**CSCI63 Final Project Report: Natural Language Processing with Kafka**

**Overview**

Whereas traditional natural language processing uses millions of lines of text to train models, Twitter provides a unique advantage to training because each message is limited to a 140 character limit. Though the volume of readily available data than to train model on, but on the other hand each word in a Twitter message has importance and thus the signal to noise ratio should be much higher.

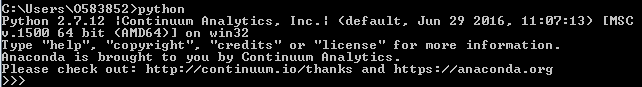
**High Frequency Data Download Installation**

*Configuration: 64-bit Windows 7, Anaconda Python*

*Requires: Bloomberg Terminal (for data source)*

Anaconda Python for Windows can be downloaded here: <https://www.continuum.io/downloads#windows>

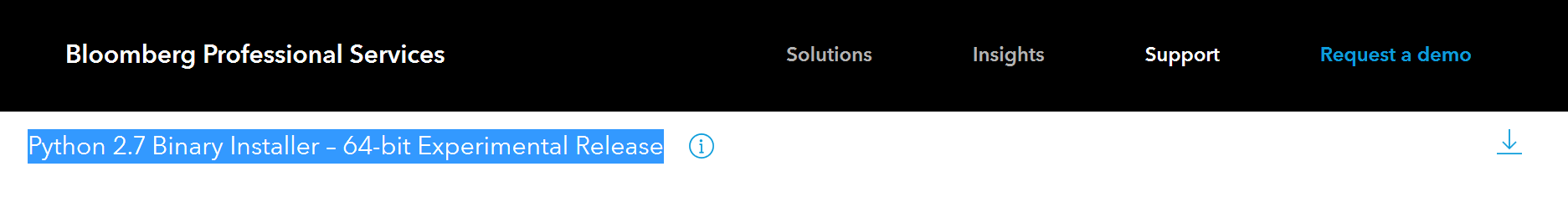
Our machine installed Anaconda Python using default settings to the directory: C:\FAST\. The version of Python installed was 2.7.



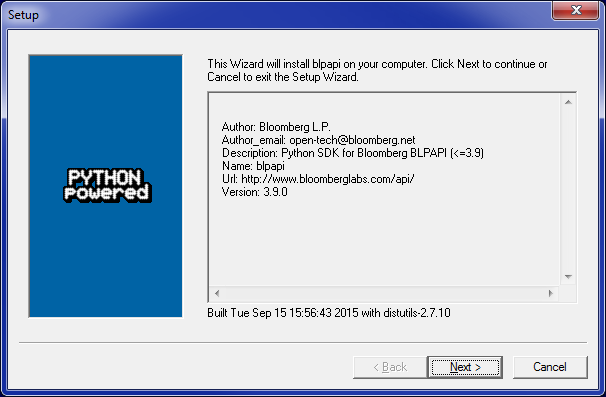
The pandas library is used and must be installed by typing “conda install pandas” in command prompt. The scikit-learn package was also installed by typing “conda install scikit-learn” in command prompt.

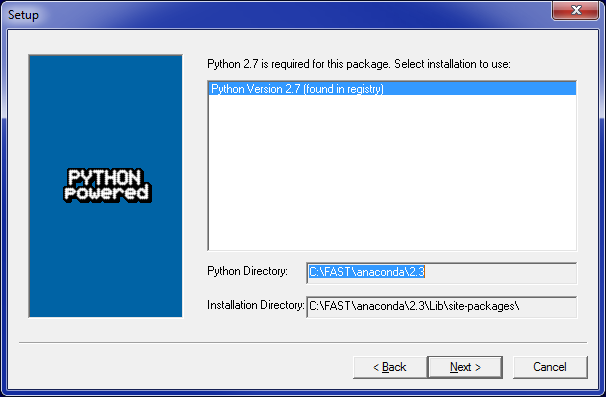
Next, download the self-extracting Bloomberg blpapi library executable from the Bloomberg website:

<https://www.bloomberg.com/professional/support/api-library/>

****

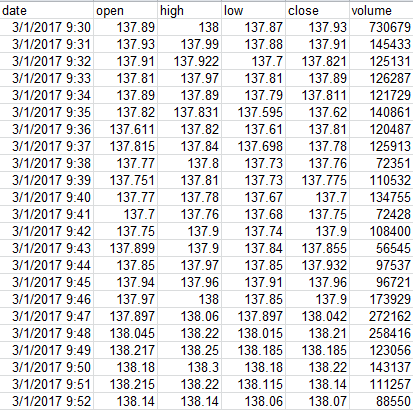
Run the downloaded file blpapi\_python\_3.9.0-win-amd64-py2.7.exe:





Our downloadIntraday.py python script imports a class called pybbg which is defined in our own file ***pybbg.py***. The pybbg class handles the interfacing with Bloomberg API. The script generates three csv files and saves to the directory H:/Course Docs/Big Data/Final Project/Results/IntradayPrices/. The following intraday information with minute-by-minute frequency on the three stocks is downloaded: open price, high price, low price, closing price, and volume. Intraday data is downloaded by running “python downloadIntraday.py” in command prompt.

**Sample File: H:/Course Docs/Big Data/Final Project/Results/IntradayPrices/IntradayPrice.AAPL.csv**



**downloadIntraday.py Code:**

**import** pybbg  
**from** datetime **import** datetime  
**from** datetime **import** timedelta  
**import** pandas **as** pd  
  
export\_path = 'H:/Course Docs/Big Data/Final Project/Data/IntradayPrices/

bbg = pybbg.Pybbg()  
bbg.service\_refData()  
tickers = ['AAPL', 'FB', 'TSLA']  
fld\_list = ['open', 'high', 'low', 'close', 'volume']  
  
end\_date = datetime(datetime.today().year, datetime.today().month, datetime.today().day) + timedelta(days=-1)  
  
**for** ticker **in** tickers:  
 df = bbg.bdib(ticker+' US Equity', fld\_list, datetime(2017, 3, 1), end\_date, eventType='TRADE', interval=1)  
 df.index = [x - pd.to\_timedelta(5, unit='h') **for** x **in** df.index]  
 df[fld\_list].to\_csv(export\_path+'IntradayPrice.'+ticker+'.csv', cols=fld\_list, index\_label='date')

**pybbg.py Code:**

# Adapted code from github.com/kyuni22/pybbg  
**from** \_\_future\_\_ **import** print\_function  
**import** blpapi  
**from** collections **import** defaultdict  
**from** pandas **import** DataFrame  
**import** pandas **as** pd  
**import** sys  
  
  
**class Pybbg**():  
 **def** \_\_init\_\_(self, host='localhost', port=8194):  
 *"""  
 Starting bloomberg API session  
 close with session.close()  
 """* # Fill SessionOptions  
 sessionOptions = blpapi.SessionOptions()  
 sessionOptions.setServerHost(host)  
 sessionOptions.setServerPort(port)  
  
 self.initialized\_services = set()  
  
 # Create a Session  
 self.session = blpapi.Session(sessionOptions)  
  
 # Start a Session  
 **if not** self.session.start():  
 **print**("Failed to start session.")  
  
 self.session.nextEvent()  
  
 **def service\_refData**(self):  
 *"""  
 init service for refData  
 """* **if** '//blp/refdata' **in** self.initialized\_services:  
 **return  
  
 if not** self.session.openService("//blp/refdata"):  
 **print**("Failed to open //blp/refdata")  
  
 self.session.nextEvent()  
  
 # Obtain previously opened service  
 self.refDataService = self.session.getService("//blp/refdata")  
  
 self.session.nextEvent()  
  
 self.initialized\_services.add('//blp/refdata')  
  
 **def bdib**(self, ticker, fld\_list, startDateTime, endDateTime, eventType='TRADE', interval=1):  
 *"""  
 Get one ticker (Only one ticker available per call); eventType (TRADE, BID, ASK,..etc); interval (in minutes)  
 ; fld\_list (Only [open, high, low, close, volumne, numEvents] available)  
 return pandas dataframe with return Data  
 """* self.service\_refData()  
 # Create and fill the request for the historical data  
 request = self.refDataService.createRequest("IntradayBarRequest")  
 request.set("security", ticker)  
 request.set("eventType", eventType)  
 request.set("interval", interval) # bar interval in minutes  
 request.set("startDateTime", startDateTime)  
 request.set("endDateTime", endDateTime)  
  
