the features. Then to check if the words in a review or not. If In it, get the value "True", otherwise, the "False", just as shown below figure 9.
    b. another is to turn the words into vectors, this method is used in deep learning(figure 10). It contains two steps, first make texts to sequences, the second steps is to pad the sequences in the same length which can be used in training.
3. Split the dataset into the training and testing dataset

Converting words to Features with NLTK

```
# get the most 3000 common words
word_features = list(filtered_words_f.keys())[:3000]

def find_features(document):
    words = set(document)
    features = {}
    for w in word_features:
        features[w] = (w in words)

    return features

print((find_features(movie_reviews.words('pos/cv004_11636.txt'))))
```

```
{'plot': True, 'two': True, 'teen': False, 'couples': False, 'go': True, 'c
e': False, 'get': False, 'accident': False, 'one': True, 'guys': True, 'die
'see': True, 'life': False, 'nightmares': False, 'deal': False, 'watch': Fa
'critique': False, 'mind': False, 'fuck': False, 'generation': False, 'touc
s': False, 'bad': False, 'package': False, 'makes': False, 'review': False,
ce': False, 'generally': False, 'applaud': False, 'films': True, 'attempt':
```

Figure 9

```
: # training documents as sequences of integers using the Tokenizer class in the Keras API
from keras.preprocessing.text import Tokenizer

# create the tokenizer
tokenizer = Tokenizer()
# fit the tokenizer on the documents
tokenizer.fit_on_texts(filter_sentence)
# sequence encode
encoded_docs = tokenizer.texts_to_sequences(filter_sentence)
print(encoded_docs[1])
```

```
Using TensorFlow backend.
```

```
[3228, 2053, 7158, 4517, 2823, 4518, 957, 7159, 706, 769, 4519, 2513, 573, 162, 16, 958, 5502, 4
0, 372, 1370]
```

```
: # pad sequences
from keras.preprocessing.sequence import pad_sequences
max_length = max([len(s.split()) for s in filter_sentence])
Xtrain = pad_sequences(encoded_docs, maxlen=max_length, padding='post')
print(Xtrain[1:3])
```

```
[[3228 2053 7158 4517 2823 4518  957 7159  706  769 4519 2513  573  162
    16  958 5502 4520  574 1178 1887 1887 7160  372 1370    0    0    0
     0    0    0    0    0    0    0    0    0    0    0]
 [ 607 1371 2514 2515    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0]]
```

Figure 10

Implementation

It mainly contain two parts:

1. Algorithm implementation
2. Twitter application

I used two ways to do algorithm implement, one uses traditional classifier, another uses CNN.

A. In the traditional classifier:

After the data processing, we get the training and testing dataset. Using the classifier to train and get the accuracy shown in figure[11].

```
#apply naiveBayes classifier
classifier = nltk.NaiveBayesClassifier.train(training_set)
print("Original Naive Bayes Algo accuracy percent:", (nltk.classify.accuracy(classifier, testing_set))*100)
#list the word from feature that can most valuable to reflect its catergory
classifier.show_most_informative_features(15)

#save classifier-naiveBayes
save_classifier = open("pickled_algos/originalnaivebayes5k.pickle","wb")
pickle.dump(classifier, save_classifier)
save_classifier.close()
```

```
Original Naive Bayes Algo accuracy percent: 72.59036144578313
Most Informative Features
              engrossing = True              pos : neg    =     21.0 : 1.0
               wonderful = True              pos : neg    =     20.3 : 1.0
                captures = True              pos : neg    =     19.0 : 1.0
               inventive = True              pos : neg    =     15.6 : 1.0
                intimate = True              pos : neg    =     15.0 : 1.0
               absorbing = True              pos : neg    =     13.7 : 1.0
              refreshing = True              pos : neg    =     13.0 : 1.0
            refreshingly = True              pos : neg    =     13.0 : 1.0
                    warm = True              pos : neg    =     12.6 : 1.0
              realistic = True               pos : neg    =     11.7 : 1.0
                 stupid = True               neg : pos    =     11.0 : 1.0
                   ages = True               pos : neg    =     11.0 : 1.0
               provides = True               pos : neg    =     11.0 : 1.0
               chilling = True               pos : neg    =     10.3 : 1.0
             mesmerizing = True              pos : neg    =     10.3 : 1.0
```

Figure 11

B. In the CNN classifier:

we used the Keras to help us build the model, it contains several layers

a. Embedding layer

In the data processing part, we know how text transform to sequence, it also belong to the process of word embedding, so is the embedding layer.

