Final Project Report

Group Members: Shiyi Yang, Yueru He

Use News Headlines to Predict Stock Movements

Introduction

When we talk about the stock, despite that people can understand the movement of the stock market as simple as reading a graph with trends, it is probably much harder to forecast what will happen in a day after, or even further. Traders at the Stock Exchange Center pay close attention to financial news every day in order to find reliable indicators of how the stock will move with their past experiences. But is it possible to forecast the movement in a systematic way, for example, through machine learning algorithms? In this final project, our team aim at exploring two algorithms to justify whether we can use the machine learning tools to predict the stock price movements precisely. If we are able to do so, people who do not have enough knowledge about financial and business can also predict the stock price using programming languages.

Paper & Data Sources

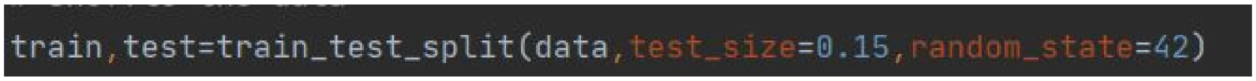
The paper we based on is *Using News Articles to Predict Stock Price Movements* by Professor Győző Gidófalvi(Gidófalvi,2004). As the title suggests, professor Gidófalvi uses machine learning algorithms to predict the movements of several stocks. The particular algorithm he uses is the Naive Bayesian algorithm. The emphasis is on labeling the articles based on the movement of stocks as precisely as possible. For example, words like “bankruptcy” and “down” are likely indicating the down movements of the stock, and words like “soar” and “increase” are likely to imply the upward movements. The key to this paper is to use Naive Bayesian to maximize the probability of correctly classifying the movement type of the article based on the content of this article. In this project, we will use the same Naive Bayesian methods to predict the stock price movements, except that we are not using the whole content of those articles, but just the headlines of them.

Unfortunately, we didn’t find the original data Professor Gidófalvi uses in his paper, but we find similar data with stock movements labeled as “1” if the stock price moves up or stay unchanged, “0” if it goes down. The stock price index is from Dow Jones Industrial Average(DIJA). Besides, the dataset includes the top 25 news headlines each day from 2008-06-08 to 2016-07-01, which provides us enough data to train and test our algorithms. In order to increase the accuracy of our predictions, we also use a parser to alter the data a little by emphasizing potential useful keywords more and making light of unimportant words like conjunctions and pronouns.

Code & Running Results

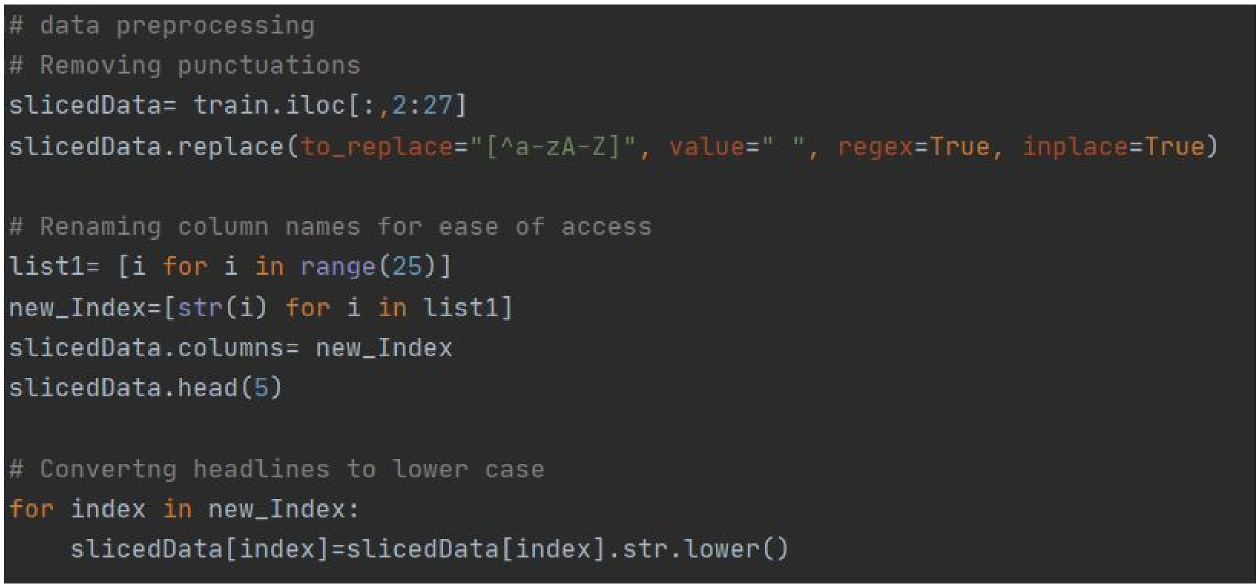
Besides the Naive Bayesian algorithm, we use LSTM and CNN on this dataset as well to compare its accuracy with the Naive Bayesian algorithm.

