**tweet download by date and keyword:**

#Variables that contains the user credentials to access Twitter API

access\_token = ""

access\_token\_secret = ""

consumer\_key = ""

consumer\_secret = ""

# # # # TWITTER STREAMER # # # #

class TwitterStreamer():

"""

Class for streaming and processing live tweets.

"""

def \_\_init\_\_(self):

pass

def stream\_tweets(self, fetched\_tweets\_filename, hash\_tag\_list):

# This handles Twitter authetification and the connection to Twitter Streaming API

listener = StdOutListener(fetched\_tweets\_filename)

auth = OAuthHandler(consumer\_key, consumer\_secret)

auth.set\_access\_token(access\_token, access\_token\_secret)

stream = Stream(auth, listener)

# This line filter Twitter Streams to capture data by the keywords:

stream.filter(languages=["en"], track=hash\_tag\_list)

# # # # TWITTER STREAM LISTENER # # # #

class StdOutListener(StreamListener):

"""

This is a basic listener that just prints received tweets to stdout.

"""

count=0

def \_\_init\_\_(self, fetched\_tweets\_filename):

self.fetched\_tweets\_filename = fetched\_tweets\_filename

def on\_data(self, data):

try:

# print(data)

with open(self.fetched\_tweets\_filename, 'a', newline='') as tf:

tweet = json.dumps(data, ensure\_ascii=False)

tf.write(data)

if (self.count%100==0):print(self.count)

self.count=self.count+1

# tweet = json.loads(data)

# with open('your\_data.json', 'a') as my\_file:

# json.dump(tweet, my\_file)

return True

except BaseException as e:

print("Error on\_data %s" % str(e))

return True

def on\_error(self, status):

print("error "+str(status))

if \_\_name\_\_ == '\_\_main\_\_':

# Authenticate using config.py and connect to Twitter Streaming API.

hash\_tag\_list = ['stockmarket', 'bitcoin', 'money', 'trading', 'forextrader', 'investment', 'wallstreet', 'stocks', 'entrepreneur', 'forex', 'trader', 'investor', 'investing', 'cryptocurrency', 'invest', 'business', 'daytrader', 'binaryoptions', 'forexsignals', 'profit', 'success', 'finance', 'wealth', 'makemoneyonline', 'forexlifestyle', 'forextrading', 'motivation', 'millionaire', 'entrepreneurship', 'daytrading']

fetched\_tweets\_filename = "tweets7.json"

twitter\_streamer = TwitterStreamer()

twitter\_streamer.stream\_tweets(fetched\_tweets\_filename, hash\_tag\_list)

def save\_30\_day\_json(tweets,search\_keyword):

a = [json.loads(json.dumps(tweets[i].\_\_dict\_\_, default=json\_util.default)) for i in range(len(tweets))]

with open(search\_keyword+'.json', 'a') as outfile:

json.dump(a, outfile)

print ('saved successfully in ',search\_keyword+'.json','file....')

def get\_tweets(start\_date,search\_keyword,debug=False):

q = start\_date.split('-')

a = datetime(int(q[0]),int(q[1]),int(q[2]))

a+=timedelta(days=1)

end\_date = (a.strftime('%Y-%m-%d'))

tweetCriteria = got.manager.TweetCriteria().setQuerySearch(search\_keyword)\

.setLang('en')\

.setSince(start\_date)\

.setUntil(end\_date)\

.setMaxTweets(3000)

# .setTopTweets(True)\

tweet\_batch = got.manager.TweetManager.getTweets(tweetCriteria)

if debug: print(start\_date,search\_keyword,len(tweet\_batch))

return tweet\_batch

def get\_30days\_tweets(start\_date,search\_keyword,debug=False):

tweets = []

for i in range(30):

tweet\_batch = get\_tweets(start\_date,search\_keyword,debug)

tweets.extend(tweet\_batch)

q = start\_date.split('-')

a = datetime(int(q[0]),int(q[1]),int(q[2]))

a+=timedelta(days=1)

start\_date = (a.strftime('%Y-%m-%d'))

open('start\_date','w').write(start\_date)

save\_30\_day\_json(tweet\_batch,search\_keyword)

# time.sleep(30)

# if debug: print (len(tweets),start\_date,search\_keyword)

return tweets

open('start\_date','w').write('2017-10-01')

open('company\_list','w').write("AMZN")

while(1):

temp = open('company\_list','r').read()

keyword\_list=list(temp.split('-'))

loop\_keyword\_list = list(keyword\_list)

start\_date =open('start\_date','r').read()

try:

for search\_keyword in loop\_keyword\_list:

print ('Running for ', search\_keyword.upper())

tweets = get\_30days\_tweets(start\_date,search\_keyword,debug=True)

print ('Tweets downloaded..')

