Final Year Project

Sentiment Analysis on Movie reviews based on Twitter Data

Papul Ghosh
University of Kalyani, West Bengal, India

Abstract: Social media sites such as Twitter is a rich source of text examples expressing positive and negative sentiment. In this project we applied some of the machine learning algorithms to the collected twitter dataset to analyse the sentiment of each review and classify its polarity as positive or negative. Implementing various algorithms including Naive Bayes classifier, k-nearest neighbour (KNN) classifier, dictionary approach we had tried to improve the classification accuracy.

I. INTRODUCTION

Many public and private sectors are interested in extracting information regarding opinions that consist of subjective expressions across a variety of products or services. This helps companies determine strategies for improving the quality of their products or to assist decision makers. The main purpose of sentiment analysis is to conclude positive sentiments or negative sentiments from given opinionated text. We explore a number of questions in relation to the sentiment analysis problem. First, we examine dataset preprocessing specific to the natural language domain of tweets. We then evaluate a number of baseline linear models for sentiment analysis. Finally, we attempt to improve on the performance of our baseline models using annotated dictionary initialized with linear model weights. All the algorithms we consider in this project are supervised methods over unigram and bigram features.

II. Literature Review

Kim and Hovy [1] proposed a system that automatically identifies the people who hold opinions about that topic and the sentiment of each opinion. Liu B [2,3] in his handbook provided basics of sentiment analysis and subjectivity (or opinion mining). Due to a wide variety of practical applications, it has been a very active research area in recent years. In [4], Hadi and Saman look at two different datasets, one with binary labels, and one with multi-class labels. For the binary classification authors applied the bag of words, and skip-gram word2vec models followed by various classifiers, including random forest, SVM, and logistic regression. For the multi-class case, we implemented the recursive neural tensor networks (RNTN). To overcome the high computational cost of training the standard RNTN we introduce the lowrank RNTN, in which the matrices involved in the quadratic term of RNTN are substituted by symmetric low-rank matrices. In [5], Kai Sheng Tai evaluate the performance of a neural network classifier over bag-of-words features in relation to simpler linear classifiers. In [6], Akshita Jaitly and Sachi Ahuja tried to increase the efficiency of sentiment analysis. To achieve maximum accuracy, a sentence level analysis was performed by taking into account oxymoron (i.e. Figure of speech in which apparently contradictory terms appear in conjunction).

III. **Dataset**

We use the dataset on a Disney movie, which is publicly available on Kaggle.com. Here is a description of the data, provided by Kaggle:

> The labelled data set consists of 25,000 IMDB movie reviews, specially selected for sentiment analysis. Each of these row contains a review on the movie followed by a label which I either 0 or 1. The sentiment of reviews is binary, meaning the IMDB negative results in a sentiment score of 0, and positive rating have a sentiment score of 1. The 25,000 review set consists of equal number of positive and negative sentiment reviews that is, both counts 12500. Furthermore for assisting the dictionary based classifier we are using two annotated positive and negative wordlist. The positive wordlist contains 2007 distinct positive words and the count for negative word is 4783.

IV. **IMPLEMENTATION**

1. Importing Libraries:

```
#Importing libraries
from nltk.corpus import stopwords
from nltk.stem.wordnet import WordNetLemmatizer
import string
import re
import numpy as np
import random
from collections import Counter


```

2. Text cleaning:

The data was cleaned by first creating a function clean which receives each review from the main and preprocess the words. Stop words (using the Python Natural Language Toolkit), whitespaces, punctuations (using the Python package Regular- Expression) were then removed. Conversion from upper to lowercase form was also performed. Moreover, all the words of two words or less less are removed from the corpus.

```
# Cleaning the text sentences so that punctuation marks, stop words & digits are removed
def clean(doc):
    doc=doc.replace('<br />',' ')
    stop_free = " ".join([i for i in doc.lower().split() if i not in stop])
    punc_free = ''.join(ch for ch in stop_free if ch not in exclude)
    normalized = " ".join(lemma.lemmatize(word) for word in punc_free.split())
    processed = re.sub(r"\d+","",normalized)
    y = processed.split()
    for word in y:
        if len(word)<=2:
            del y[y.index(word)]
        word.lower()
    return y

```

3. Applying Bigram:

The python definition getwords splits each review into separate words and applies bigram approach to improve the sentiment analysis.

```
29
30  #this method returns important words after applying bigram from a sentence as list
31  def getwords(sentence):
32      w = sentence.split(" ")
33      w= w + [w[i]+' '+w[i+1] for i in range(len(w)-1)]
34      w= list(set(w))
35      return w
36
37
```