We split the dataset into a train set and test set after shuffling it:

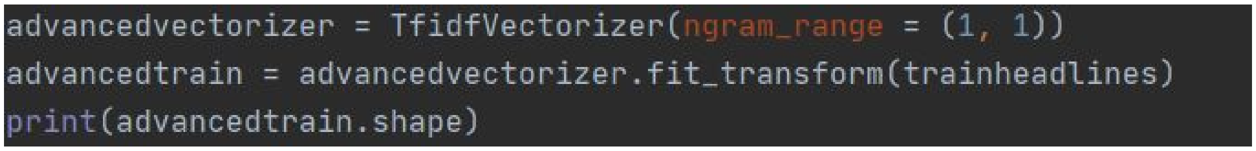


```
train,test=train_test_split(data,test_size=0.15,random_state=42)
```

Firstly we use the Keras tokenizer and TFIDF method to eliminate noise and alter the weights of the original dataset:

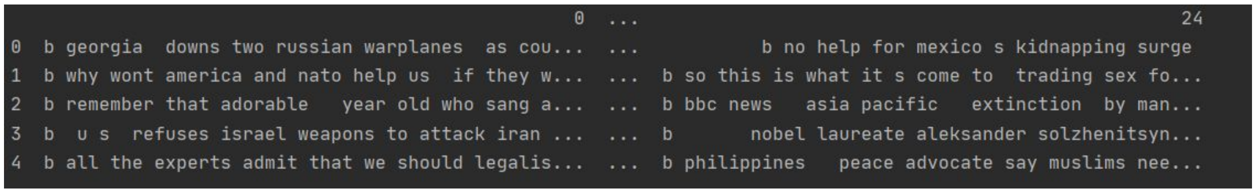


```python
# data preprocessing
# Removing punctuations
slicedData= train.iloc[:,2:27]
slicedData.replace(to_replace="[^a-zA-Z]", value=" ", regex=True, inplace=True)

# Renaming column names for ease of access
list1= [i for i in range(25)]
new_Index=[str(i) for i in list1]
slicedData.columns= new_Index
slicedData.head(5)

# Convertng headlines to lower case
for index in new_Index:
    slicedData[index]=slicedData[index].str.lower()
```