keyword\_list.remove(search\_keyword)

open('company\_list','w').write('-'.join(keyword\_list))

except Exception as e:

print (e)

open('start\_date','w').write(start\_date)

open('company\_list','w').write('-'.join(keyword\_list))

print ('error occoured on: ')

print (start\_date,search\_keyword)

time.sleep(30)#sleeping for too many request error

print ('continuing')

continue

temp = open('company\_list','r').read()

keyword\_list=list(temp.split('-'))

loop\_keyword\_list = list(keyword\_list)

start\_date =open('start\_date','r').read()

print (start\_date,loop\_keyword\_list)

with open('data.json','r') as out:

red = json.load(out)

print ((red[0].keys()))

# print ((red[0]['formatted\_date']))

# p= datetime.datetime.fromtimestamp(1577918767)

# print (p.strftime('%B/%Y/%d/ %H-%M-%S'))

for i in red:

if i['id'] == red[0]['id']:

print (i['text'])

**Model for stock predicton:**

data\_path = '/content/drive/My Drive/Colab Notebooks/tweets\_data\_analysis/'

with open(data\_path+'tokenizer.pickle', 'rb') as handle:

tokenizer = pickle.load(handle)

#now directly use tokenizer.texts\_to\_matrix(['your input string'])

senti\_model = load\_model(data\_path+'senti\_model.tf')

xy = pd.read\_csv(data\_path+'aapl.csv')

print (xy)

print (xy.columns)

# this function removes all punctuations and non text things

def data\_cleansing(list):

return\_output = []

for string in list:

letters\_only = re.sub("[^a-zA-Z]", " ", string)

words = letters\_only.lower().split()

output = re.sub(r"\b[a-zA-Z]\b", "", " ".join(words))

# output = ' '.join(words)

return\_output.append(output)

return (return\_output)

def pre\_process\_data(df):

# grouping date by date.. and aggregating date like username , text in list for single date

df = df.groupby(by='date').agg({'username':list,

'text':list,

'favorites':list,

'replies':list,

'retweets':list,

'author\_id':list,

'urls':list,

'mentions':list,

'hashtags':list,

'diff\_percent':'first',

'output\_binary':'first',

'timestamp':'first',

'max\_diff':'first',

'diff':'first',

'date':'first'})

df['text'] = df['text'].apply(lambda x:data\_cleansing(x))

# 0 for negative sentence and 1 for positive sentence

# predicting mood of sentence using previosly trainned model

df['text\_mood'] = [senti\_model.predict\_classes(tokenizer.texts\_to\_matrix(i)) for i in df['text']]

df['positive\_count'] = [np.count\_nonzero(i==1) for i in df["text\_mood"]]

df['negative\_count'] = [np.count\_nonzero(i==0) for i in df["text\_mood"]]

df['total\_count'] = df['positive\_count']+df['negative\_count']

# merging log list into single sum value.

df['favorites'] = [sum(i) for i in df['favorites']]

df['retweets'] = [sum(i) for i in df['retweets']]

df['replies'] = [sum(i) for i in df['replies']]

return df

#reading all stocks file together in single data frame

import pandas as pd

import glob

path = data\_path # use your path

all\_files = ['aapl.csv', 'amzn.csv', 'tsla.csv', 'googl.csv']

li = []

for filename in all\_files:

df = pd.read\_csv(data\_path+filename, index\_col=None, header=0)

df = pre\_process\_data(df)

# df.to\_csv(data\_path+'ISL\_'+filename, index = False)

li.append(df)

raw = pd.concat(li, axis=0, ignore\_index=True)

#reading single stock file

#reading Amazon stock

raw = pd.read\_csv(data\_path+'amzn.csv')

data = pre\_process\_data(raw)

# data = raw

x = (data[['favorites','replies','retweets','positive\_count','negative\_count','total\_count']])

y = data['diff']

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.20, random\_state=1000)

print (x\_train.shape)

#regression

from sklearn import linear\_model

from sklearn.metrics import mean\_squared\_error, r2\_score, accuracy\_score

regr = linear\_model.LinearRegression()

