Natural Language Processing

Stock Market Prediction based on Market Sentiment

Table of Contents

General Information

Definition of Terms

- API: The application program interface (API) defines how software components should interact.
- Sentiment analysis: this is the process of opinion mining to gauge positive, neutral or negative 'feeling' that the market holds, at given point time, relating to a stock performance.

Software Dependencies

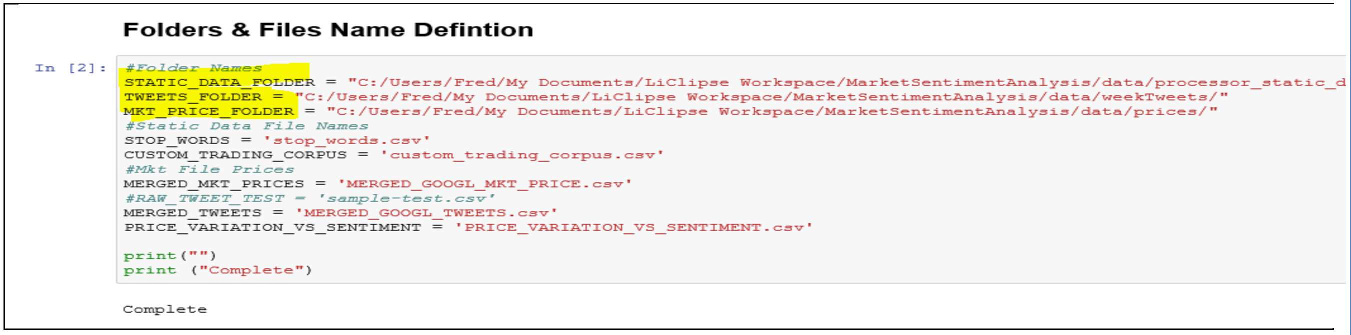
- Software - LiClipse, Python 3.5.2, jupyter notebook.
- Specialised Libraries - csv, NLTK, re, pprint, datetime, random, os, sys, ateutil, matplotlib, pandas, sklearn. NLTK.corpus, urllib , json, pickle, oauth2,uuid.

Hardware

Windows 10 64bits platform, supported by an Intel Core i7-67000HQ processor / RAM: DDR4 8GB.

How to Run the Code

- Market Data and Sentiment Download:
  - Unzip the folder *liclipse_files.*
  - *I*mport the files present in the *liclipse_files* folder into LiClipse.
  - Start the *scheduler.py*. It triggers the download of the price and Twitter's messages on an hourly basis. Please note, prices are only downloaded once at any time between 21.30 and 23.00 (GMT time)
- The Market and Sentiment pre-processor, feature vector creation, sentiment generation, training/testing labels creation, sentiment predictive power prediction require the following steps:
  - Unzip the *data* folder
  - In the section *Folders & Files Name Definition* change the three highlighted variables to the location of the unzipped respective directories: *…/data/processor_static_data, …/data/prices* and *…/data/weekTweets*



Folders & Files Name Defintion

```
In [2]: #Folder Names
        STATIC_DATA_FOLDER = "C:/Users/Fred/My Documents/LiClipse Workspace/MarketSentimentAnalysis/data/processor_static_d
        TWEETS_FOLDER = "C:/Users/Fred/My Documents/LiClipse Workspace/MarketSentimentAnalysis/data/weekTweets/"
        MKT_PRICE_FOLDER = "C:/Users/Fred/My Documents/LiClipse Workspace/MarketSentimentAnalysis/data/prices/"
        #Static Data File Names
        STOP_WORDS = 'stop_words.csv'
        CUSTOM_TRADING_CORPUS = 'custom_trading_corpus.csv'
        #Mkt File Prices
        MERGED_MKT_PRICES = 'MERGED_GOOGL_MKT_PRICE.csv'
        #RAW_TWEET_TEST = 'sample-test.csv'
        MERGED_TWEETS = 'MERGED_GOOGL_TWEETS.csv'
        PRICE_VARIATION_VS_SENTIMENT = 'PRICE_VARIATION_VS_SENTIMENT.csv'

        print("")
        print ("Complete")

        Complete
```

  - Start *sentiment_analysis.ipynb* from a command line as follows:
    >> jupyter notebook sentiment_analysis.ipynb

Input Files

tweet_GOOGL_today_tweets_[date]-[time]_today.csv	Template all the tweet message files downloaded on an hourly basis (during trading hours). For example: tweet_GOOGL_today_tweets_2017-01-31-21.43.05_today.csv
GOOGL_[Year_Month_Day].csv	Template all the market price message files downloaded at close of business. For example: GOOGL_2017_01_04_.csv

Output Files

MERGED_GOOGL_TWEETS.csv	Contains all the cleaned-up & date merged tweets message
MERGED_GOOGL_MKT_PRICE.csv	Contains all the cleaned-up & date merged market prices
DEBUG_SENTIMENT.txt	List each bag of words feature vector and record the sentiment for each feature, as well as the overall sentiment for each vector.
PRICE_VARIATION_VS_SENTIMENT.csv	Base dataset used by the machine learning algorithms. List the market price information (5 min tick), the average sentiment data (5min tick) and other derived information.

Abstract

An experiment consisting of downloading Twitter's messages, relating to the GOOGL stock, in parallel to its 5min ticking market price data over a 20 days' period, showed that sentiments do not seem to have significant prediction power on the stock intra-day trend prediction. This result is disappointing and seems to contradict conclusions reached in the literature. A number of limitations, included but not limited to sparse data, could be at the origin of this result.

Introduction

The ability to predict stock market price trends ahead of time has always been the investors' grail. The main two axes employed to attempt to discover market price trends have historically been: on one hand, *fundamental economic analysis*. This is the analysis of companies accounting balances as well as macro- and micro-economics analysis. On the other hands, *technical analysis* [18]; the use of statistical measures to predict future price moves based on past prices, volatilities and volumes. Although these techniques usually compete with themselves, there are quite complementary. Usually investors use them in parallel to forge an investment decision. However, there is one aspect that cannot directly be captured with these techniques: the prediction of trend changes due to "the herd behaviour" [15]. This is particularly present in intra-day trading (i.e. short term trading), where a large volume of agents decides to buy/sell stocks based on some news viewed as being favourable or catastrophic. With the development of near real-time financial news, broadcasted by news agency (e.g. Reuters), and the emergence of online news and social networking service (e.g. Twitter), it has now become easier to capture the sentiment growing in the market, following an event. It is thought that if sentiments can be captured early enough, investors can react faster and anticipate the potential sharp change in trend [5].
The purpose of this report is to build a prototype Python application that parses tweeter messages relating to a selected stock price (i.e. GOOGL). Then perform a trend prediction using price and volume information attributes for two famous machine learning models, namely the Support Vector Machine (SVM) and the Naïve Bayes models. Finally, add the sentiment attribute to the models and check whether the trend prediction accuracy is improved. GOOGL is selected as it is one of the most influential and liquid stock of the S&P500, i.e. stock with a high market capitalisation and with high daily exchange volume [3].

A Short Literature Review

In recent years, researchers have focused their attention on stock market volatility and trend prediction using sentiment analysis and machine learning. The first step in the process is to produce a sentiment time series. Research papers focus on either i) gathering sentiment data from financial information platforms only (such as Reuters, Bloomberg, the Financial Times, etc.) [9], social media platforms only (e.g. Twitter) [6][12], or both [11]. They then choose to use either a unigram, a n-gram approach [1] or a specialised corpus [2] to extract sentiment from sentences. Then for the prediction phase different methodologies are deployed. The most simplistic approach is to create an autoregressive (AR) model that regresses the stock market returns against lagged variables (such as technical indicators). Then machine learning algorithms are selected such as the Neural Network (NN) [6], the Support Vector Machine (SVM) [5], the Logistic Regression [5] models. The performance accuracy is used as a benchmark. On completion, the sentiment time series is added to the equation. The same models are run and the new accuracy is produced. There is a consensus over all these studies that sentiment analysis has significant impact in terms of improving prediction accuracy. More robust and complete statistical approaches were also applied recently [14] to establish the statistical significance and stability of the accuracy results. The authors proved using a linear Granger-causal framework and a Monte Carlo simulation, that negative sentiment seemed to play a role in the down trend of the S&P500. Positive sentiment did not seem to have any significant influence.

Data Description

To conduct this experiment, the first step is to gather the necessary data; i.e. the GOOGL Twitter's messages and market prices. The remainder of this section details the approach.
The data gathering process took place over a period of 20 working days. This is necessary to provide a large enough dataset to perform statistical inferences. The data collected is of two kinds:

i) The stock market prices for GOOGL. It is collected via the Yahoo!Finance API. It ticks every 5 minutes from the opening time at 14.30 to 21.00 GMT time. The timestamp is automatically converted to GMT in the Yahoo!Finance *API*. This provides approximately 80 prices a day (hence 1,600 prices over a 20 days' period).
Table 1 below describes the schema of the GOOGL stock price file downloaded from Yahoo!Finance. Table 2 provides a few records.

Column Name	**Description**
Date	The stock price date time (GMT)
Open	The Opening Level
High	The highest price in a 5 minutes' time period
Low	The lowest price in a 5 minutes' time period
Close	The closing price in a 5 minutes' time period

Table 1 – The stock price file schema

```
Date,Open,High,Low,Close,Volume
1477661640,818.6850,830.0000,818.0000,829.7875,359900
1477661940,822.7900,822.8200,817.0000,818.8100,171100
1477662241,826.5500,826.7300,822.6800,822.7800,170400
…
```

Table 2 – Output examples for the GOOGL stock price

ii) The sentiment data is collected via the Twitter API. Tables 3/4 respectively define the output schema and provide two message instances. The time is automatically converted to GMT in the Twitter's API.