 # Send the request  
 self.session.sendRequest(request)  
 data = defaultdict(dict)  
 # Process received events  
 **while** True:  
 # We provide timeout to give the chance for Ctrl+C handling:  
 ev = self.session.nextEvent(500)  
 **for** msg **in** ev:  
 barTickData = msg.getElement('barData').getElement('barTickData')  
 **for** i **in** range(barTickData.numValues()):  
 **for** j **in** range(len(fld\_list)):  
 data[(fld\_list[j])][barTickData.getValue(i).getElement(0).getValue()] = barTickData.getValue(  
 i).getElement(fld\_list[j]).getValue()  
  
 **if** ev.eventType() == blpapi.Event.RESPONSE:  
 # Response completly received, so we could exit  
 **break** data = DataFrame(data)  
 data.index = pd.to\_datetime(data.index)  
 **return** data  
  
 **def stop**(self):  
 self.session.stop()  
  
  
**def isstring**(s):  
 # if we use Python 3  
 **if** (sys.version\_info[0] == 3):  
 **return** isinstance(s, str)  
 # we use Python 2  
 **return** isinstance(s, basestring)  
  
  
**def processMessage**(msg):  
 SECURITY\_DATA = blpapi.Name("securityData")  
 SECURITY = blpapi.Name("security")  
 FIELD\_DATA = blpapi.Name("fieldData")  
  
 securityDataArray = msg.getElement(SECURITY\_DATA)  
 **for** securityData **in** securityDataArray.values():  
 **print**(securityData.getElementAsString(SECURITY))  
 fieldData = securityData.getElement(FIELD\_DATA)  
 **for** field **in** fieldData.elements():  
 **for** i, row **in** enumerate(field.values()):  
 **for** j **in** range(row.numElements()):  
 e = row.getElement(j)  
 **print**("Row %d col %d: %s %s" % (i, j, e.name(), e.getValue()))

**Dictionary Sentiment Model**

One way to extract information from financial-related text is to perform sentiment or tone analysis. We apply a financial sentiment dictionary using a bag-of-words approach by assigning each word and word vector (tweet) a positive or negative value. Whereas the Harvard Psychosociological Dictionary, or Harvard-IV-4 TagNeg (H4N), is the common source for word classification, Loughran and McDonald [2011] found that it substantially misclassifies words when gauging tone in financial applications. The two professors from University of Notre Dame found that 73.8% of negative word counts according to the Harvard list were attributable to words that are not typically not negative in financial context (e.g. *tax, cost, capital, board, liability, foreign*, etc.). In addition, they used different variations or inflections of words (e.g. *accidental*, *accidentally*, and *accidents* from the base word *accident*). We found that the list of FinNeg (2,355 words) and FinPos (354 words) were a little imbalanced. Perhaps this provides some empirical backing to the Anna Karenina principle: “Happy families are all alike; every unhappy family is unhappy in its own way.” The word list and other general information are available here: <http://www3.nd.edu/~mcdonald/Word_Lists.html>.

Not to be outdone by University of Notre Dame, Harvard has updated their word list. Whereas the original Harvard-IV-4 TagNeg list contained 1,045 positive and 1,160 negative words, the new list contains 1,637 positive and 2,006 negative words. This list and other general information are available here: <http://www.wjh.harvard.edu/~inquirer/spreadsheet_guide.htm>. We will be testing whether the domain-specific dictionary or general dictionary produces a more accurate prediction on our StockTwits dataset.

**Calculations**

Each tweet ***n*** is cleaned and formatted into a list of all-caps words ***wi***. This is used to calculate the single-tweet sentiment score ***Sn***.

The sentiment score can range between -1 and +1. If a tweet contained three positive financial words and two negative words, then the sentiment score would be +0.2.

Let the number of tweets on a distinct day ***d*** be defined as ***nd***. When we aggregate across tweets for a company on each distinct day ***d***, we calculate the following:

We are using 45 days of historical tweets (between 3/1 and 4/30). In order to Note that we use 0.022.