X\_train, X\_test, Y\_train, y\_test= train\_test\_split(x, y, test\_size=0.2, random\_state=0)

regr.fit(X\_train, Y\_train)

y\_pred=regr.predict(X\_test)

print("Variance score: %.2f" % r2\_score(y\_test,y\_pred)) # varience

print("Mean squared error: %.2f" % mean\_squared\_error(y\_test,y\_pred)) #squaring errors for removing negative values

print (accuracy\_score(y\_test,y\_pred))

x=len(y\_pred)

import matplotlib.pyplot as plt

m=[]

for i in range(x):

m.append(i)

plt.scatter(m[:20],y\_pred[:20],color="red")

plt.scatter(m[:20],y\_test[:20],color="green")

plt.plot(m[:20],y\_test[:20],color="green")

plt.show()

#plotting all data frame with date on x axis

import matplotlib.pyplot as plt

x = data['date']

y = data['diff\_percent']

z = data['total\_count']

a = data['positive\_count']

b = data['negative\_count']

c = data['favorites']

d = data['retweets']

plt.rcParams["figure.figsize"] = (15,15)

fig, (ax1, ax2, ax3, ax4, ax5,ax6) = plt.subplots(6)

ax1.plot(x, y)

ax1.text(.5,.9,'Stock change in %', horizontalalignment='center', transform=ax1.transAxes)

ax2.plot(x, z)

ax2.text(.5,.9,'total tweets', horizontalalignment='center', transform=ax2.transAxes)

ax3.plot(x, a)

ax3.text(.5,.9,'positive\_count', horizontalalignment='center', transform=ax3.transAxes)

ax4.plot(x, b)

ax4.text(.5,.9,'negative count', horizontalalignment='center', transform=ax4.transAxes)

ax5.plot(x, c)

ax5.text(.5,.9,'favorites', horizontalalignment='center', transform=ax5.transAxes)

ax6.plot(x,d)

ax6.text(.5,.9,'retweets', horizontalalignment='center', transform=ax6.transAxes)

model = Sequential()

model.add(layers.Dense(2000,input\_dim=6, activation='relu'))

model.add(layers.Dense(3000, activation='relu'))

model.add(layers.Dense(3000, activation='relu'))

model.add(layers.Dense(300, activation='relu'))

model.add(layers.Dense(300, activation='relu'))

model.add(layers.Dense(2, activation='sigmoid'))

model.compile(loss='sparse\_categorical\_crossentropy',optimizer='adam',metrics=['acc']) #optimizers adam, rmsprop, sgd

history=model.fit(x\_train,y\_train, epochs=2, verbose=1, validation\_data=(x\_test,y\_test), batch\_size=256)

**tweet\_sentimental\_model\_training:**

col\_names = ['label','id','date','waste','username','text']

data = pd.read\_csv(data\_path, encoding='latin-1',names=col\_names)

manual = pd.read\_csv('/content/drive/My Drive/Colab Notebooks/tweets\_data\_analysis/testdata.manual.2009.06.14.csv', encoding='latin-1')

def data\_cleansing(string):

letters\_only = re.sub("[^a-zA-Z]", " ", string)

words = letters\_only.lower().split()

output = re.sub(r"\b[a-zA-Z]\b", "", " ".join(words))

return (output)

data['text'] = data['text'].apply(lambda x:data\_cleansing(x))

x = data['text'].values

y = data['label'].values

tokenizer = Tokenizer(num\_words=1000)

tokenizer.fit\_on\_texts(x)

print ("tokenizer is ready")

print ("done")

tokenizer\_json = tokenizer.word\_index

with open('/content/drive/My Drive/Colab Notebooks/tweets\_data\_analysis/token\_wordindex.json', 'w') as f:

f.write(json.dumps(tokenizer\_json))

with open('tokenizer.json') as f:

data = json.load(f)

tokenizer = tokenizer\_from\_json(data)

# saving

with open('/content/drive/My Drive/Colab Notebooks/tweets\_data\_analysis/tokenizer.pickle', 'wb') as handle:

pickle.dump(tokenizer, handle, protocol=pickle.HIGHEST\_PROTOCOL)

# loading

with open('tokenizer.pickle', 'rb') as handle:

tokenizer = pickle.load(handle)

x = tokenizer.texts\_to\_matrix(x)

le = preprocessing.LabelEncoder()

y = le.fit\_transform(y)

print (len(y))