Column Name	**Description**
created_at	The message creation date time (GMT)
Text	The message written by the user

Table 3 – The stock price file schema returned by a Yahoo!Finance call

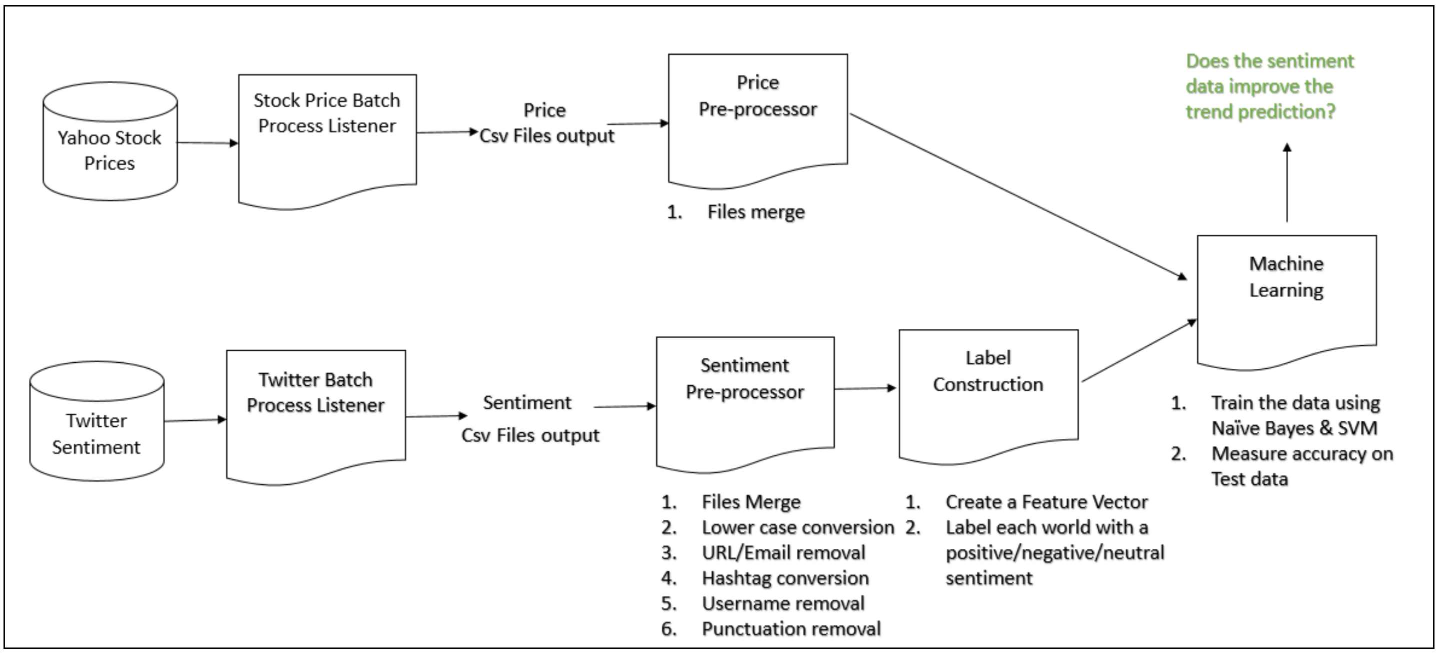
```
created_at,text
b'Wed Jan 04 06:27:11 +0000 2017',"b'Googl Alphabet Inc. Stock Price, Detailed Quote, And   :
https://t.co/XQgRwPoZ6J ., https://t.co/hN4gKEtOR1'"
"b'Googl Alphabet Inc. Stock Price, Detailed Quote, And   : https://t.co/XQgRwPoZ6J .,
https://t.co/hN4gKEtOR1'",b'Wed Jan 04 06:16:19 +0000 2017'

b'Wed Jan 04 06:16:19 +0000 2017',b'timesofindia : We have launched training for all the budding entrepreneurs of
India: Googl\\u2026 https://t.co/Vn7NTkBZ0C) https://t.co/7mR7FjYwIN'
b'timesofindia : We have launched training for all the budding entrepreneurs of India: Googl\\u2026
https://t.co/Vn7NTkBZ0C) https://t.co/7mR7FjYwIN',b'Wed Jan 04 06:10:25 +0000 2017'
…
```

Table 4 – Output examples for the GOOGL stock price file (Twitter API call)

It is interesting to note that the stock price dates are provided as a *long* not a *datetime* object. Furthermore, Twitter's messages are duplicated (highlighted in grey in Table 4). Furthermore, the *created_text* and *text* are flipped in the duplicated version. These issues are discussed in the next section. Due to the quantity of data being produced, it would not have been practical to store it in the Appendix. All the datasets generated by the different API and the one fabricated for the need of the application are available in the provided zip file named: MktSenAnalysis.zip, under the *data* folder. The code for connecting and gathering data from the different APIs is available in Appendix A.

Methodology

The methodology is inspired by other work in this area [1], [2] and [8], and is summarised in Figure 1 below.



Yahoo Stock Prices
Stock Price Batch Process Listener
Price Csv Files output
Price Pre-processor
1. Files merge
Does the sentiment data improve the trend prediction?
Machine Learning
1. Train the data using Naïve Bayes & SVM
2. Measure accuracy on Test data
Twitter Sentiment
Twitter Batch Process Listener
Sentiment Csv Files output
Sentiment Pre-processor
1. Files Merge
2. Lower case conversion
3. URL/Email removal
4. Hashtag conversion
5. Username removal
6. Punctuation removal
Label Construction
1. Create a Feature Vector
2. Label each world with a positive/negative/neutral sentiment

Figure 1: The stock price and tweet messages extraction, transformation and analysis.

Market data clean-up

The first step involves data clean-up for each data source type. The stock market date records are converted from *long* to a *datetime* format (to enhance readability). Daily files are merged into one unique file. As the *datetime* values are in GMT, no further conversion is required. All records are stored in data/time ascending order. The code for this transformation can be found in the Appendix B. Table 5 shows a few records of the merged file (c.f. ./data/prices/*MERGED_GOOGL_MKT_PRICE.csv*).

```
Date,Open,High,Low,Close,Volume
2016-12-30 14:34:02,802.1150,802.8500,801.2200,801.3100,50200
2016-12-30 14:39:02,800.4400,802.3150,800.4400,801.6700,10300
2016-12-30 14:44:02,800.3100,801.3092,800.0900,800.6700,9100
2016-12-30 14:49:04,800.0600,800.9400,799.8300,800.1800,11100
2016-12-30 14:54:02,800.0400,800.6100,799.4400,800.2600,13400
...
```

Table 5 – Output examples for the merge GOOGL stock price file (MERGED_GOOGL_MKT_PRICE.csv)

Feature vector creation

The sentiment data is cropped from the Twitter API on an hourly basis between the trading hours' start and end times. This is necessary as only the last 100 records are provided by each Twitter's API call. Daily files are combined into a single file. The code, available in the Appendix C, ensures that each tweet is unique (i.e. no duplicates). All records are stored in date/time ascending order. Table 6 shows a few records of the merged file (c.f. ./data/weekTweets/*MERGED_GOOGL_TWEETS.csv*).

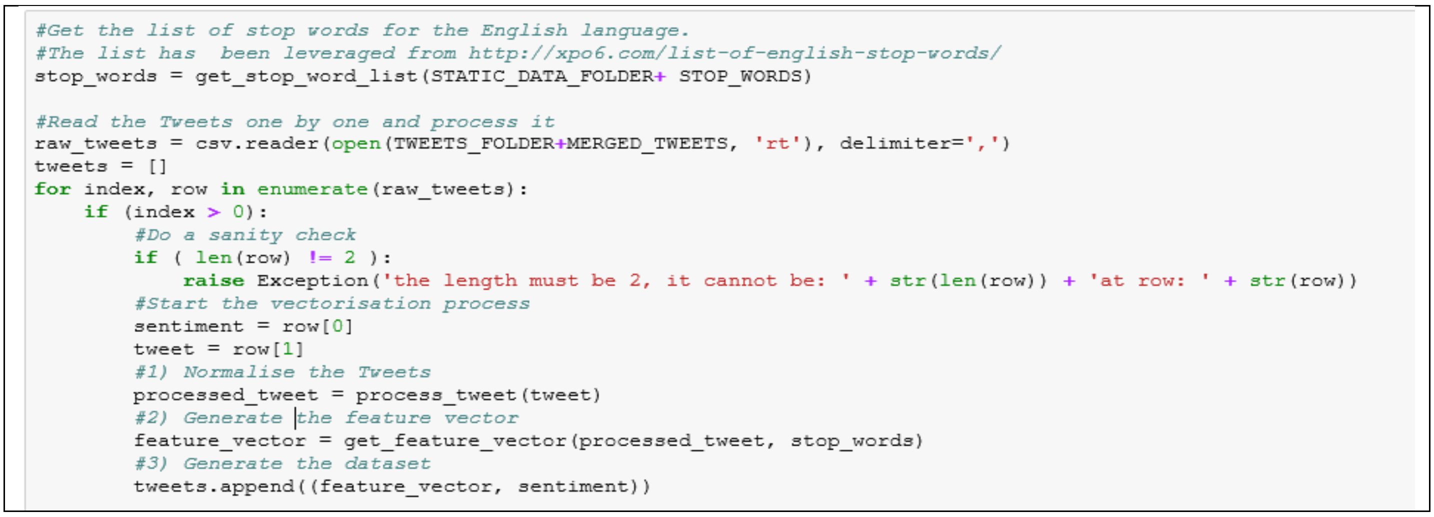
```
created_at,text
...
2017-01-04 14:35:27,b'I went with $TSLA in this group but would like to have seen $AMZN or
$GOOGL rather than $NFLX. https://t.co/tuvFsK4FLd'
2017-01-04 14:36:05,"b'Googl trying to clear, some gap ahead'"
2017-01-04 14:37:38,"b'RT @Setting4Success: #Setting4Success The fascinating life of Google
founder and Alphabet CEO Larry Page (GOOG, GOOGL) #News #smallbusiness\\u2026'"
2017-01-04 14:37:42,"b'Nice to play options on $AAPL, $FB, $GOOGL, $AMZN, $TSLA'"
 ...
```

Table 6 – Output examples for the merged tweet file (MERGED_GOOGL_TWEETS.csv)

In order to tag each tweet with an overall sentiment (*Positive*, *Negative* or *Neutral*), each tweet is processed as follows:

i) Normalisation step: it converts the text in a more standard form
ii) Feature vectors generation step: messages are broken down in a list of words
iii) Overall sentiment generation (c.f. *Code Snippet 1*).

The following section details these steps in detail. The full code relating to these transformations is available in Appendix D. The most relevant part of the logic is discussed below.