1. Remove the stop words and punctuation.
2. training documents as sequences of integers using the Tokenizer class in the Keras API
3. Pad all reviews to the length of the longest review
4. Take the sequence to the embedding layer.

The Embedding layer is initialized with random weights and will learn an embedding for all of the words in the training dataset. The Embedding layers has 3 arguments, “input_dim” is the vocabulary size, refer to the total number of words in our vocabulary, plus one for unknown words. “Output_dim”, This is the size of the vector space in which words will be embedded. “input_length”: This is the length of input sequences.

b. Conv 1d layer
c. maxpooling1D layer

A CNN configuration in this project has 64 filters, and a kernel size of 3 with a rectified linear (‘relu’) activation function. This is followed by a pooling layer that reduces the output of the convolutional layer by half.

d. Flatten layer

Next, the 2D output from the CNN part of the model is flattened to one long 2D vector to represent the ‘features’ extracted by the CNN.

e. Dropout layer

Dropout can be applied between layers using the Dropout Keras layer. It can reduce the complexity of model and reduce the overfitting problems.

f. Dense layer

A dense layer is a classic fully connected neural network layer : each input node is connected to each output node. The output layer uses a sigmoid activation function to output a value between 0 and 1 for the negative and positive sentiment in the review.

g. Fit the network on the training data.

We use a binary cross entropy loss function because the problem we are learning is a binary classification problem. The efficient Adam implementation of stochastic gradient

descent is used and we keep track of accuracy in addition to loss during training. The model is trained for 3 epochs through the training data.

h. Evaluate the model
i. Save the model for the future usage

```
model1 = Sequential()
model1.add(Embedding(vocab_size, 50, input_length=max_length))
model1.add(Dropout(0.5))
model1.add(Conv1D(filters=64, kernel_size=3, activation='relu'))
model1.add(MaxPooling1D(pool_size=2))
model1.add(Dropout(0.5))
model1.add(Flatten())
model1.add(Dense(100, activation='relu'))
model1.add(Dense(1, activation='sigmoid'))
print(model1.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_28 (Embedding)     (None, 39, 50)            971600
_________________________________________________________________
dropout_24 (Dropout)         (None, 39, 50)            0
_________________________________________________________________
conv1d_15 (Conv1D)           (None, 37, 64)            9664
_________________________________________________________________
max_pooling1d_15 (MaxPooling (None, 18, 64)            0
_________________________________________________________________
dropout_25 (Dropout)         (None, 18, 64)            0
_________________________________________________________________
flatten_13 (Flatten)         (None, 1152)              0
_________________________________________________________________
dense_40 (Dense)             (None, 100)               115300
_________________________________________________________________
dense_41 (Dense)             (None, 1)                 101
```

:

```
# compile network
model1.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# fit network
model1.fit(x_train, y_train, epochs=3, verbose=2, batch_size=64, shuffle=True)
```

Figure 12

The second part consist of：

1. Reload the model;
2. Get the twitters streaming data from Twitter by its API；

We can search for a word and streaming the twitters mentioned this word. For example, we can get the streaming data about “Trump”. In this application, we acquire the 400 twitters in real time.

3. The data processing ;

The data we acquire need to process, just as the data processing part. It contains several step:

   a. Remove the punctuation and the stop words;
   b. Load the filtered vocabulary of dataset, and remove the words that not in this vocabulary bag.
   c. Pad the filtered sentences to sequence
4. Take the sequence to the model we reload before, and predict its class.
5. Plot the results to show the sentimental tendency ;

Refinement

We use this 3 method to improve our result.