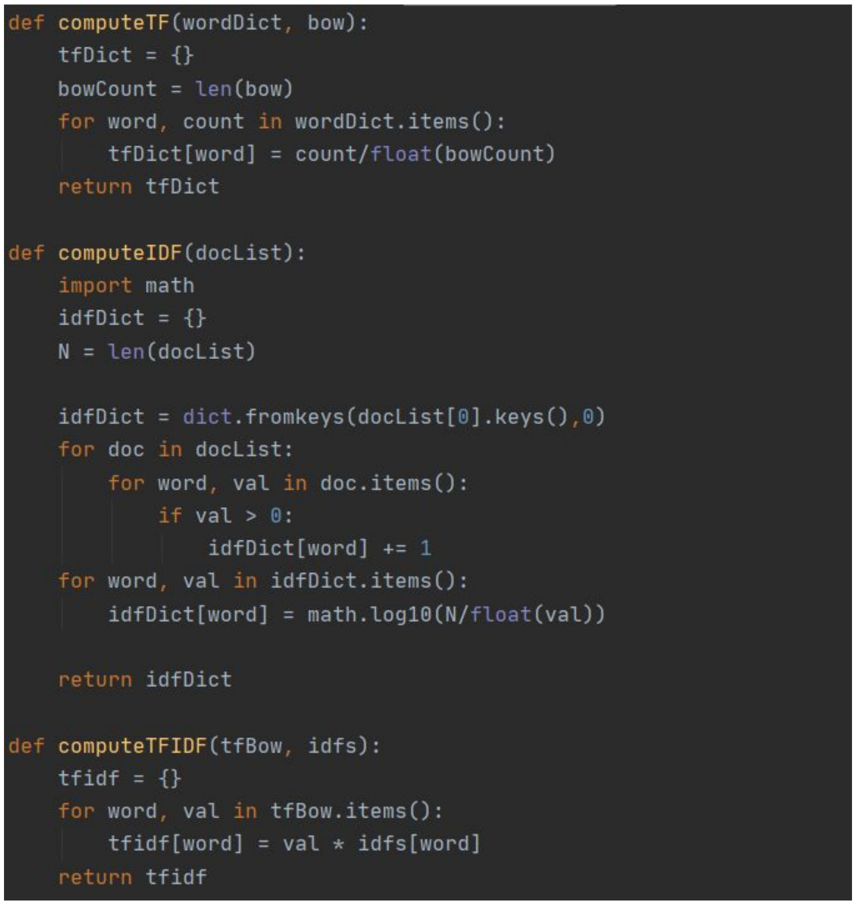
```python
advancedvectorizer = TfidfVectorizer(ngram_range = (1, 1))
advancedtrain = advancedvectorizer.fit_transform(trainheadlines)
print(advancedtrain.shape)
```

Now from the following, we can see that all the punctuations in the headlines are removed, and all words are converted to lowercase so that the machine won't treat words like "Plane" and "plane" as different ones.



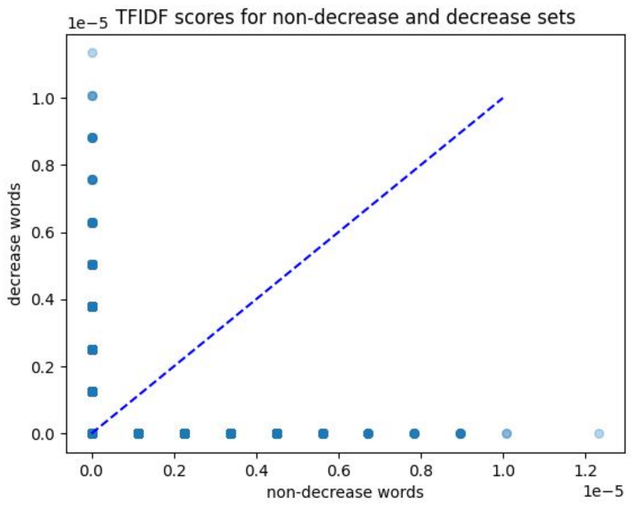
```
                                                  0  ...                                                 24
0  b georgia  downs two russian warplanes  as cou...  ...           b no help for mexico s kidnapping surge
1  b why wont america and nato help us  if they w...  ...  b so this is what it s come to  trading sex fo...
2  b remember that adorable   year old who sang a...  ...  b bbc news   asia pacific   extinction  by man...
3  b  u s  refuses israel weapons to attack iran ...  ...  b       nobel laureate aleksander solzhenitsyn...
4  b all the experts admit that we should legalis...  ...  b philippines   peace advocate say muslims nee...
```

Also with TFIDF, we split the dataset into the "non-decrease set" and "decease set," which represent days that the stock price goes up and goes down respectively. Then we use these two sets as our corpus for TFIDF. This is a less efficient way than split each headline as treating one document, but we so far had failed to do so. We set up the following functions to compute TFIDF scores step by step:

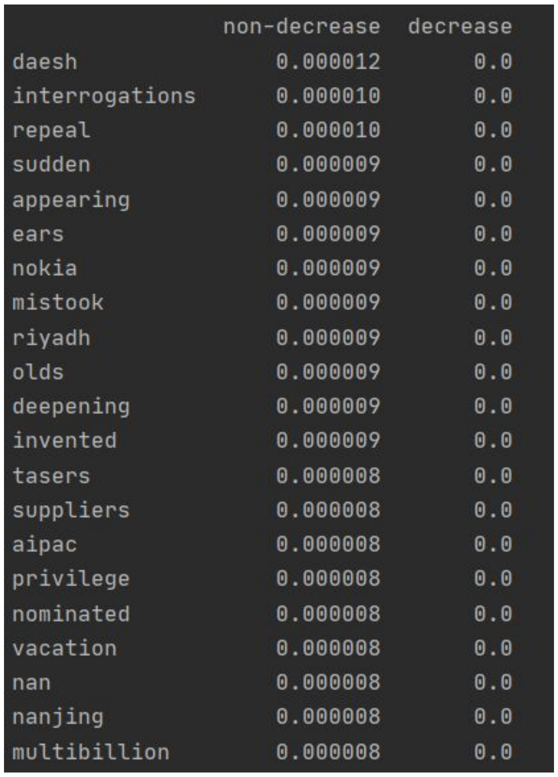


```
def computeTF(wordDict, bow):
    tfDict = {}
    bowCount = len(bow)
    for word, count in wordDict.items():
        tfDict[word] = count/float(bowCount)
    return tfDict

def computeIDF(docList):
    import math
    idfDict = {}
    N = len(docList)

    idfDict = dict.fromkeys(docList[0].keys(),0)
    for doc in docList:
        for word, val in doc.items():
            if val > 0:
                idfDict[word] += 1
    for word, val in idfDict.items():
        idfDict[word] = math.log10(N/float(val))

    return idfDict

def computeTFIDF(tfBow, idfs):
    tfidf = {}
    for word, val in tfBow.items():
        tfidf[word] = val * idfs[word]
    return tfidf
```

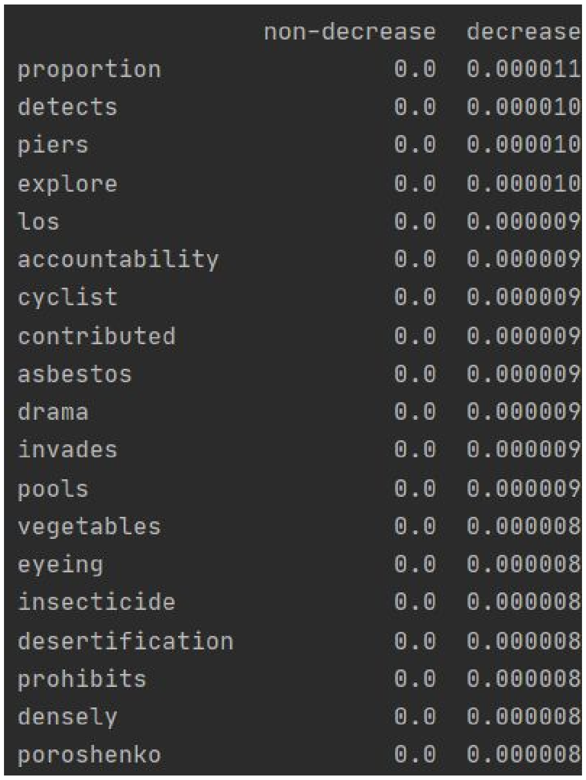
We get over 30,000 unique words from this step, and we have the figure below to illustrate our findings:



The points are so distinctive, but we can see that the TFIDF scores are overlapping with each other: the less transparent the point is, the more data points there are. We imply that the highly distinctive performance is caused by a large number of different words but only a few documents. But still, we are able to have some conclusions on it. Here are the words with the highest weights in non-decrease set and decrease set, separately:



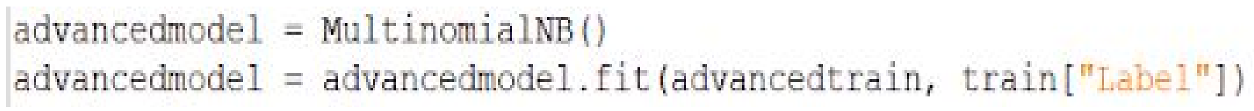
```
                non-decrease  decrease
daesh               0.000012       0.0
interrogations      0.000010       0.0
repeal              0.000010       0.0
sudden              0.000009       0.0
appearing           0.000009       0.0
ears                0.000009       0.0
nokia               0.000009       0.0
mistook             0.000009       0.0
riyadh              0.000009       0.0
olds                0.000009       0.0
deepening           0.000009       0.0
invented            0.000009       0.0
tasers              0.000008       0.0
suppliers           0.000008       0.0
aipac               0.000008       0.0
privilege           0.000008       0.0
nominated           0.000008       0.0
vacation            0.000008       0.0
nan                 0.000008       0.0
nanjing             0.000008       0.0
multibillion        0.000008       0.0
```



```
                  non-decrease  decrease
proportion                 0.0  0.000011
detects                    0.0  0.000010
piers                      0.0  0.000010
explore                    0.0  0.000010
los                        0.0  0.000009
accountability             0.0  0.000009
cyclist                    0.0  0.000009
contributed                0.0  0.000009
asbestos                   0.0  0.000009
drama                      0.0  0.000009
invades                    0.0  0.000009
pools                      0.0  0.000009
vegetables                 0.0  0.000008
eyeing                     0.0  0.000008
insecticide                0.0  0.000008
desertification            0.0  0.000008
prohibits                  0.0  0.000008
densely                    0.0  0.000008
poroshenko                 0.0  0.000008
```

All scores are pretty low due to the high volume of different words. From the non-decease set, TOP 3 words are “DAESH,” or the Islamic State of Iraq and the Levant, interrogations, and repeal; the TOP 3 words from the deceased set are proportion, detects, and piers. Some are politically related, but others don’t make much sense for stock movements.

Then we use Naive Bayesian to predict the stock movements:

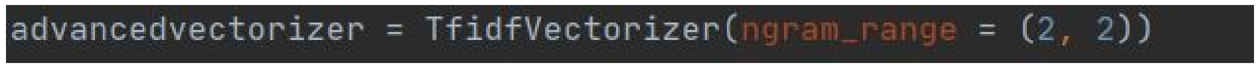


```
advancedmodel = MultinomialNB()
advancedmodel = advancedmodel.fit(advancedtrain, train["Label"])
```

There are built-in functions for Naive Bayesian, and we choose MultinomialNB for our project since the data are represented as word vector counts and is especially suitable for qualitative variables like vocabularies.

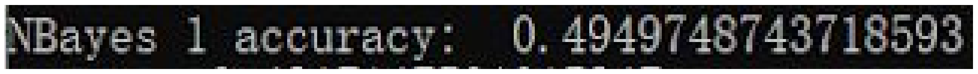
The original paper uses the “Rainbow Naive Bayesian Classifier” package in C language. Since we use Python as the programming language for our project, we are able to duplicate the procedure the Rainbow Naive Bayesian Classifier package does to the data with tokenizer, TFIDF methods, and manually split train and test sets.

We also use 2-gram modeling to capture the contiguous sequence of 2 items in headlines at a time. As follows:

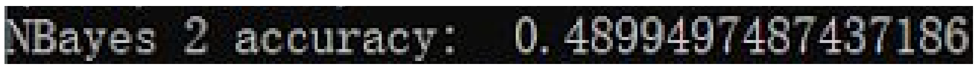


```
advancedvectorizer = TfidfVectorizer(ngram_range = (2, 2))
```

We get similar accuracy scores for both models, but the 1-gram model is a little more accurate than the 2-gram model :