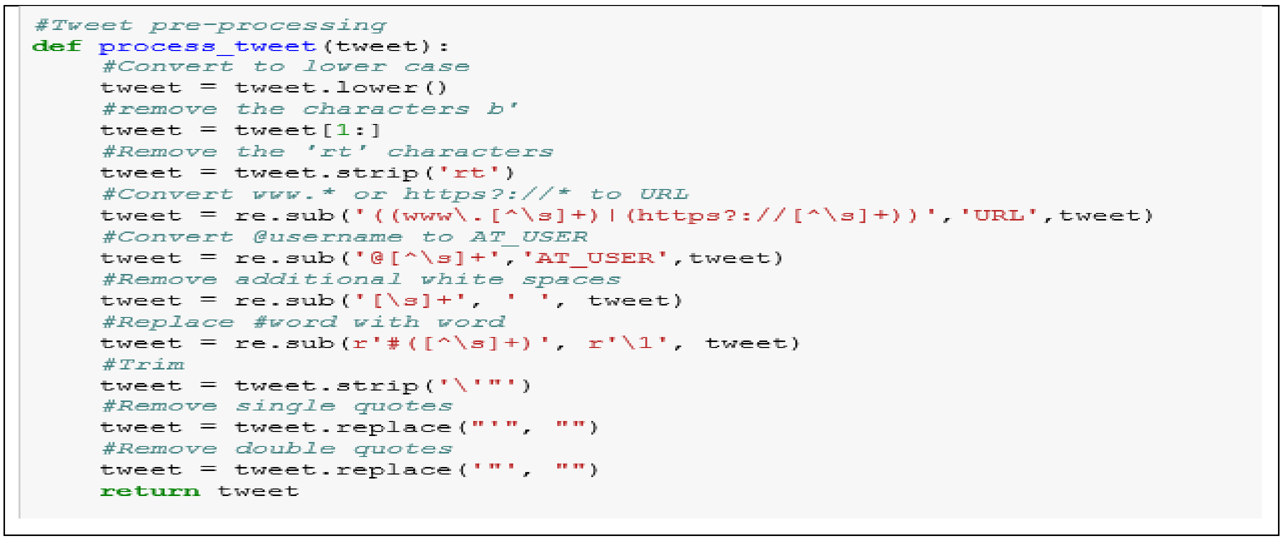
```
#Get the list of stop words for the English language.
#The list has  been leveraged from http://xpo6.com/list-of-english-stop-words/
stop_words = get_stop_word_list(STATIC_DATA_FOLDER+ STOP_WORDS)

#Read the Tweets one by one and process it
raw_tweets = csv.reader(open(TWEETS_FOLDER+MERGED_TWEETS, 'rt'), delimiter=',')
tweets = []
for index, row in enumerate(raw_tweets):
    if (index > 0):
        #Do a sanity check
        if ( len(row) != 2 ):
            raise Exception('the length must be 2, it cannot be: ' + str(len(row)) + 'at row: ' + str(row))
        #Start the vectorisation process
        sentiment = row[0]
        tweet = row[1]
        #1) Normalise the Tweets
        processed_tweet = process_tweet(tweet)
        #2) Generate the feature vector
        feature_vector = get_feature_vector(processed_tweet, stop_words)
        #3) Generate the dataset
        tweets.append((feature_vector, sentiment))
```

Code Snippet 1 – Calling the #1) normalisation step and the #2) the feature vector generation

The normalisation process involves the following steps (c.f. *Code Snippet 2*). For each word in a tweet:

i) Convert to lower case.
ii) Remove the "b" character inserted at the beginning of each tweet.
iii) Remove the "rt" character inserted at the beginning of each tweet.
iv) Convert the "www." and "https://" address into a single tag "URL".
v) Remove the "@username" and replace with AT_USER.
vi) Remove unnecessary white spaces.
vii) Convert hashtags to the exact same words (e.g. '#happy' to 'happy'), as they may contain a sentiment information.
viii) Trim any white spaces at the beginning and end of the tweet
ix) Remove single and double quotes

Points iv) and v) above are implemented as the proposed architecture does neither follow the URL, nor attempt to establish the message originator. A message from Warren Buffet may have more impact than an opinion voiced by a mundane user, however this is a factor not in scope for this project.

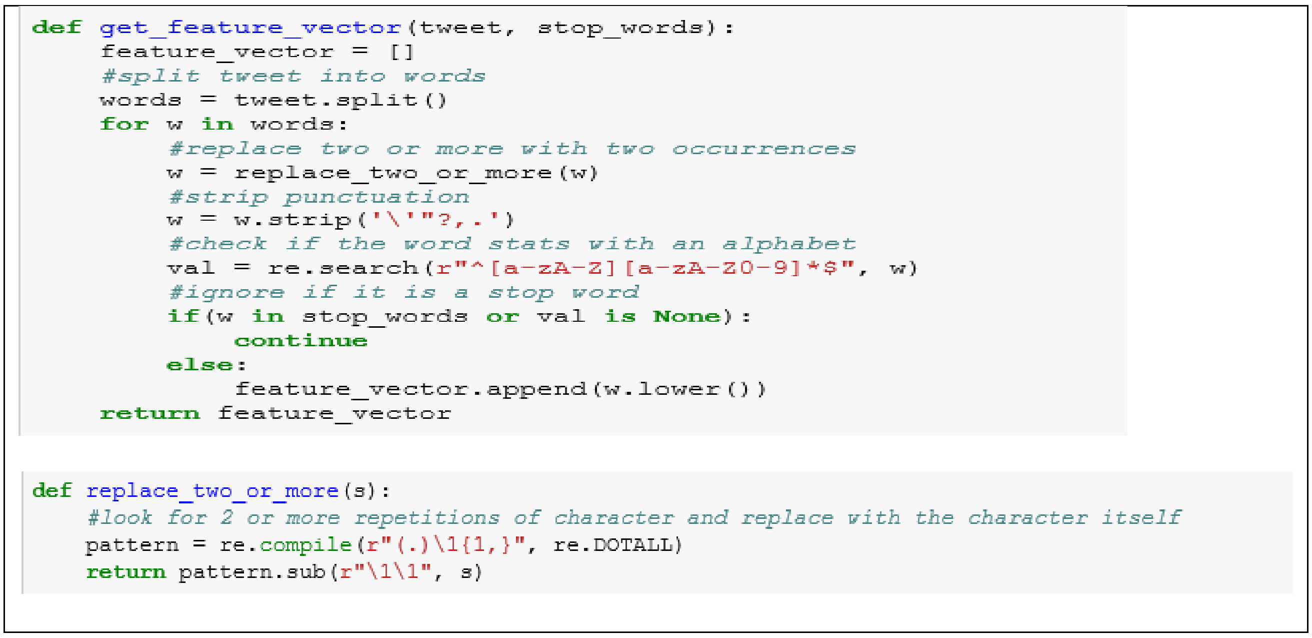


```
#Tweet pre-processing
def process_tweet(tweet):
    #Convert to lower case
    tweet = tweet.lower()
    #remove the characters b'
    tweet = tweet[1:]
    #Remove the 'rt' characters
    tweet = tweet.strip('rt')
    #Convert www.* or https?://* to URL
    tweet = re.sub('((www\.[^\s]+)|(https?://[^\s]+))','URL',tweet)
    #Convert @username to AT_USER
    tweet = re.sub('@[^\s]+','AT_USER',tweet)
    #Remove additional white spaces
    tweet = re.sub('[\s]+', ' ', tweet)
    #Replace #word with word
    tweet = re.sub(r'#([^\s]+)', r'\1', tweet)
    #Trim
    tweet = tweet.strip('\'"')
    #Remove single quotes
    tweet = tweet.replace("'", "")
    #Remove double quotes
    tweet = tweet.replace('"', "")
    return tweet
```

Code Snippet 2 – The *process_tweet()* function normalises the tweets

The feature vector creation, as shown in *Code Snippet 3* below reduces each tweet to a word list that contain the sentiment information. The bag of words (i.e. a feature vector of unigrams) is generated as follows. For each word in a tweet:

i) The *replace_two_or_more()* function removes the character duplication.
ii) Remove the punctuation (e.g. “,.?, etc.”)
iii) Keep words that start with a letter of the alphabet (e.g. any numbers are not considered)
iv) Ignore stop words as they do not convey significant information. Therefore, they are discarded (e.g. ‘a’,’about’, above’, etc.). A list of approximately 300 stop words were copied from [10] and stored in the *./data/processor_static_data /stop_words.csv* file.



```
def get_feature_vector(tweet, stop_words):
    feature_vector = []
    #split tweet into words
    words = tweet.split()
    for w in words:
        #replace two or more with two occurrences
        w = replace_two_or_more(w)
        #strip punctuation
        w = w.strip('\'"?,.')
        #check if the word stats with an alphabet
        val = re.search(r"^[a-zA-Z][a-zA-Z0-9]*$", w)
        #ignore if it is a stop word
        if(w in stop_words or val is None):
            continue
        else:
            feature_vector.append(w.lower())
    return feature_vector
```

```
def replace_two_or_more(s):
    #look for 2 or more repetitions of character and replace with the character itself
    pattern = re.compile(r"(.)\1{1,}", re.DOTALL)
    return pattern.sub(r"\1\1", s)
```

Code Snippet 3 – The *get_feature_vector()* generates the bag of words for each tweet

Table 7 shows the result of the above transform on two randomly selected tweeter messages.