**Dictionary Sentiment Model Installation**

The Financial (Loughran and McDonald) word dictionary was downloaded from this website: <http://www3.nd.edu/~mcdonald/Word_Lists_files/LoughranMcDonald_MasterDictionary_2014.xlsx> and saved to the following location: H:/Course Docs/Big Data/Final Project/Docs/LoughranMcDonald\_MasterDictionary\_2014.xlsx. Columns H and I indicate negative and positive financial words, respectively.

The Harvard word dictionary was downloaded from this website: <http://www.wjh.harvard.edu/~inquirer/inquirerbasic.xls> and saved to the following location: H:/Course Docs/Big Data/Final Project/Docs/inquirerbasic.xls. We made a slight change to this file. Because different connotations of the same word were listed as multiple rows in the spreadsheet, we removed the ‘#X’ tag which denotes multiple entries, and the first entry of each word was used.

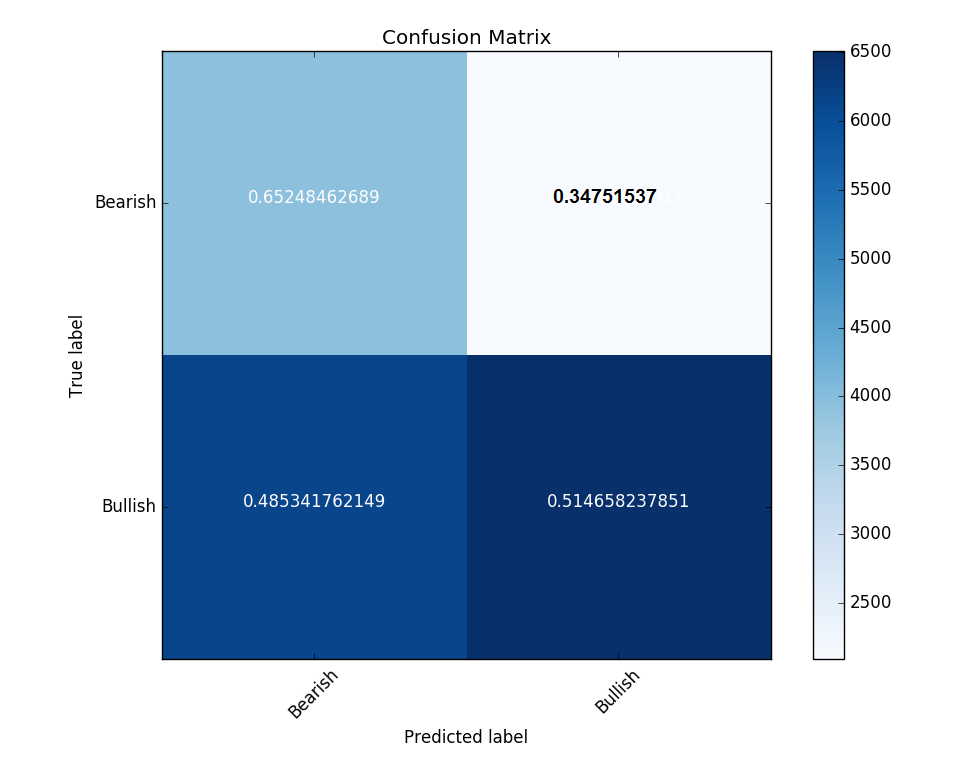
StockTwits provides us with messages and corresponding tags (“Bullish”, “Bearish”, or “None”), which allows us to test model accuracy very easily. We use the dictionary to count the number of positive and negative words, as defined by whichever word dictionary we use. If the number of positive words is greater, then the tweet is predicted to be “Bullish”. If the number of negative words is greater, then the tweet is predicted to be “Bearish”. For the purpose of testing accuracy, we ignore all tweets that are tagged with “None”. This allows for some tweets to be bullish or bearish in sentiment, but for whatever reason the user decided not to tag them.

The code below was run using the command “python dictTesting.py”. The variable in Line 9 was set to “Harvard” or “Financial” to test using the Harvard and Financial dictionaries, respectively.