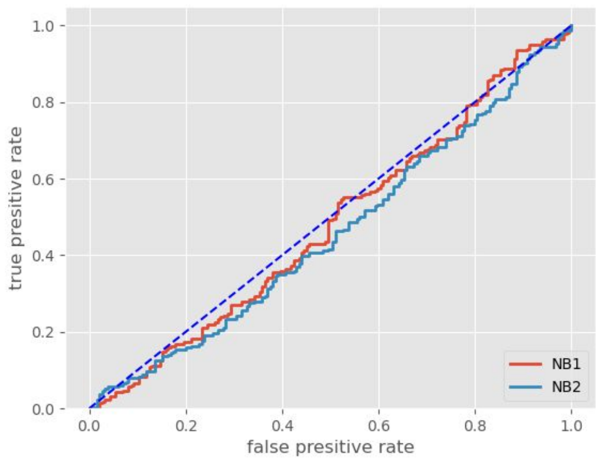


```
NBayes 1 accuracy:  0.4949748743718593
```



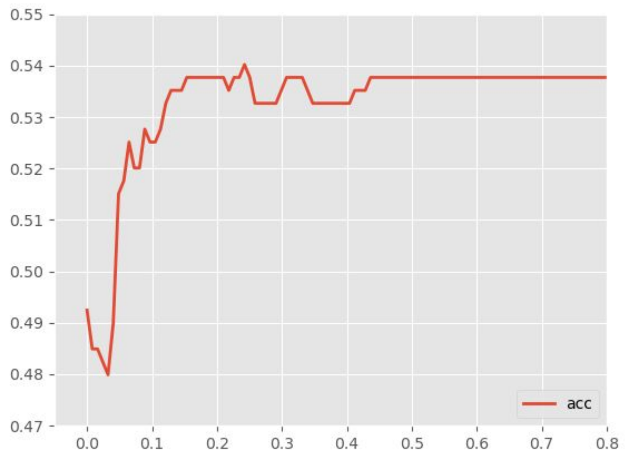
```
NBayes 2 accuracy:  0.4899497487437186
```

The ROC curves of Naive Bayesian are as follows:



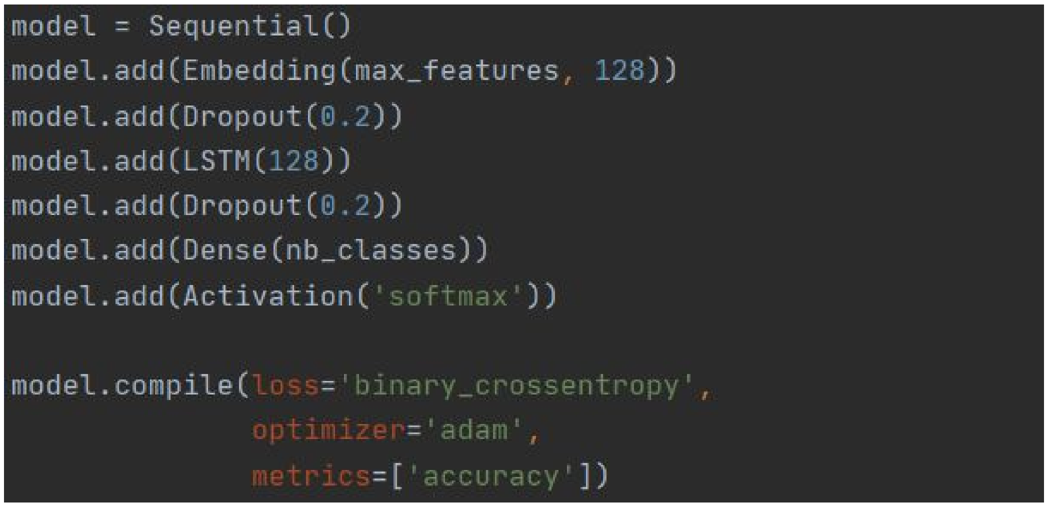
The performance of the 1-gram model and the 2-gram model is pretty similar, as suggested before.