Sample 1
Original tweet
2017-01-19 14:48:33,"b""Alphabet (GOOGL) Buys Twitter's Platform for Developers #mobiledev https://t.co/BCq2sK2i7U"""
Output (after normalisation &feature generation)
(['alphabet', 'buys', 'twitters', 'platform', 'developers', 'mobiledev'], '2017-01-19 14:48:33')

Sample 2
Original tweet
2017-01-19 14:48:32,"b""#Google won't give up on the failed #GooglePlus. Noble, but pointless. https://t.co/bUj0dPkuBD $GOOG $GOOGL #socialmedia #TechNews"""
Output (after normalisation &feature generation)
(['google', 'wont', 'failed', 'googleplus', 'noble', 'pointless', 'socialmedia', 'technews'], '2017-01-19 14:48:32'),

Table 7 – Bag of word extraction output

Feature vector sentiment generation

The final stage of this process is to tag each unigram against a sentiment and deduce the overall sentiment of the sentence. The full code is available in Appendix E.
First, each word is checked against a custom-made trading corpus (c.f. *./data/processor_static_data /custom_trading_corpus.csv*) to extract the corresponding sentiment (c.f. *Code Snippet 4)*. The trading corpus is a list of words used by stock market trading experts. These words have a definition and sentiment specific to this domain. Lexical resource for opinion mining, such as *SentiWordNet* do not support this vocabulary in the context of stock trading. At best, it returns a *Neutral* sentiment for these words. At worst, it returns an exception, as they are not recognised. The stock trading words were selected from [4] and [19] and tagged manually, as shown in table 7bis.

```
for word in bag_of_words:
    bFound = False
    sentiment = NA
    word = word.strip()
    #Look for the sentiment in the custom trading corpus
    #and increment the one of the counters (positive,negative,neutral)
    for row_ctc in custom_traiding_corpus_deep_copy:
        if row_ctc[0].strip() == word:
            if row_ctc[2].strip() == POSITIVE_SENTIMENT:
                count_positive += 1
                bFound = True
            elif row_ctc[2].strip() == NEGATIVE_SENTIMENT:
                count_negative += 1
                bFound = True
            elif row_ctc[2].strip() == NEUTRAL_SENTIMENT:
                count_neutral += 1
                bFound = True
            else:
                bFound = False
```

Code Snippet 4 – Sentiment counter incrementation logic for word belonging into the trading corpus

term,definition,sentiment
…
hedge,this is used to limit your losses,***negative***
rally,a rapid increase in the general price level of the market or of the price of a stock,***positive***
amaranthed,a fund that takes very large bets and collapses (like the energy hedge fund Amaranth did in 2006),***negative***
bear,an investor who has a negative view on a market and is likely to be net short,***negative***
bearish,characterized by or associated with falling share prices,***negative***
blowup,when a large amount of investors in a fund or ETF redeem all capital from the fund due to poor performance (or other issues) causing the fund or ETF to become illiquid and close,***negative***
…

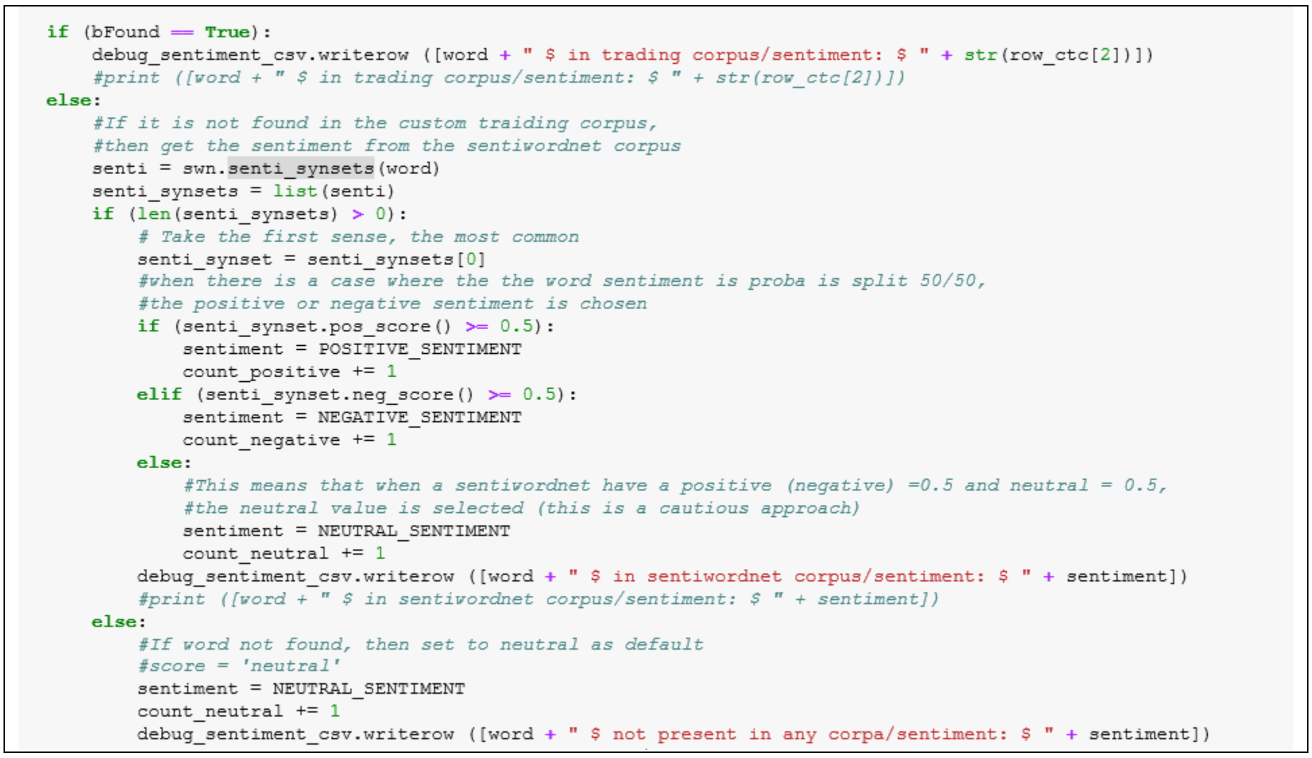
Table 7bis – Examples the trading corpus records (custom_trading_corpus.csv)

When a word is absent from this corpus, it is then passed through the *SentiWordNet* API [17] (c.f. *Code Snippet 5)*. *SentiWordNet* returns a sentiment likelihood (a.k.a. score) between 0 and 1 for each sentiment category. The category with the largest likelihood is then selected to represent the sentiment for this word. However, there are cases, where two or more categories share an equal likelihood, for e.g.:

- 50% split occurs between a *Positive* (*Negative*) category and the *Neutral* category, or
- 50% split occurs between a *Positive* and a *Negative* category.

In the first case the *Positive* (*Negative*) category is preferred. In the second case, the *Neutral* category is selected. This logic is represented in *Code Snippet 5.*

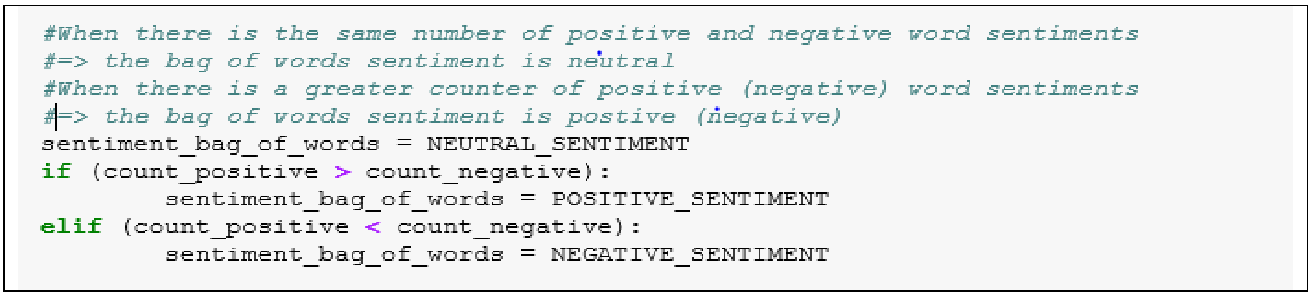
When a word is absent from both the corpus and the *SentiWordNet* API, then it is tagged with a *Neutral* sentiment.



```
if (bFound == True):
    debug_sentiment_csv.writerow ([word + " $ in trading corpus/sentiment: $ " + str(row_ctc[2])])
    #print ([word + " $ in trading corpus/sentiment: $ " + str(row_ctc[2])])
else:
    #If it is not found in the custom traiding corpus,
    #then get the sentiment from the sentiwordnet corpus
    senti = swn.senti_synsets(word)
    senti_synsets = list(senti)
    if (len(senti_synsets) > 0):
        # Take the first sense, the most common
        senti_synset = senti_synsets[0]
        #when there is a case where the the word sentiment is proba is split 50/50,
        #the positive or negative sentiment is chosen
        if (senti_synset.pos_score() >= 0.5):
            sentiment = POSITIVE_SENTIMENT
            count_positive += 1
        elif (senti_synset.neg_score() >= 0.5):
            sentiment = NEGATIVE_SENTIMENT
            count_negative += 1
        else:
            #This means that when a sentiwordnet have a positive (negative) =0.5 and neutral = 0.5,
            #the neutral value is selected (this is a cautious approach)
            sentiment = NEUTRAL_SENTIMENT
            count_neutral += 1
        debug_sentiment_csv.writerow ([word + " $ in sentiwordnet corpus/sentiment: $ " + sentiment])
        #print ([word + " $ in sentiwordnet corpus/sentiment: $ " + sentiment])
    else:
        #If word not found, then set to neutral as default
        #score = 'neutral'
        sentiment = NEUTRAL_SENTIMENT
        count_neutral += 1
        debug_sentiment_csv.writerow ([word + " $ not present in any corpa/sentiment: $ " + sentiment])
```

Code Snippet 5 – Sentiment counter increment logic for words belonging to the trading corpus

Early observations of the sentiment data scores showed that deducing the sentiment of a feature vector based on the highest frequency of a *Positive*, *Negative* or *Neutral* sentiment generated a systematic *Neutral* sentiment for each feature vector. This is due to the important skew of *Neutral* sentiment unigrams in each vector. Consequently, it was decided to give a larger weight to either *Positive* or *Negative* sentiments. The logic is based on frequency of *Positive* (*Negative*) sentiment in a vector. The presence of a greater number of *Positive* (*Negative*) sentiments in a feature vector induces an overall *Positive* (*Negative*) sentiment. When this is not the case, the feature vector is tagged with a *Neutral* sentiment (c.f. Code Snippet 6). The results of this logic are shown in Table 8.



```
#When there is the same number of positive and negative word sentiments
#=> the bag of words sentiment is neutral
#When there is a greater counter of positive (negative) word sentiments
#=> the bag of words sentiment is postive (negative)
sentiment_bag_of_words = NEUTRAL_SENTIMENT
if (count_positive > count_negative):
        sentiment_bag_of_words = POSITIVE_SENTIMENT
elif (count_positive < count_negative):
        sentiment_bag_of_words = NEGATIVE_SENTIMENT
```

Code Snippet 6 – The feature vector sentiment tagging algorithm

```
neutral $ in sentiwordnet corpus/sentiment: $ neutral
bulls $ in sentiwordnet corpus/sentiment: $ neutral
need $ in sentiwordnet corpus/sentiment: $ neutral
retake $ in sentiwordnet corpus/sentiment: $ neutral
bears $ in sentiwordnet corpus/sentiment: $ neutral
need $ in sentiwordnet corpus/sentiment: $ neutral
weekly $ in sentiwordnet corpus/sentiment: $ neutral
new $ in sentiwordnet corpus/sentiment: $ neutral
long $ in trading corpus/sentiment: $ positive
"sentiment_scores: {'positive': 1, 'negative': 0, 'neutral': 8}"
***********************************
"***Analysis bag of words: (['neutral', 'bulls', 'need', 'retake', 'bears', 'need', 'weekly', 'new',
'long'], '2017-01-04 20:01:55') / sentiment: positive"
***********************************
```

Table 8 – Example of a sentiment generated for a feature vector. Most words' sentiment comes from the *SentiWordNet NTLK* library, one word ('long') comes from the trading corpus.