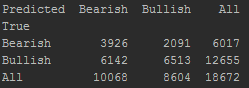
**dictTesting.py Code:**

**import** pandas **as** pd  
**from** sklearn.metrics **import** confusion\_matrix  
**import** matplotlib.pyplot **as** plt  
**import** Preprocess **as** pp  
**import** itertools  
**import** numpy **as** np  
  
# Set dictionary to 'Harvard' or 'Financial'  
DICTIONARY = 'Harvard'  
  
  
**def plot\_confusion\_matrix**(cm, classes,  
 normalize=False,  
 title='Confusion matrix',  
 cmap=plt.cm.Blues):  
 *"""  
 This function prints and plots the confusion matrix.  
 Normalization can be applied by setting `normalize=True`.  
 """* plt.imshow(cm, interpolation='nearest', cmap=cmap)  
 plt.title(title)  
 plt.colorbar()  
 tick\_marks = np.arange(len(classes))  
 plt.xticks(tick\_marks, classes, rotation=45)  
 plt.yticks(tick\_marks, classes)  
 **if** normalize:  
 cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]  
 **print**("Normalized confusion matrix")  
 **else**:  
 **print**('Confusion matrix, without normalization')  
 **print**(cm)  
 thresh = cm.max() / 2.  
 **for** i, j **in** itertools.product(range(cm.shape[0]), range(cm.shape[1])):  
 plt.text(j, i, cm[i, j],  
 horizontalalignment="center",  
 color="white" **if** cm[i, j] > thresh **else** "black")  
 plt.tight\_layout()  
 plt.ylabel('True label')  
 plt.xlabel('Predicted label')  
  
  
# calculate the signal from text in cleaned message  
**def calc\_sentiment**(message):  
 words = message.upper().split()  
 pos\_count = sum([(word **in** fin\_pos) **for** word **in** words])  
 neg\_count = sum([(word **in** fin\_neg) **for** word **in** words])  
 **if** pos\_count > neg\_count:  
 **return** 'Bullish'  
 **elif** neg\_count > pos\_count:  
 **return** 'Bearish'  
 **else**:  
 **return** 'None'  
  
# read in twitter data and sentiment dictionary from files  
data\_paths = ['H:/Course Docs/Big Data/Final Project/Data/StockTwits/AAPL.20170430.191643.csv',  
 'H:/Course Docs/Big Data/Final Project/Data/StockTwits/FB.20170502.024702.csv',  
 'H:/Course Docs/Big Data/Final Project/Data/StockTwits/TSLA.20170501.033001.csv']  
export\_path = 'H:/Course Docs/Big Data/Final Project/Results/Sentiment Analysis-1/test\_dict\_output.csv'  
  
**if** DICTIONARY == 'Financial':  
 # use financial dictionary  
 dict\_path = 'H:/Course Docs/Big Data/Final Project/Docs/LoughranMcDonald\_MasterDictionary\_2014.xlsx'  
 df\_dict = pd.read\_excel(dict\_path)  
 fin\_pos = df\_dict['Word'][df\_dict['Positive'] != 0].tolist()  
 fin\_neg = df\_dict['Word'][df\_dict['Negative'] != 0].tolist()  
  
**elif** DICTIONARY == 'Harvard':  
 # use harvard dictionary  
 dict\_path = 'H:/Course Docs/Big Data/Final Project/Docs/inquirerbasic.xls'  
 df\_dict = pd.read\_excel(dict\_path)  
 fin\_pos = df\_dict[df\_dict['Positiv'] == 'Positiv'].index.tolist()  
 fin\_neg = df\_dict[df\_dict['Negativ'] == 'Negativ'].index.tolist()  
**else**:  
 **print** 'Error: Improper dictionary chosen.'  
  
  
df\_data = pd.DataFrame()  
**for** data\_path **in** data\_paths:  
 **if** len(df\_data) == 0:  
 df\_data = pd.read\_csv(data\_path)  
 **else**:  
 df\_newfile = pd.read\_csv(data\_path)  
 df\_data = pd.concat([df\_data, df\_newfile])  
  