We then take a closer look at how the Naive Bayesian Classifier performs with different Lidstone Smoothing (pseudo-count) parameters. It turns out that the accuracy increases by 3% with the smoothing parameter equal to 0.2 instead of not smoothing at all. The reason is that with no smoothing, even one word in the text being examined does not appear in the headlines would cause the probability of the whole text to be zero. Smoothing eliminates this probability.

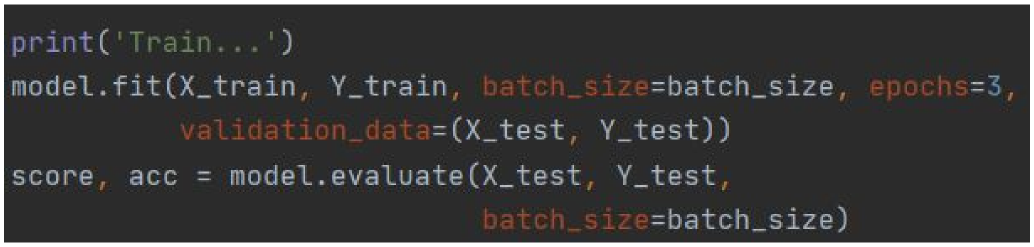


Then we use LSTM and CNN on the same dataset:

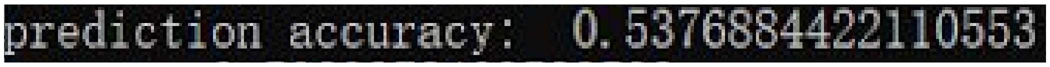
LSTM



```
model = Sequential()
model.add(Embedding(max_features, 128))
model.add(Dropout(0.2))
model.add(LSTM(128))
model.add(Dropout(0.2))
model.add(Dense(nb_classes))
model.add(Activation('softmax'))

model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

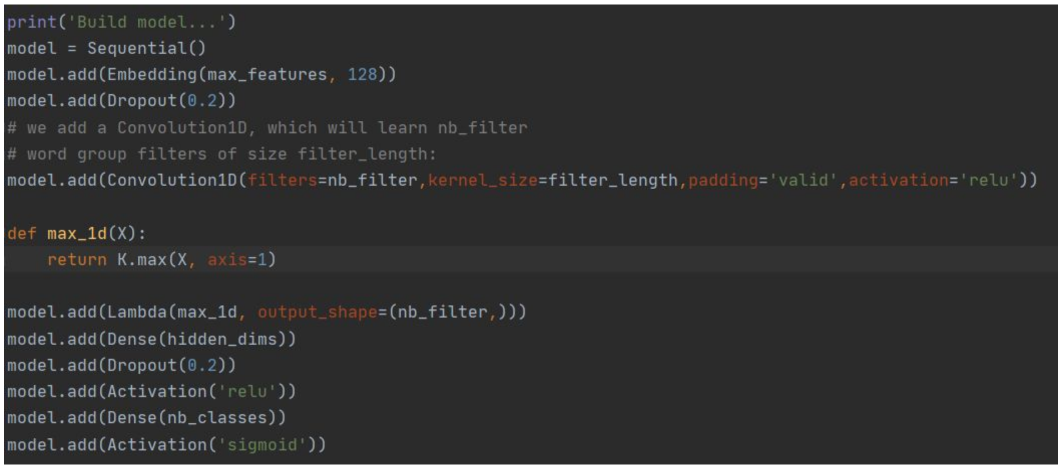


```
print('Train...')
model.fit(X_train, Y_train, batch_size=batch_size, epochs=3,
          validation_data=(X_test, Y_test))
score, acc = model.evaluate(X_test, Y_test,
                            batch_size=batch_size)
```

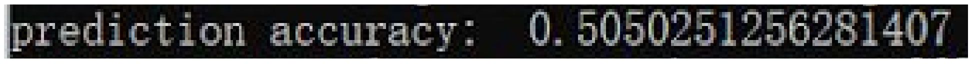