Testing impact of sentiment

In order to establish the impact of the sentiment on the trend prediction, a base data set is constructed. It contains, but is not limited to, the set of attributes that are used by the different machine learning models. The columns description is available in table 9 below. Each row contains the data for a 5 minutes' tick interval (a.k.a. period). The code is available in Appendix F.

Raw Market Data Columns	
Date	Date/time (e.g. 30/12/2016 16:14:06)
Open	The price at the start of the period
High	The highest in the period
Low	The lowest in the period
Close	The price at the end of the period
Volume	The volume of share exchange during the period
Derived Market Data Columns	
Average	The average price, i.e. (High+Low+Close)/3
%Move	The percentage move between two period
%Abs_Move	The absolute percentage move between two period
Log_Return	The price log return for the period, i.e. log (Average t+1 / Average t)
Up_Down	When Log_Return > epsilon, then *Up* When Log_Return < -epsilon, then *Dowm* Else *Neural* Epsilon > 0 is a fudge parameter that enables the generation of the *neutral* label from the market data. When Epsilon is set to 0, then this creates a class unbalancing issue, as this is very rare that the market remains fully stable from one period to the next.
Up_Down_Value	When Up_Down = Up then 1 When Up_Down = Down then -1 Else 0
Log_Return_M1	The 1-day lag log return
Sma_1	The 1-day simple moving average
Sma_5	The 5-day simple moving average
Sma_20	The 20-day simple moving average
Volume_M1	The volume for the previous period (t-1)
Average Sentiment Information	
Sentiment	The average sentiment value for the period, generated in the above paragraph (c.f. Table 8)
Sentiment_Value	When Sentiment = Up then 1 When Sentiment = Down then -1 Else 0
Sentiment_Value_M1	The sentiment value for the previous period (t-1)

Table 9 – The base dataset

The problem at hand relates to a supervised classification problem:

- Supervised - the data is sectioned in two sets: a training and test sets, respectively accounting for 75% and 25% of the full population. The selected models are fitted against the training dataset and the test accuracy is generated against the test set.
- Classification - the response variable *Up_Down_Value* is a label containing three values: 1, 0 and -1. It indicates respectively an *Up*, *Down* and *Neutral* trend in a period. This is effectively a proxy for the daily price log return* (R_t) given as follows:

$$R_t = \log (Average_{t+1} / Average_t) \qquad (1)$$

The log return is usually used in finance, instead of the arithmetic return ($Average_{t+1}$ - $Average_t$) as it stabilises the variance and improves normality [14].

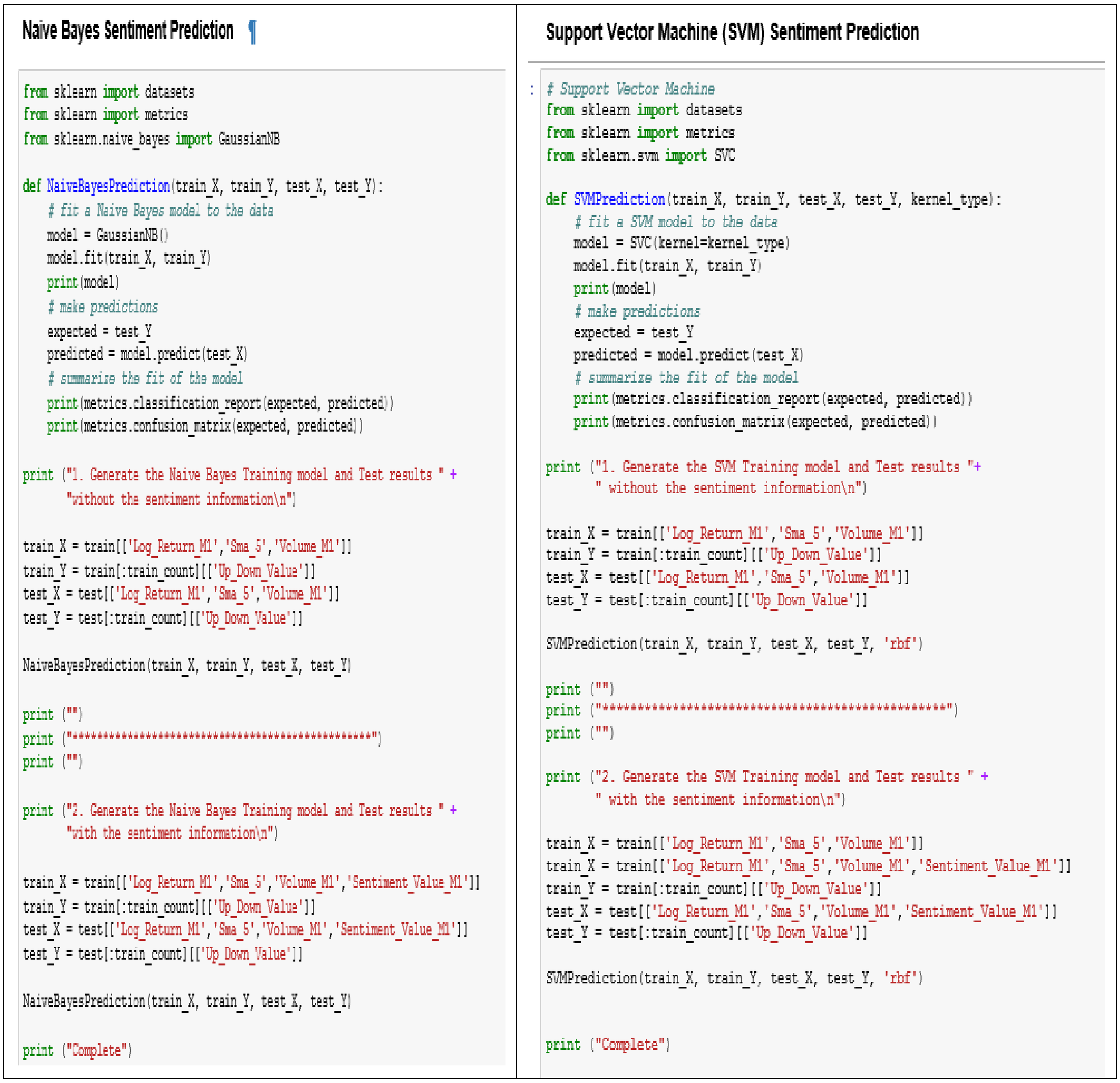
The response variable is set to a category value instead of a numeric value, to be easily handled by the supervised classification algorithm, supported in the Python *scikit-learn* machine learning framework. The explanatory variables are selected by a *forward selection* procedure. The initial small list of attributes start with *Log_Return_M1*, *Sma_1*, *Sma_5*, *Sma_20* and the *Volume_M1.* The attribute selection process ended with following attribute list: Sma_5_t + $Volume_{t-1}$ and R_{t-1} ,as they carried the most predictive power. The models M1, in e.q. (1) and M2, in e.q. (2) are built and run against a Naïve Bayes and Support Vector Machine (SVM) algorithms. This is to establish whether the addition of the sentiment information has a predictive power on the stock market trend. The only difference between (M1) and (M2) is the addition of the sentiment attribute ($Sent_{t-1}$) in M2.

$$\text{M1:} \quad R_t = R_{t-1} + Sma\_5_t + Volume_{t-1}, \tag{2}$$

$$\text{M2:} \quad R_t = R_{t-1} + Sma\_5_t + Volume_{t-1} + Sent_{t-1}, \tag{3}$$

Results

E.q. (2) is run against a Naïve Bayes and a Support Vector Machine (with an RBF kernel) model to establish the accuracy rate benchmarks, for the endogenous variables only (c.f. *Code Snippet 6*). They respectively showed a test accuracy of 36% and 29%. The fact that these accuracy levels are low, and well below the accuracy levels provided in the above literature, is not important in this limited experience. The only purpose is to establish whether the sentiment data has an impact on the prediction quality. Moreover, there is no attempt made to optimise the models' hyper-parameters in this project.
Eq (3), which contains the sentiment extra parameter ($Sent_{t-1}$), is run against the same models (c.f. Code Snippet 6). The accuracy rate is 37% and 29% for each instance.