1. Boosting

We have training the data by several traditional algorithms, such as NaiveBayesClassifier, BernoulliNB_classifier, LogisticRegression_classifier and so on. Do we just need to choose one of the them as our final algorithm? Actually, we don’t need do that. Combine several classifier algorithms is the common technique. Turning the several weaker classifiers to a better one is called boosting.

In this project, we create a voting system. For example. Among the 5 classifier, 3 of them classify the result as 1, and 2 of them classify the result as 0. The voting system then will consider the result as 1.

```
voted_classifier = VoteClassifier(classifier,
                                  NuSVC_classifier,
                                  LinearSVC_classifier,
                                  SGDClassifier_classifier,
                                  MNB_classifier,
                                  BernoulliNB_classifier,
                                  LogisticRegression_classifier)

print("voted_classifier accuracy percent:", (nltk.classify.accuracy(voted_classifier, testing_set))*100)
```

Figure 13

2. Dropout layer

At first, the dropout layer is not applied in the CNN model, the result is poor. It is very easy to overfit. By using the dropout layer, we reduce the complexity of the model, and avoid the overfitting problem. I tried several times, set the dropout to 0.5 and get the best accuracy.

IV. Results

Model Evaluation and Validation

We random shuffle our datasets first, then split it into training data and testing data. We use our training data to train our model, then use the testing data to evaluate this model.

For the first corpus we get the results:

classifier	accuracy percent
MNB_classifier	79.0
NaiveBayes	77.0
BernoulliNB	75.0
LogisticRegression	79.0
SGDClassifier	77.0
LinearSVC	76.0
NuSVC	81.0
Boosting	83.0

Table 1

We can see from the table, the boosting method improve the the model.

For the second corpus, the short movie reviews get the result shown below:

classifier	accuracy percent
MNB_classifier	71.6867
NaiveBayes	71.2349
BernoulliNB	70.7831
LogisticRegression	70.1807
NuSVC	69.1265
Boosting	72.2891
CNN	75.638138
LSTM	74.06

Table 2

Compare to the first corpus, the accuracy of second corpus is reduced. The reason is that the first corpus has longer sentence than the second, have more information to do the classifier.

However, our goal is to do the twitter sentimental analysis, thus we need the short corpus that similar to twitter's length. Using the boosting method, we get better accuracy compare to the traditional classifier, up to 72.3%.

To get a better result, I train a CNN model, the accuracy is improved by 3% than the boosting method.

I also train a LSTM model, which not improve the accuracy than the CNN, so I drop this method for our twitter application.

Justification

Compare to the benchmark, we wish using the traditional classifier to get 70% accuracy, and finally result is 72%, which is a good start. Then I go further, train a CNN for this sentimental analysis, and get a accuracy 75.6%, which is what I expect in the benchmark.

V. Conclusion

Free-Form Visualization

The final result that visualize the twitter for searching the word: Trump.

The twitter result is classified as negative and positive. If the result is positive, we puls 1, if the result is negative, we minus 1. The Y-axis show the cumulative result. The x-axis is the number of Twitter we used. The tendency of the figure can reflect people's attitude to "Trump".