4. Trainset-Testset splitting:

The very next step after cleaning the reviews is to split the dataset into training and testing dataset. After shuffling 25000 reviews the negative and positive labelled reviews are stored in separated in two different lists. Then each of the lists are splitted in 3:2 ratio. The first 60% of both the positive and negative list are combined to generate trainset and the rest 40% constitutes the test set.

```
55  random.shuffle(ds)
56
57  poslines=[]
58  neglines=[]
59  for i in ds:
60      if i[1]==1:
61          poslines.append(i[0])
62      else:
63          neglines.append(i[0])
64
65  possplit=int(len(poslines)*0.6)
66  negsplit=int(len(neglines)*0.6)
67
68  train_clean_sentences= [(x,1) for x in poslines[:possplit]] + [(x,0) for x in neglines[:negsplit]]
69  #y_train= [1]*possplit + [0]*negsplit
70  y_train=np.append(y_train,[[1]*possplit + [0]*negsplit])
71
72  test_clean_sentences= [(x,1) for x in poslines[possplit:]] + [(x,0) for x in neglines[negsplit:]]
73  #y_test= [1]*(len(poslines)-possplit) + [0]*(len(neglines)-negsplit)
74  y_test=np.append(y_test,[[1]*(len(poslines)-possplit) + [0]*(len(neglines)-negsplit)])
75
```

5. Applying Models:

After preprocessing the dataset we applied 3 sentiment analysis method to predict the sentiments as positive or negative. Firstly we applied K Nearest Neighbour classifier, then Naïve Bayes Classifier and finally we tried to improve the Naïve Bayes model by assisting it with a positive and a negative wordlist.

a. K Nearest Neighbour Classifier (KNN):

The basic idea of this algorithm is as follows: if you calculate the similarity between the document to be classified and each of the training documents, the one among the k that is more similar will be indicating to which class or category the document to be classified should be assigned. The one that accumulates more points, will be the suitable candidate.
Here we are counting the frequency of each word in the training set, and assigning it to a python dictionary named freq. Next, we compare test reviews with each train reviews and counted the total frequency of the common words of each test and train review. To scale down the total frequency we have implemented logarithm tool (using the Python logarithm Toolkit) to calculate this score. Finally we have sorted the score list in descending order and considered the 5 (ie. we have considered K as 5) maximum values in the list. Depending on the labels of the 5 most common training reviews of the corresponding test review we have calculated the predicted label. The code is as follows:

```
77
78  train_len= len(train_clean_sentences)
79  freq={}
80  trainfeatures= []
81  for line,label in train_clean_sentences:
82      words= getwords(line)
83      for word in words:
84          freq[word]= freq.get(word, 0) + 1
85      trainfeatures.append((words,label))
86
87  predicted_labels_knn=np.array([])
88
89  for testline,testlabel in test_clean_sentences:
90      testwords= getwords(testline)
91      results=[]
92      for trainwords, trainlabel in trainfeatures:
93          #find all words in common between these two sentences
94          commonwords= [x for x in trainwords if x in testwords]
95          score= 0.0
96          for word in commonwords:
97              score += log(train_len/freq[word])
98          results.append((score, trainlabel))
99
100     results.sort(reverse=True)
101     toplab= [x[1] for x in results[:5]]
102     numones= toplab.count(1)
103     numnegones= toplab.count(0)
104
105     if numnegones> numones:
106         predicted_labels_knn=np.append(predicted_labels_knn,0)
107     else:
108         predicted_labels_knn=np.append(predicted_labels_knn,1)
109
```

b. Naïve Bayes Classifier:

The Naïve Bayes algorithm is one of the easiest to implement but still the most effective. This model is based on probabilistic theory, particularly in the Bayes theorem, which allows us to estimate the probability of an event out of the probability that another event will occur, on which the first depends. Here we are creating two python dictionary poswords and negwords that store the frequencies of each word in the positive and negative reviews present in the training dataset. Although we are removing the words with very high (we the minimum limit as 10) as well as very low (we the maximum limit as 2000) frequencies respectively as they are considered to be valueless in the sentiment prediction. Thus we are predicting the overall sentiment of each review by considering the positiveness or negativeness of each word in the test reviews.

```
75
76  poswords={}
77  negwords={}
78
79
80  for line,label in train_clean_sentences:
81      words= getwords(line)
82      for word in words:
83          if label==1: poswords[word]= poswords.get(word, 0) + 1
84          if label==0: negwords[word]= negwords.get(word, 0) + 1
85
86  poswords = { k : v for k,v in poswords.items() if (v>=10 & v<=2000)}
87  negwords = { k : v for k,v in negwords.items() if (v>=10 & v<=2000)}
88
89  predicted_labels_NB=np.array([])
90
91  for testline,testlabel in test_clean_sentences:
92      testwords= getwords(testline)
93      totpos, totneg= 0.0, 0.0
94      for word in testwords:
95          a= poswords.get(word,0.0)# + 1.0
96          b= negwords.get(word,0.0)# + 1.0
97          if ((a!=0.0)|(b!=0.0)):
98              totpos+= a/(a+b)
99              totneg+= b/(a+b)
100     if (totpos>totneg):
101         predicted_labels_NB=np.append(predicted_labels_NB,1)
102     else:
103         predicted_labels_NB=np.append(predicted_labels_NB,0)
104
```

c. Naïve Bayes Classifier with annotated wordlist:

This method used two annotated dictionary containing dedicated positive and negative wordlists. This wordlists played substitute role in the classification algorithm assisting the Naïve Bayes classifier. If a test review word be present either in the positive training wordlist (that is the poswords list) or in the negitive training wordlist (that is the negwords list), then the prediction will be performed based on the typical Naïve Bayes algorithm. But if the word be present in both the poswords and negwords list, then the label prediction will depend on the presence of that particular word in the assisting annotated dictionaries. This approach will definitely improve the accuracy of the sentiment prediction.

```
poswords={}
negwords={}


for line,label in train_clean_sentences:
    words= getwords(line)
    for word in words:
        if label==1: poswords[word]= poswords.get(word, 0) + 1
        if label==0: negwords[word]= negwords.get(word, 0) + 1

poswords = { k : v for k,v in poswords.items() if (v>=10 & v<=max(poswords.values())*2*3)}
negwords = { k : v for k,v in negwords.items() if (v>=10 & v<=max(negwords.values())*2/3)}

predicted_labels_NB=np.array([])

for testline,testlabel in test_clean_sentences:
    testwords= getwords(testline)
    totpos, totneg= 0.0, 0.0
    a,b=0.0,0.0
    for word in testwords:
        if word in poswords and word in negwords:
            if word in posTokens:
                a= poswords.get(word)
            elif word in negTokens:
                b= negwords.get(word)
        elif word in poswords:
            a= poswords.get(word)
        elif word in negwords:
            b= negwords.get(word)
      #  else:
       #     a=0.0
        #    b=0.0
        #a= poswords.get(word,0.0)# + 1.0
        #b= negwords.get(word,0.0)# + 1.0
        if ((a!=0.0)|(b!=0.0)):
            totpos+= a/(a+b)
            totneg+= b/(a+b)
    if (totpos>totneg):
        predicted_labels_NB=np.append(predicted_labels_NB,1)
    else:
        predicted_labels_NB=np.append(predicted_labels_NB,0)
```

6. Calculate Accuracy:

Finally after implementing various classifier models we compare each element of the actual and predicted label list and thus calculate the accuracy percentage. The accuracy calculation for the Naïve-Bayes classifier is shown here:

```
105 print ("\n----------------PREDICTIONS BY NAIVE-BAYES------------------")
106
107 ap_pp=0
108 ap_pn=0
109 an_pp=0
110 an_pn=0
111 acc=0
112
113 for i in range(len(y_test)):
114                 if predicted_labels_NB[i]==y_test[i]:
115                     acc+=1
116                     if(y_test[i]==1):
117                         ap_pp+=1
118                     else:
119                         an_pn+=1
120                 else:
121                      if(y_test[i]==1):
122                         ap_pn+=1
123                      else:
124                         an_pp+=1
125 accuracy= acc*100/len(y_test)
126 print('    Pos      Neg')
127 print('  ',ap_pp,'    ',an_pp)
128 print('  ',ap_pn,'    ',an_pn)
129 print('Accuracy: ', accuracy,'%')
130
```

V. CONCLUSION

With the large amount of data driving business decisions, accuracy is imperative. Here our task was divides=d into two parts. The first part consists preprocessing of data and splitting the single dataset in training and testing set. In the second part we have implemented some supervised sentiment classifier models and compared the predicted labels with the actual sentiments and thus calculated the accuracy. We have achieved a satisfactory accuracy of 88% in Naïve Bayes classifier. Unfortunately the overall performance of predicting the negative reviews is deficient. In the later period of time we will focus on improving this issue as well as will try to implement some more efficient approach that will improve the model performance. We will also focus on identifying and altering the misspelled words in the comments to get an e_ective sentiment analysis. This will lead to a better understanding of human language opinions and better processing of human thought by a machine.

III. REFERENCES

[1] Kim S-M, Hovy E (2004) Determining the sentiment of opinions In: Proceedings of the 20th international conference on Computational Linguistics, page 1367. Association for Computational Linguistics, USA.
[2] Liu B (2010) Sentiment analysis and subjectivity In: Handbook of Natural Language Processing, Second Edition. Taylor and Francis Group, Boca.
[3] Liu B, Hu M, Cheng J (2005) Opinion observer: Analysing and comparing opinions on the web In: Proceedings of the 14th International Conference on World Wide Web, WWW "05, 342–351. ACM, New York, NY, USA.
[4] Pouransari, H., & Ghili, S. (2014). *Deep learning for sentiment analysis of movie reviews*. Technical report, Stanford University.
[5] Tai, K. S. (2013). Sentiment Analysis of Tweets: Baselines and Neural Network Models. *CS229 Final Project December*, *13*.
[6] Jaitly, A., & Ahuja, S. (2018). IMPROVING THE ACCURACY FOR SENTENCE LEVEL SENTIMENT ANALYSIS. *International Journal of Advanced Research in Computer Science*, *9*(4).