Naive Bayes Sentiment Prediction ¶

```python
from sklearn import datasets
from sklearn import metrics
from sklearn.naive_bayes import GaussianNB

def NaiveBayesPrediction(train_X, train_Y, test_X, test_Y):
    # fit a Naive Bayes model to the data
    model = GaussianNB()
    model.fit(train_X, train_Y)
    print(model)
    # make predictions
    expected = test_Y
    predicted = model.predict(test_X)
    # summarize the fit of the model
    print(metrics.classification_report(expected, predicted))
    print(metrics.confusion_matrix(expected, predicted))

print ("1. Generate the Naive Bayes Training model and Test results " +
       "without the sentiment information\n")

train_X = train[['Log_Return_M1','Sma_5','Volume_M1']]
train_Y = train[:train_count][['Up_Down_Value']]
test_X = test[['Log_Return_M1','Sma_5','Volume_M1']]
test_Y = test[:train_count][['Up_Down_Value']]

NaiveBayesPrediction(train_X, train_Y, test_X, test_Y)

print ("")
print ("**************************************************")
print ("")

print ("2. Generate the Naive Bayes Training model and Test results " +
       "with the sentiment information\n")

train_X = train[['Log_Return_M1','Sma_5','Volume_M1','Sentiment_Value_M1']]
train_Y = train[:train_count][['Up_Down_Value']]
test_X = test[['Log_Return_M1','Sma_5','Volume_M1','Sentiment_Value_M1']]
test_Y = test[:train_count][['Up_Down_Value']]

NaiveBayesPrediction(train_X, train_Y, test_X, test_Y)

print ("Complete")
```

Support Vector Machine (SVM) Sentiment Prediction

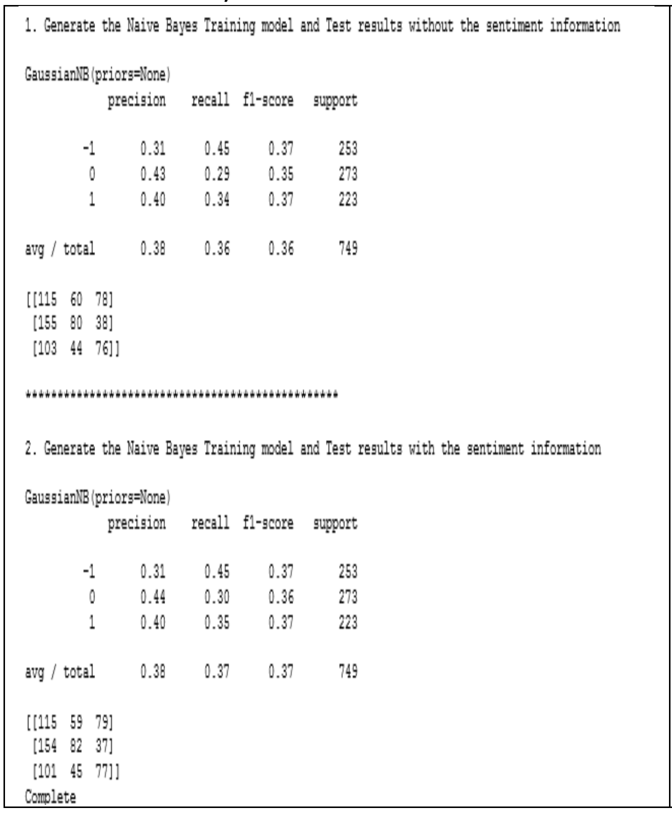
```python
# Support Vector Machine
from sklearn import datasets
from sklearn import metrics
from sklearn.svm import SVC

def SVMPrediction(train_X, train_Y, test_X, test_Y, kernel_type):
    # fit a SVM model to the data
    model = SVC(kernel=kernel_type)
    model.fit(train_X, train_Y)
    print(model)
    # make predictions
    expected = test_Y
    predicted = model.predict(test_X)
    # summarize the fit of the model
    print(metrics.classification_report(expected, predicted))
    print(metrics.confusion_matrix(expected, predicted))

print ("1. Generate the SVM Training model and Test results "+
       " without the sentiment information\n")

train_X = train[['Log_Return_M1','Sma_5','Volume_M1']]
train_Y = train[:train_count][['Up_Down_Value']]
test_X = test[['Log_Return_M1','Sma_5','Volume_M1']]
test_Y = test[:train_count][['Up_Down_Value']]

SVMPrediction(train_X, train_Y, test_X, test_Y, 'rbf')

print ("")
print ("**************************************************")
print ("")

print ("2. Generate the SVM Training model and Test results " +
       " with the sentiment information\n")

train_X = train[['Log_Return_M1','Sma_5','Volume_M1']]
train_X = train[['Log_Return_M1','Sma_5','Volume_M1','Sentiment_Value_M1']]
train_Y = train[:train_count][['Up_Down_Value']]
test_X = test[['Log_Return_M1','Sma_5','Volume_M1','Sentiment_Value_M1']]
test_Y = test[:train_count][['Up_Down_Value']]

SVMPrediction(train_X, train_Y, test_X, test_Y, 'rbf')


print ("Complete")
```

Code Snippet 6 – The Naïve Bayes and SVM confusion matrices and quality measures code (c.f. sentiment_analysis.ipynb)

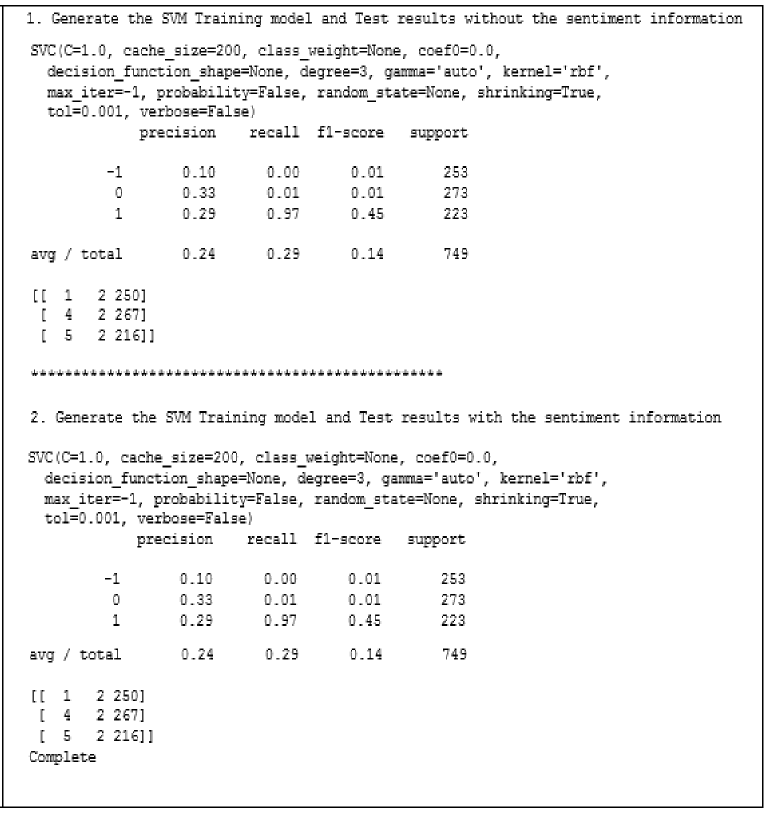
Total Population	Prediction Positive	Prediction Negative
Expected Positive	True Positive (TP)	False Negative (FN)
Expected Negative	False Positive (FP)	True Negative (TN)

Measure	Formula	Description
Sensitivity (a.k.a. recall)	TP/(TP+FN)	It measures the proportion of positives that are correctly identified (i.e. how good is a test at detecting the positives).
Specificity	TN/(TN+FP)	It measures the proportion of negatives that are correctly identified (i.e. how good is a test at detecting false alarms).
Precision	TP/(TP+FP)	It measures the proportion of positives that are relevant.
F1-Score	2*TP/(2*TP+FP+ FN)	It is a measure of the test accuracy. F1-Score is always between 0 (worst) and 1(best).
Accuracy	(TP+TN)/(TP+FN+FP+TN)	It is a measure of the statistical bias. Accuracy is between 0 (maximum bias) and 1 (no bias).

Table 10 – Quality measure definition



```
1. Generate the Naive Bayes Training model and Test results without the sentiment information

GaussianNB(priors=None)
             precision    recall  f1-score   support

         -1       0.31      0.45      0.37       253
          0       0.43      0.29      0.35       273
          1       0.40      0.34      0.37       223

avg / total       0.38      0.36      0.36       749

[[115  60  78]
 [155  80  38]
 [103  44  76]]

**************************************************

2. Generate the Naive Bayes Training model and Test results with the sentiment information

GaussianNB(priors=None)
             precision    recall  f1-score   support

         -1       0.31      0.45      0.37       253
          0       0.44      0.30      0.36       273
          1       0.40      0.35      0.37       223

avg / total       0.38      0.37      0.37       749

[[115  59  79]
 [154  82  37]
 [101  45  77]]
Complete
```



```
1. Generate the SVM Training model and Test results without the sentiment information

SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape=None, degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
             precision    recall  f1-score   support

         -1       0.10      0.00      0.01       253
          0       0.33      0.01      0.01       273
          1       0.29      0.97      0.45       223

avg / total       0.24      0.29      0.14       749

[[  1   2 250]
 [  4   2 267]
 [  5   2 216]]

**************************************************

2. Generate the SVM Training model and Test results with the sentiment information

SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape=None, degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
             precision    recall  f1-score   support

         -1       0.10      0.00      0.01       253
          0       0.33      0.01      0.01       273
          1       0.29      0.97      0.45       223

avg / total       0.24      0.29      0.14       749

[[  1   2 250]
 [  4   2 267]
 [  5   2 216]]
Complete
```

Table 11 – Naïve Bayes/SVM confusion matrices and quality measures for M1 and M2

Evaluation

This experiment demonstrates that sentiment may not have a significant impact on the ability to predict stock market trend. However, this result should be treated with caution due to the series of limitations discussed in the next section.

Limitations

The results of this experiment are affected by numerous limitations:

- The custom trading corpus does not contain all possible stock trading words, only the most commonly used. Furthermore, it only contains one word version, i.e. there is no plural, conjugated verbs, etc. This means the word 'sells' would not be found, although 'sell' exists in the corpus. As the current code does not implement stemming (root finding), the custom corpus has a limited power in matching the custom corpus word list with words not supported by the *SentiWordNet API*.
- An initial testing of the *SentiWordNet* API showed that words such as 'soar', 'plummet' are returned as 'neutral' in *SentiWordNet*, when they should be considered as either a *Positive* or *Negative* sentiment. The *Vader* corpus is also investigated, but it showed the same limitations.
- Abbreviations and spelling mistakes are neither handled by the *SentiWordNet* nor the custom corpus. Therefore, a *Neutral* sentiment is generated by the current code. This could have the tendency to skew the feature vectors sentiment towards the *Neutral* category.
- The words are organised in a feature vector of unigrams. Therefore, the sentiment resulting from the context is overlooked. For example, a message displaying "Buy GOOGL, sell if threshold xzy is attained". Then the sentiment for 'Buy' and 'Sell' would cancel one another. This would produce an overall *Neutral* sentiment for this message, instead of a *Positive* sentiment.
- All users are treated as equal, in terms of opinion quality. A lambda user expressing an opinion and an expert opinion are weighted equally. If they were to disagree, during a 5 minutes' period, then the overall sentiment would result in a *Neutral* sentiment, instead of being the sentiment expressed by the expert.
- Only tweets posted in English are collected. This means any user sentiments writing in other languages are discarded by default.
- The tweet data is sparse over a 5 minutes' period Therefore, a few messages could contribute heavily on the average sentiment outcome.
- Numerous tweets relating to GOOGL are not directly concerned with the GOOGL stock price, or factors potentially influencing it. These tweets affect the overall sentiment but they should be filtered out as they create noise. For example, the message 'The fascinating life of Google founder and Alphabet CEO Larry Page' carries a *Neutral* vector sentiment. This has in turn an impact on the overall sentiment in the period.
- There are numerous re-tweets at different timestamp (for example 'The fascinating life of Google founder and Alphabet CEO Larry Page' is tweeted 8 times), which skews the overall sentiment.
- The machine learning test accuracy prediction is a bit crude. Usually, a sliding time window is used to generate average test accuracy result for a time series.