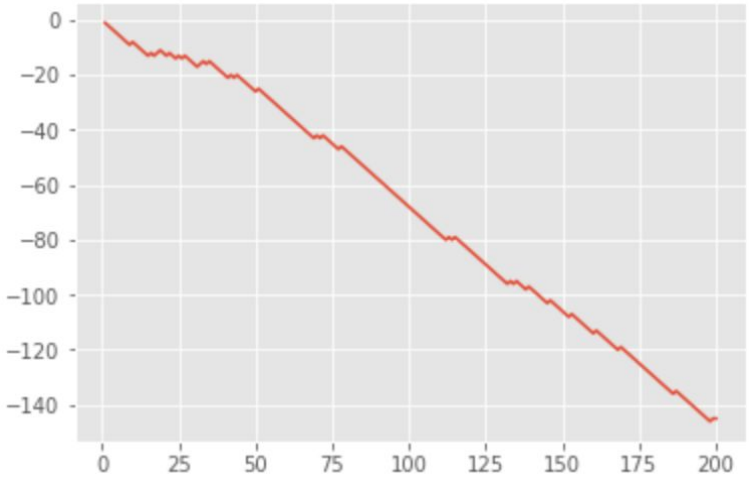

Figure 14 The traditional classifier twitter visualization

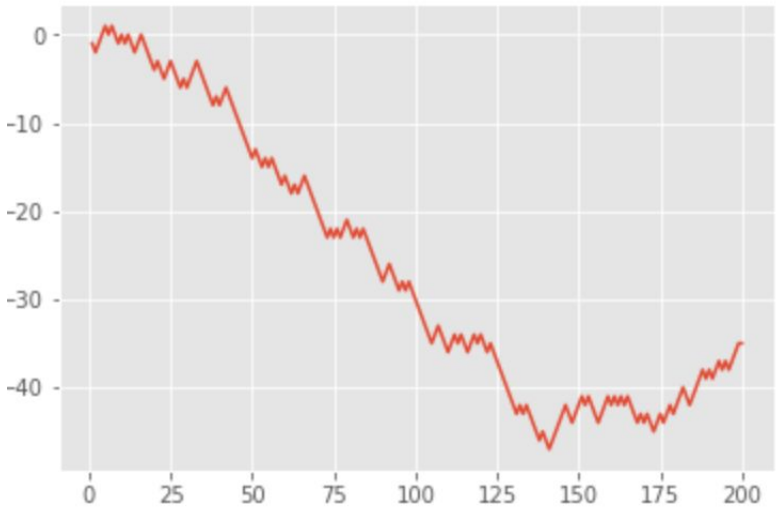

Figure 15 CNN Twitter application visualization

We can see that the sentimental tendency go down for both figures. Why? Is that because People don't like the word "Trump"? Maybe, but most of importantly, people are more likely to express their bad feeling on Internet other than good feeling.

However, the traditional classifier twitter application get a bigger negative number than CNN. I think that the result of CNN is more receivable as it has a better accuracy when training the the classifier model.

Reflection

The process used for this project can be summarized using the following steps:

1. Problem statement, and datasets were found
2. Data exploration and processing
3. Benchmark was created for classifier
4. Train the classifier using the processed data
5. A Twitter sentimental application was created by running the classifier

The second step is the most difficult part for me. I first use the traditional classifiers. In this part, I find it is hard to understand how to extract the feature. The most frequent 5k words were the feature. After understand that, I thought it is the same when I train CNN model.

However, in neural network, all the words should be transformed to matrix, then it can be trained. After reading some articles, I know how to transform word into sequence, some methods such as one-hot coding, word2vec, fasttext and so on. Plus, I know how to do word embedding just by using keras.

The second difficult part is to train the CNN model. There are many parements in the training process, I first apply the model parements most model will use. Such as the filter set to 16, kernel_size set to 4, dropout set to 0.3 and so on. After Reading some articles, I know set dropout to 0.5 sometimes get a better result. Try many times, I just get the a receivable result that can match the benchmark.

Improvement

Though I have done this project, but there are many things I can make in the future to improve the project:

1. I think I can use a transfer learning to train our words to get a better matrix. Like load pre-trained Word2vec and GloVe word embedding models from Google.
2. Find a better corpus if I want to do the Twitter sentimental analysis, such as the corpus from Twitter. Movie reviews have much information related to film, but for Twitter, it has everything. Puls, some special words or symbols,which may be removed by traditional method, can express some meaning in Twitter.
3. It said that LSTM is better for Natural Language problem. I tried several times, get a worse result. So, maybe because I didn’t get deep into it. I wish in the future, train a LSTM model for NLP problem.