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.25, random\_state=1000)

model = Sequential()

model.add(layers.Dense(300,input\_dim=1000, activation='relu'))

model.add(layers.Dense(2, activation='softmax'))

model.compile(loss='sparse\_categorical\_crossentropy',optimizer='adam',metrics=['acc']) #optimizers adam, rmsprop, sgd

history=model.fit(x\_train,y\_train, epochs=2, verbose=1, validation\_data=(x\_test,y\_test), batch\_size=256)

model.save('/content/drive/My Drive/Colab Notebooks/tweets\_data\_analysis/senti\_model.tf')

inp = "this market is going is worst to eat me alive"

print ("token is ready")

inp = tokenizer.texts\_to\_matrix([inp])

print (inp.shape)

print (inp)

z = model.predict\_classes(inp)

print (z)

model = load\_model('/content/drive/My Drive/Colab Notebooks/tweets\_data\_analysis/senti\_model.tf')

model.save('keras.h5')

!mkdir model

!tensorflowjs\_converter --input\_format keras keras.h5 model/

!zip -r model.zip model

**Sentiment analysis:**

data\_path = '/content/drive/My Drive/Colab Notebooks/tweets\_data\_analysis/training.1600000.processed.noemoticon.csv'

col\_names = ['label','id','date','waste','username','text']

dataframe = pd.read\_csv(data\_path, encoding='latin-1',names=col\_names)

len(dataframe)

dataframe.columns

dataframe.head()

((dataframe [~(dataframe ['text'].str.contains('[A-Za-z]'))].count()[0])/ dataframe.count()[0])\*100

dataframe.head()

dataframe.columns,len(dataframe.columns)

len(dataframe)

dataframe.rename(columns={'text':'tweet\_txt'},inplace=True)

dataframe.head()

dataframe.dtypes

def data-cleansing(corpus):

letters = re.sub("[^a-zA-Z]", " ", corpus)

words = letters.lower().split()

return( " ".join( words ))

df['tweet-txt'] = df['tweet-txt'].apply(lambda x:data-cleansing(x))

df['sentit-value']= dataframe.tweet\_txt.apply(lambda x:TextBlob(str((x).encode('ascii', 'ignore'))).senti.polarity)

TextBlob(str(('hello hate').encode('ascii', 'ignore'))).sentiment.polarity

dataframe.dtypes

dataframe ['senti\_score']=np.where(dataframe.sentiment\_value<=0.0,1,0)

dataframe ['senti\_description']=np.where(dataframe.senti\_value<=0.0,'negative','positive')

dataframe.head()

print(dataframe.senti\_score.value\_counts(),'\n\n',df.senti\_description.value\_counts())

from wordcloud import WordCloud, STOPWORDS

stopword = set(STOPWORDS)

def wordcloud(source,stop):

tmp = dataframe [dataframe ['senti\_description']==source]

clean-text=[]

for each in tmp['tweet-txt']:

clean-text.append(each)

clean-text = ' '.join(clean-text)

if source == 'positive' :

color=’black’

else:

color=’white’

if (stop=="yes"):

wordcloud = WordCloud(background\_color=color,

width=3000,

height=3500,stopword = stopword

).generate(clean-text)

else:

wordcloud = WordCloud(background\_color=color,

width=3000,

height=3500

).generate(clean-text)

print('==='\*35)

print('word cloud of '+source+' is plotted below')

plt.figure(1,figsize=(8,8))

plt.imshow(wordcloud,interpolation='bilinear')

plt.show()

stopword.add('co')

stopword.add('https')

stopword.add('hey')

stopword.add('hello')

stopword.add('school')

wordcloud('positive',"yes")

wordcloud('negative',"yes")

vector = TfidfVectorizer(stop\_words='english',strip\_accents='unicode', token\_pattern=r'\w{2,}')

train-features = vector.fit\_transform(train-corpus)

test-features=vector.transform(test-corpus)

print(train-features.shape)

print(test-features.shape)

vector.get\_feature\_names()

def ML\_Pipeline(clf\_name):

clasification = Classifiers[clasification \_name]

fit = clasification.fit(train\_features,train['sentiment\_description'])

predict = clasification.predict(test\_features)

Accuracy = accuracy\_score(test['sentiment\_description'],predict)

Confusion\_matrix = confusion\_matrix(test['sentiment\_description'], predict)

print('==='\*30)

print('Accuracy of '+ clasification \_name +' is '+str(Accuracy))

print('==='\*30)

print(Confusion\_matrix)