Conclusion

The intra-day sentiment analysis of Twitter's messages, relating to the GOOGL stock, over a 20 days' period, showed that sentiments do not seem to have a significant prediction power on the stock trend prediction. This conclusion is disappointing and contradicts results reached in the literature. However, this experiment differs from the proposed literature in that it concerns the ability to predict sentiment in a 5 minutes' time window. The analysis suffers from sparse tweet messages in such a small period of time. In the worst-case scenario, one message would entirely contribute to the sentiment for that period. This is not an issue faced in the current research, as the minimum time bucket is one day. Furthermore, contrary to financial news messages, tweets content quality such as i) the message relevance, ii) the spelling correctness and iii) the writers' opinion play an important role in the ability of the entire proposed architecture to suggest a correct overall sentiment. Furthermore, it is demonstrated that the *SentiWordNet* corpus is limited in its ability to correctly tag sentiments. A first improvement step would be to weigh messages by their writer expertise levels. Then, messages/articles should be gathered from other sources such as other social media platforms (e.g. Facebook) or financial news agencies (e.g. Reuters). The message sentiment could also be weighted by the quality of the source of information. For example, a message/article coming from a renowned institution could bear a heavier weight than a tweet. A stemmer and lemmatiser could also be implemented as part of a more encompassing solution to enhance the precision of the sentiment generation.

Bibliography

[1] Aase KG., (2010), *Text Mining of News Articles for Stock Price Predictions* [Online]*, Available at: https://pdfs.semanticscholar.org/e9a4/7336c8acebe1e7798482773de2a882c96dff.pdf* [08-Dec-2016]

[2] Azar P.D., (2009), *Sentiment Analysis in Financial News* [Online], Available at: http://people.csail.mit.edu/azar/wp-content/uploads/2011/09/thesis.pdf [Accessed: 08-Dec-2016]

[3] *A look at the largest companies in the S&P 500* [Online], Associated Press, Available at: http://finance.yahoo.com/news/look-largest-companies-p-500-230815867.html [Accessed: 08-Dec-2016]

[4] *Appendix: Finance Slang* [Online], Hedgeable, Available at: https://www.hedgeable.com/education/trading-slang [Accessed: 08-Dec-2016]

[5] Ding, T., Fang, V., Zuo, D. (NA), *Stock Market Prediction based on Time Series Data and Market Sentiment* [Online], Available at: http://murphy.wot.eecs.northwestern.edu/~pzu918/EECS349/final_dZuo_tDing_vFang.pdf [Accessed: 08-Dec-2016]

[6] Hajek P., Olej V., Myskova R. (2013), *Forecasting Stock Prices using Sentiment Information in Annual Reports – A Neural Network and Support Vector Regression Approach*, WSEAS TRANSACTIONS on BUSINESS and ECONOMICS, Issue 4, Volume 10.

[7] Hutto, C.J., Gilbert, E., (NA), *VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text* [Online], Available at: http://comp.social.gatech.edu/papers/icwsm14.vader.hutto.pdf [Accessed: 19-Jan-2017]

[8] Janardhana R., (2012), *How to build a twitter sentiment analyzer?* [Online] Available at: https://www.ravikiranj.net/posts/2012/code/how-build-twitter-sentiment-analyzer/#training-the-classifiers [Accessed: 08-Dec-2016]

[9] Joshi K, Bharathi H.N., Rao J. (NA), Stock Trend Prediction Using News Sentiment Analysis [Online], Available at: *https://arxiv.org/ftp/arxiv/papers/1607/1607.01958.pdf* [Accessed: 13-Apr-2017]

[10] List of English Stop Words [Online], Available at: *http://xpo6.com/list-of-english-stop-words/* [Accessed: 19-Jan-2017]

[11] Mao H., Counts S., Bollen J. (NA), Predicting Financial Markets: Comparing Survey, News, Twitter and Search Engine Data [Online], Available *at https://arxiv.org/pdf/1112.1051.pdf* [Accessed: 13-Apr-2017]

[12] Mittal A, Goel A. (NA), Stock Prediction Using Twitter Sentiment Analysis [Online], Available at: *http://cs229.stanford.edu/proj2011/GoelMittal-StockMarketPredictionUsingTwitterSentimentAnalysis.pdf* [Accessed: 13-Apr-2017]

[13] Olaniyan R., Stamate D., Lahcen O, Logofatu D. (2015), Sentiment and stock market volatility predictive modelling - A hybrid approach, In: Eric Gaussier; Longbing Cao; Patrick Gallinari; James Kwok; Gabriela Pasi and Osmar Zaiane, eds. Data Science and Advanced Analytics (DSAA), 2015. 36678 2015. IEEE International Conference on. Paris: IEEE, pp. 1-10.

[14] Olaniyan R., Stamate D., Logofatu D. (2015), Social web-based anxiety index's predictive information on S&P 500 revisited, Proceedings of the 3rd Intl. Symposium on Statistical Learning and Data Sciences.

[15] Phung, A. (NA), *Behavioural Finance: Key Concepts -Herd Behavior* [Online], Available at: *http://www.investopedia.com/university/behavioral_finance/behavioral8.asp* [Accessed: 08-Dec-2016]

[16] Pulling Yahoo Finance data using PYTHON [Online], Available at: *http://stackoverflow.com/questions/24233385/pulling-yahoo-finance-data-using-python* [Accessed: 13-Apr-2017]

[17] SentiWordNet [Online], Available at: *http://sentiwordnet.isti.cnr.it/*, [Accessed: 08-Dec-2016]

[18] *Technical Analysis* [Online], Investodepia, Available at: *http://www.investopedia.com/terms/t/technicalanalysis.asp* [Accessed: 08-Dec-2016]

[19] 25 Basic Stock Market Trading Terms you should know [Online], Available at: *http://www.timothysykes.com/2013/06/trading-terms-you-need-to-know/* [Accessed: 08-Dec-2016]

Appendix

Appendix A – The Market Data & Twitter Sentiment Extraction Code

The scheduler is a Python process that triggers the download of market data and Twitter sentiment on a hourly basis. The Twitter API is called via the *get_twitter_data.TwitterData()* method, whereas the YAHHO!Finance API is called via the *get_stock_price.pulldata()* function (c.f. *Code Snippet xx*). Each method is detailed in the below paragraph.

```python
import time
import datetime
import get_twitter_data
import get_stock_price

#This is a simple scheduler that fetch market data and
#Twitter's sentiments every hour
while True:
    #Get the current date
    current_date = datetime.datetime.now()
    current_date_str = current_date.strftime('%Y-%m-%d-%H.%M.%S')

    #The stock market ticker
    keyword = 'GOOGL'

    #Fetch the Twitter data for today
    the_time = 'today'
    twitterData = get_twitter_data.TwitterData()
    tweets = twitterData.getTwitterData(keyword, the_time)

    #Fetch the market data
    if (current_date > datetime.datetime(current_date.year, current_date.month, current_date.day, 21, 30, 0,0) and
        current_date < datetime.datetime(current_date.year, current_date.month, current_date.day, 23, 0, 0,0)):
        get_stock_price.pullData(keyword)

    print("ran at: " + current_date_str)
    time.sleep(3600)
```

Code Snippet 7 – The scheduler (c.f. *scheduler.py*)

The *pulldata()* function is borrowed from [16]. The objective is to down the price and volume information from the Yahoo!Finance API, transform the stream into a CSV and store it (c.f. *Code Snippet 8*).

```
#This code is borrowed from the below post
#http://stackoverflow.com/questions/24233385/pulling-yahoo-finance-data-using-python
#Modified by Frederic Marechal
import urllib.request
import time
import datetime
#This function takes a stock name (e.g. GOOGL), connect to the Yahoo!Finance API.
#It downloads the stock data for the date t, t-1 and t-2.
#It transforms the streamed data in a CSV and store the file locally.
#The result is a file containing the following attributes Date/Open/High/Low/Close/Volume
def pullData(stock):
  current_date = datetime.datetime.now().strftime('%Y_%m_%d')
  fileLine = "data/prices/" + stock + '_' +current_date + '_.csv'
  #Create the Yahoo!Finance request URL, based on the stock name
  urltovisit = 'http://chartapi.finance.yahoo.com/instrument/1.0/'+stock+'/chartdata;type=quote;range=3d/csv'
  #Call the Yahoo!Finance API and get the data
  with urllib.request.urlopen(urltovisit) as f:
    sourceCode = f.read().decode('utf-8')
  splitSource = sourceCode.split('\n')
  #Transform the stream into a CSV format and store.
  for eachLine in splitSource:
    splitLine = eachLine.split(',')
    if len(splitLine) == 6:
      if 'values' not in eachLine:
        saveFile = open(fileLine,'a')
        linetoWrite = eachLine+'\n'
        saveFile.write(linetoWrite)

  print('Pulled', stock)
  print('...')
  time.sleep(.5)
```

Code Snippet 8 – The market data download code (c.f. *get_stock_price.py*)