```
prediction accuracy:  0.5376884422110553
```

CNN



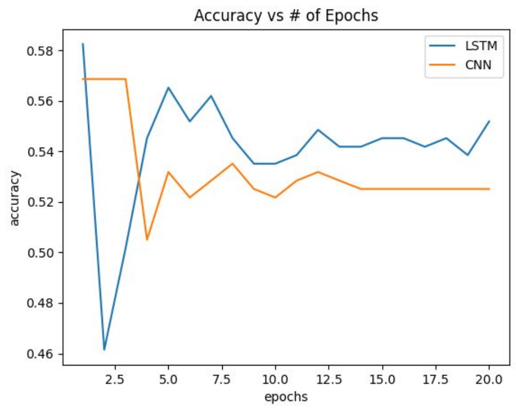
```
print('Build model...')
model = Sequential()
model.add(Embedding(max_features, 128))
model.add(Dropout(0.2))
# we add a Convolution1D, which will learn nb_filter
# word group filters of size filter_length:
model.add(Convolution1D(filters=nb_filter,kernel_size=filter_length,padding='valid',activation='relu'))

def max_1d(X):
    return K.max(X, axis=1)

model.add(Lambda(max_1d, output_shape=(nb_filter,)))
model.add(Dense(hidden_dims))
model.add(Dropout(0.2))
model.add(Activation('relu'))
model.add(Dense(nb_classes))
model.add(Activation('sigmoid'))
```

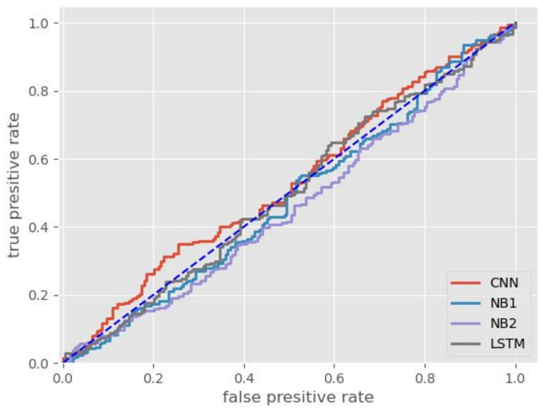


```
prediction accuracy:  0.5050251256281407
```

The accuracy of both algorithms converges after the number of epoch equals 6. We can observe that LSTM has higher accuracy than CNN after convergence, although it has much lower accuracy at the second epoch.

We compare the ROC curves of three algorithms and get the following, which shows that CNN has slightly better performance in terms of ROC, which counters the situation shows in the accuracy vs epoch plot. The reason for this, as we suppose, could be that CNN achieves higher AUC in the cost of a higher false negatives rate than that of the LSTM.





Compare Results & Discussion

We originally ran all models with no modifications to the text content in the dataset, which leads to poor performance in each algorithm. Then we use the tokenizers and the TFIDF method to improve the dataset, which increase the accuracy of all three algorithms significantly. We, therefore, conclude that text preprocessing is very helpful when we deal with natural languages.

More importantly, by comparison of the ROC curves of these three algorithms, we can see that Naive Bayes has worse performance than the other two. However, when we are running the algorithms, the Naive Bayes classifier gives results in a shorter time. Although LSTM and CNN end up with higher accuracy, Naive Bayes is certainly easier to implement than LSTM and CNN.

We also notice that highly politically related words, like DAESH, privilege, prohibits are influential. Besides, the stock market is also sensitive to highly positive or negative words, like invades, interrogation. Those words from the news are indicative and can indeed help one to decide to buy or sell the stock in general.

Further Improvements

We have yet achieved the goal of assigning more precise TFIDF scores for each characteristic words in the headline, so the next step is to improve on this. In addition, we need to build a parser to analyze the "type" of the news headlines so that we won't get indicators that do not make sense. This requires more knowledge of NLP. Unfortunately, we do not have a strong ability to do so for now, but this is a necessary step to improve the accuracy of the classifiers.

Reference:

Gidófalvi, G. (2004, December). Using News Articles to Predict Stock Price Movements. Retrieved September 18, 2020, from http://cseweb.ucsd.edu/~elkan/254spring01/gidofalvirep.pdf