# clean up data  
# remove stop words to reduce dimensionality  
df\_data["stop\_text"] = df\_data["Body"].apply(pp.remove\_stops)  
# remove other non essential words, think of it as my personal stop word list  
df\_data["feat\_text"] = df\_data["stop\_text"].apply(pp.remove\_features)  
# tag the words remaining and keep only Nouns, Verbs and Adjectives  
df\_data["tagged\_text"] = df\_data["feat\_text"].apply(pp.tag\_and\_remove)  
# lemmatization of remaining words to reduce dimensionality & boost measures  
df\_data["text"] = df\_data["tagged\_text"].apply(pp.lemmatize)  
# select only the columns we care about  
df\_data = df\_data[['ID', 'Symbol', 'text', 'Sentiment']]  
  
# calculate sentiment prediction using dictionary  
df\_data['Prediction'] = df\_data['text'].apply(calc\_sentiment)  
  
# summarize tweet counts  
**print** 'Total '+str(len(df\_data))+' tweets'  
**print** 'Actual None: '+str(len(df\_data[df\_data['Sentiment'] == 'None']))+' tweets'  
**print** 'Predict None: '+str(len(df\_data[df\_data['Prediction'] == 'None']))+' tweets'  
**print** 'Either None: '+str(len(df\_data[((df\_data['Sentiment'] == 'None')|(df\_data['Prediction'] == 'None'))]))+' tweets'  
  
# write to file  
act\_scores = df\_data[((df\_data['Sentiment'] != 'None')&(df\_data['Prediction'] != 'None'))]['Sentiment'].tolist()  
dict\_scores = df\_data[((df\_data['Sentiment'] != 'None')&(df\_data['Prediction'] != 'None'))]['Prediction'].tolist()  
messages\_list = df\_data[((df\_data['Sentiment'] != 'None')&(df\_data['Prediction'] != 'None'))]['text'].tolist()  
output = pd.DataFrame({'Predicted': dict\_scores, 'Actual': act\_scores, 'Tweet': messages\_list})  
output[['Predicted', 'Actual', 'Tweet']].to\_csv(export\_path, index=False)  
  
# create data summary table  
table\_totals = pd.crosstab(pd.Series(act\_scores), pd.Series(dict\_scores), rownames=['True'], colnames=['Predicted'], margins=True)  
pd.options.display.float\_format = '{:.2f}'.format  
table\_perc = pd.crosstab(pd.Series(act\_scores), pd.Series(dict\_scores), rownames=['True'], colnames=['Predicted']).apply(**lambda** r: r/r.sum(), axis=1)  
**print** table\_totals  
**print** table\_perc  
  
# Compute and plot confusion matrix  
cnf\_matrix = confusion\_matrix(y\_true=act\_scores, y\_pred=dict\_scores)  
plt.figure()  
plot\_confusion\_matrix(cnf\_matrix, classes=['Bearish', 'Bullish'], title='Confusion Matrix', normalize=True)  
plt.show()