ML\_Pipeline('lg')

clasification = RandomForestClassifier(random\_state=40,n\_estimators=200,n\_jobs=-1)

fit = clasification.fit(train-features,train['senti-description'])

words = vector.get\_feature\_names()

importance = clasification.feature\_importances\_

impordataframe = pd.DataFrame({'Word' : words,'Importance' : importance})

impordataframe = impordf.sort\_values(['Importance', 'Word'], ascending=[0, 1])

impordataframe.head(10)

impordataframe.loc[impordataframe['Importance']<=0.0]

clasification = LogisticRegression(random\_state=40,C=5,max\_iter=200)

fit = clasification.fit(train\_features,train['senti\_description'])

predict = clasification.predict(test\_features)

Accuracy = accuracy\_score(test['senti\_description'],predict)

Confusion\_matrix = confusion\_matrix(test['sentit\_description'],predict)

print('==='\*30)

print('Accuracy of '+ 'lr' +' is '+str(Accuracy))

print('==='\*30)

print(Confusion\_matrix)

DeepLearning :

df\_filtered['senti-description']=np.where(df\_filtered.score=='"0"',0,1)

train-corpus = []

test-corpus = []

model-corpus=[]

for each in train['tweet']:

    train-corpus.append(each)

for each in test['tweet']:

    test-corpus.append(each)

for each in dataframe\_filtered['tweet']:

    alltext = ''.join([c for c in each if c not in punctuation])

    model-corpus.append(alltext)

alltext2 = ' '.join(reviews-split)

words = all-text2.split()

Count-words = Counter(words)

totalwords = len(words)

sortwords = countwords.most\_common(totalwords)

reviewslen = [len(x) for x in reviewsint]

pd.Series(reviewslen).hist()

pd.Series(reviewslen).describe()

reviewsint = [ reviewsint[i] for j, l in enumerate(reviewslen) if l>0 ]

encodedlabels = [ encodedlabels[i] for j, l in enumerate(reviewslen) if l> 0 ]

features = np.zeros((len(reviewsint), seqlength))

    for j, review in enumerate(reviewsint):

        reviewlen = len(review)

        if reviewlen <= seqlength:

            zeroes = list(np.zeros(seqlength-reviewlen))

            new = zeroes+review

        elif reviewlen > seqlength:

            new = review[0:seqlength]

        features[i,:] = np.array(new)

    return features

splitfrac = 0.8

lenfeat=len(features)

train\_x = features[0:int(splitfrac\*lenfeat)]

train\_y = encodedlabels[0:int(splitfrac\*lenfeat)]

remain\_x = features[int(splitfrac\*lenfeat):]

remain\_y = encodedlabels[int(splitfrac\*lenfeat):]

valid\_x = remain\_x[0:int(len(remain\_x)\*0.5)]

valid\_y = remain\_y[0:int(len(remain\_y)\*0.5)]

test\_x = remain\_x[int(len(remain\_x)\*0.5):]

test\_y = remain\_y[int(len(remain\_y)\*0.5):]

class SentimentLSTM(nn.Module):

def \_\_init\_\_(self, vocabsize, outputsize, embeddingdim, hiddendim, nlayers, dropprob=0.5):

super().\_\_init\_\_()

        self.outputsize = outputsize

        self.nlayers = nlayers

        self.hiddendim = hiddendim

        self.embedding = nn.Embedding(vocabsize, embeddingdim)

        self.lstm = nn.LSTM(embeddingdim, hiddendim, nlayers,

                            dropout=dropprob)

        self.dropoutlayer = nn.Dropout(0.2)

        self.fc = nn.Linear(hiddendim, outputsize)

        self.sig = nn.Sigmoid()

    def fwd(self, y, hidden):

        batchsize = y.size(0)

        embed = self.embedding(y)

        lstmout, hidden = self.lstm(embed, hidden)

        lstmout = lstmout.contiguous().view(-1, self.hidden\_dim)

        dout = self.dropout(lstmout)

        dout = self.fc(dout)

        sigout = self.sig(dout)

        sigout = sigout.view(batchsize, -1)

        sigout = sigout[:, -1]

        return sigout, hidden

    def inithidden(self, batch\_size):

        wg = next(self.parameters()).data

        if (train\_on\_gpu):

            hidden = (wg.new(self.nlayers, batchsize, self.hiddendim).zero\_().cuda(),

                  wg.new(self.nlayers, batchsize, self.hiddendim).zero\_().cuda())

        else:

            hd = (wg.new(self.nlayers, batchsize, self.hiddendim).zero\_(),

                      wg.new(self.nlayers, batchsize, self.hiddendim).zero\_())

        return hd