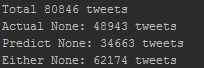
**Harvard Dictionary Output:**

****

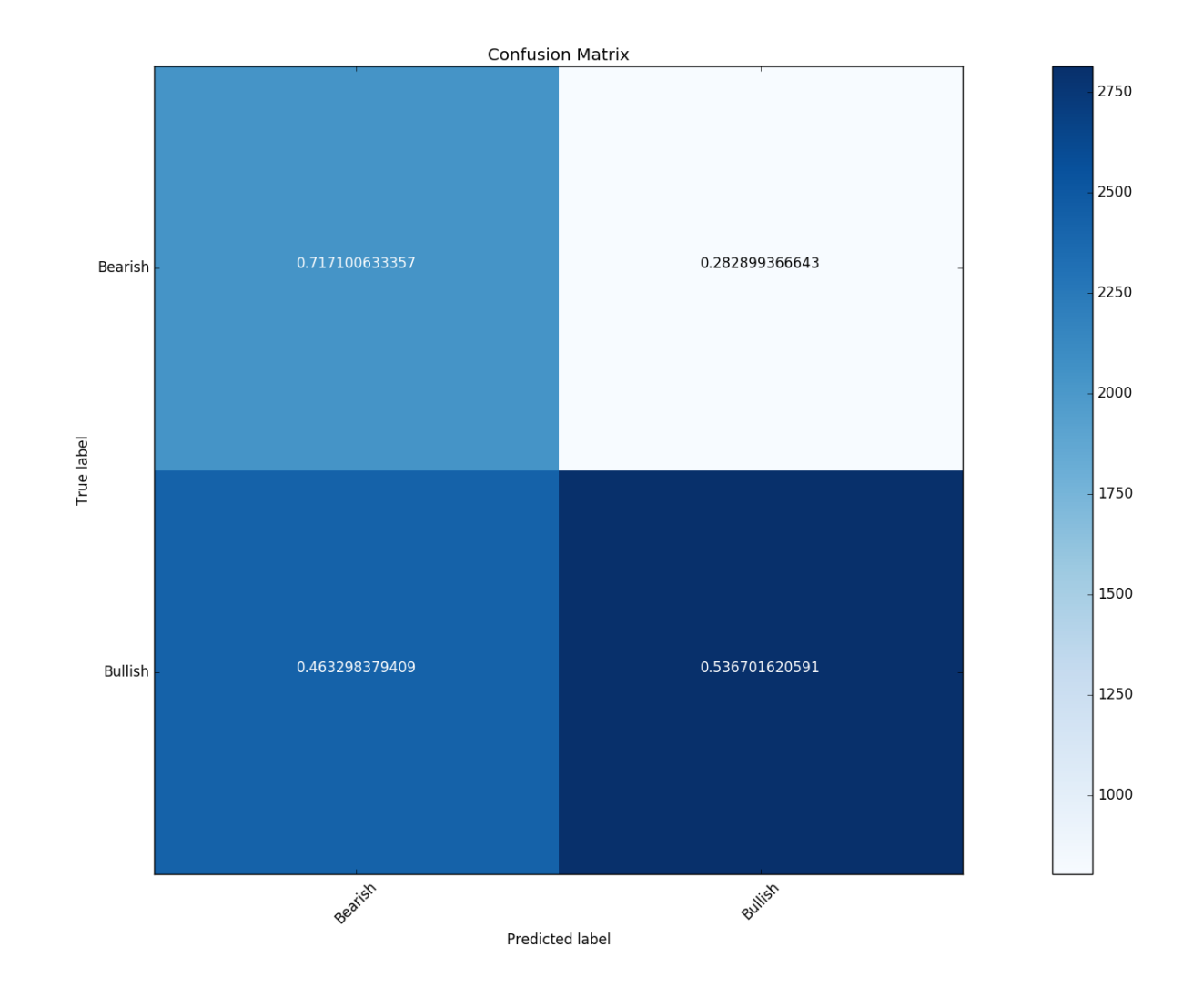
**Count**

****

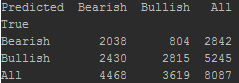
**Excluded**

****

**Financial Dictionary Output:**

****

**Count**

****

**Excluded**

****

**Analysis**

On Bullish tweets, the Harvard dictionary accuracy of 51.5% was not much different from the Financial dictionary’s accuracy of 53.7%. Neither dictionary does much better than a coin flip. On Bearish tweets, however, the Harvard dictionary is able to correctly predict 65.2% of tweets. The Financial dictionary again does better at 71.7%.

The number of (non-neutral) forecasts made is also an important metric to consider. Single tweets tend to be a very condensed set of words and many of the tweets do not have a single word which classifies as either bullish or bearish. Out of all 80,846 tweets considered, only 48,943 of them had user-provided StockTwit tags on them which we could use to test accuracy. Of those, the model using a Financial dictionary made non-neutral forecasts on 8,087 tweets. The model using the general Harvard dictionary made non-neutral forecasts on 18,672 tweets. We are able to produce sentiment scores on twice as many tweets using the general Harvard dictionary (38% vs 17% of all possible). Therefore, while we find that the accuracy of the financial dictionary is better, the general Harvard dictionary is the better option to use in sentiment analysis of StockTwits.

**Reference**

Loughran, T., and McDonald, B. [2011]. “When is a Liability not a Liability? Textual Analysis, Dictionaries, and 10-Ks”, SSRN, <https://papers.ssrn.com/sol3/papers.cfm?abstract_id=1331573>