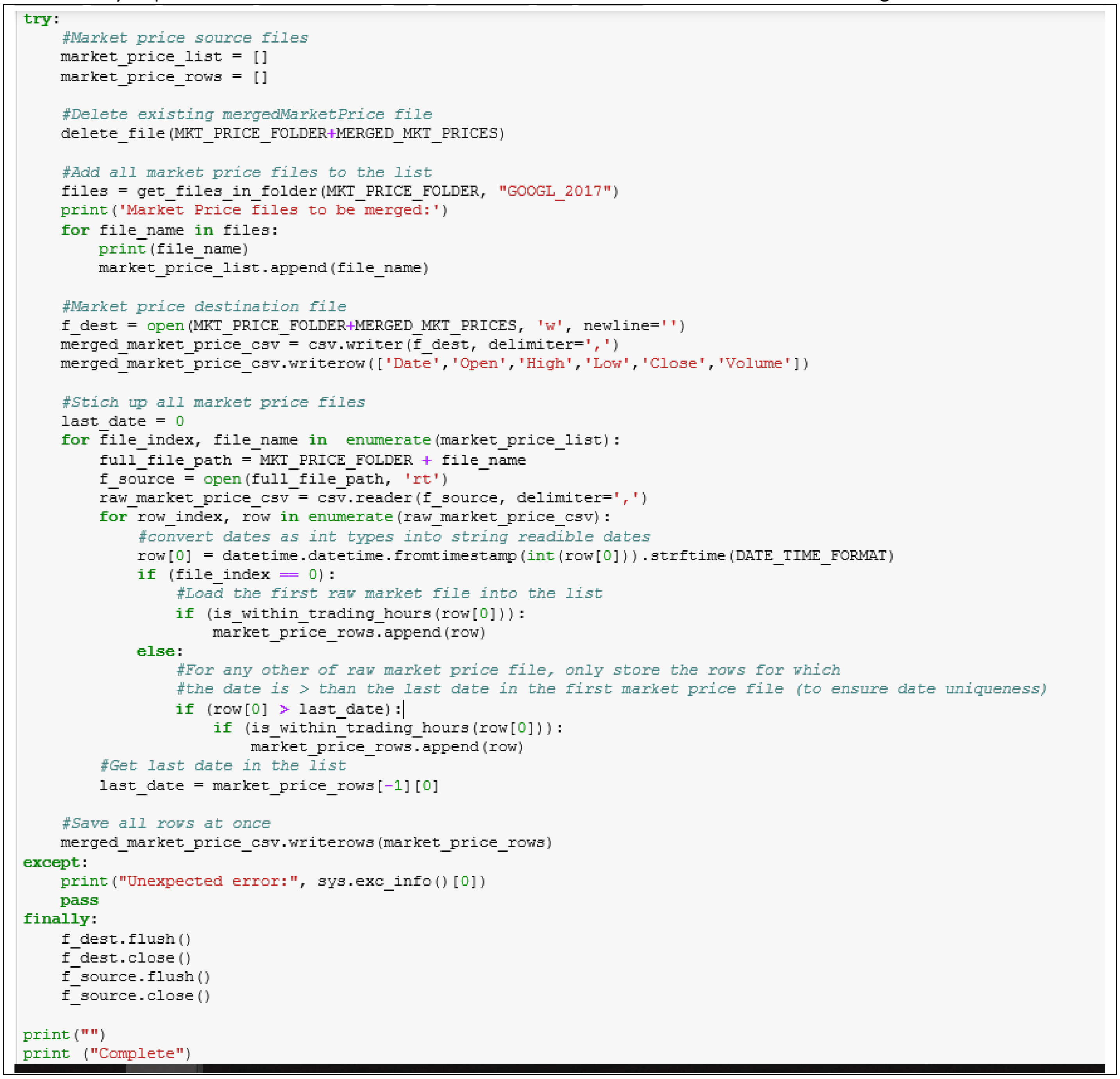
The next piece of code, borrowed and modified from [8], enables the connection to the Twitter's API to download targeted Twitter's messages, for example a number of attributes (e.g. the text and creatin_date) for a given stock (e.g. GOOGL). The connection necessitate a Twitter's authorisation/authentication, which parameters are defined in the *config.json* file, in order to access the data.

```
#This code is originally proposed by Janardhana [8]
#Modified by Frederic Marechal
import argparse
import urllib
import json
import datetime
import random
import os
import pickle
from datetime import timedelta
import oauth2
import urllib.parse
import csv
import get_twitter_data
import json
import uuid

#Class responsible for connecting to the Twitter API (via authorisation/authentication) and
#for collecting Twitter's data.
class TwitterData:
  #Class constructor - set the parameters that will be used by the object at runtime
  def __init__(self):
    self.currDate = datetime.datetime.now()
    self.weekDates = []
    self.weekDates.append(self.currDate.strftime("%Y-%m-%d"))
    for i in range(-1,7):
      dateDiff = timedelta(days=-i)
      newDate = self.currDate + dateDiff
      self.weekDates.append(newDate.strftime("%Y-%m-%d"))

  #Get the Twitter data stream based on a keyword (e.g. GOODL), and a timeframe (e.g. today, yesterday, etc..)
  def getTwitterData(self, keyword, time):
    self.weekTweets = {}
    #Set the current date to a string
    current_date = datetime.datetime.now().strftime('%Y-%m-%d-%H.%M.%S')
    #Get the Twitter information for a timeframe
    if(time == 'lastweek'):
      for i in range(0,6):
        params = {'since': self.weekDates[i+1], 'until': self.weekDates[i]}
        self.weekTweets[i] = self.getData(keyword, params)
      filename = 'data/weekTweets/weekTweets_'+urllib.parse.unquote(keyword.replace("+", " "))+ "_" + current_date + "_" + time +'.txt'
      outfile = open(filename, 'wb')
      pickle.dump(self.weekTweets, outfile)
      outfile.close()
    elif(time == 'yesterdayAndToday'):
      for i in range(0,2):
        params = {'since': self.weekDates[i], 'until': self.weekDates[i+1]}
        self.weekTweets[i] = self.getData(keyword, params)
    elif(time == 'today'):
      for i in range(0,1):
        params = {'since': self.weekDates[i], 'until': self.weekDates[i+1]}
        self.weekTweets[i] = self.getData(keyword, params)
    #Define a file name
    fileName = 'data/weekTweets/tweet' + '_' + keyword + '_' + time + '_' + 'tweets' +  "_" + current_date + "_" + time + '.csv'
    #Create a csv writer object and store both the schema and data into it
    csv_out = open(fileName, mode='w', newline='')
    writer = csv.writer(csv_out)
    fields = ['created_at', 'text']
    writer.writerow(fields)
    #print the values to the CSV
    print ("start writing Tweets to csv: " + keyword +  "_" + current_date + "_" + time )
    for item in self.weekTweets.values():
      #writes a row and gets the fields from the json object
      #screen_name and followers/friends are found on the second level hence two get methods)
      i = 0
      while (i < len(item)-1):
            writer.writerow([self.encode_string(item[i]),self.encode_string(item[i+1])])
            i = i+1
        csv_out.close()
        print ("End writing Tweets to csv: " + keyword +  "_" + current_date + "_" + time )

        return self.weekTweets

    #Encode a string to unicode
    def encode_string(self, str):
        return str.encode('unicode-escape',errors='strict')

    #The configuration parser. It loads the 'config.json' file and
    #gather the Twitter authorisation tokens to establish a connection with the Twitter's aPI
    def parse_config(self):
___config = {}
___# Load the config information from a file
___if os.path.exists('config.json'):
_____with open('config.json') as f:
_______config.update(json.load(f))
___else:
_____# Else load them from a command line
_____parser = argparse.ArgumentParser()

_____parser.add_argument('-ck', '--consumer_key', default=None, help='Your developper `Consumer Key`')
_____parser.add_argument('-cs', '--consumer_secret', default=None, help='Your developper `Consumer Secret`')
_____parser.add_argument('-at', '--access_token', default=None, help='A client `Access Token`')
_____parser.add_argument('-ats', '--access_token_secret', default=None, help='A client `Access Token Secret`')

_____args_ = parser.parse_args()
_____def val(key):
          return config.get(key)\
                 or getattr(args_, key)\
                 or input('Your developper `%s`: ' % key)
_____config.update({
          'consumer_key': val('consumer_key'),
          'consumer_secret': val('consumer_secret'),
          'access_token': val('access_token'),
          'access_token_secret': val('access_token_secret'),
        })
___return config

    #This function takes a Twitter's URL and establish a connection based on the configuration authorisation/authentication details
    def oauth_req(self, url, http_method="GET", post_body=None,
                 http_headers=None):
___#Load the configuration
___config = self.parse_config()
___#Get the required tokens
___consumer = oauth2.Consumer(key=config.get('consumer_key'), secret=config.get('consumer_secret'))
___token = oauth2.Token(key=config.get('access_token'), secret=config.get('access_token_secret'))
___#Establish a client authorisation request
___client = oauth2.Client(consumer, token)
___#Get the content information, on client's authorisation request
___resp, content = client.request(
        url,
        method=http_method,
        body=post_body or b"",
        headers=http_headers
      )
___return content

    #Define the Twitter data requirement and get the data
    def getData(self, keyword, params = {}):
        #Define the data properties to recover
        maxTweets = 100
        url = 'https://api.twitter.com/1.1/search/tweets.json?'
        data = {'q': keyword,
            'lang': 'en',
            'result_type': 'recent',
            'count': maxTweets,
            'include_entities': 0,
            'result_type':'mixed'}
        #Add any additional parameters
        if params:
            for key, value in params.items():
                data[key] = value
        #Create the url
        url += urllib.parse.urlencode(data)
        #Get the bag of data
        response = self.oauth_req(url)
        jsonData = json.loads(str(response,'utf-8'))
        tweets = []
        #Display errors
        if 'errors' in jsonData:
            print ("API Error")
            print (jsonData['errors'])
        else:
            #Only retain relevant attributes (the full ist is available at:
            #https://dev.twitter.com/rest/reference/get/search/tweets
            for item in jsonData['statuses']:
                tweets.append(self.defaultValue(item,'created_at'))
                tweets.append(self.defaultValue(item,'text'))
        return tweets

    #Define a default value for missing data
    def defaultValue(self, item, name, defaultValue=""):
        if not isinstance(item[name], (bool)):
            if len(item[name]) > 0:
                return item[name]
            else:
                return defaultValue
        return item[name];
```

Code Snippet 9 – The Twitter’s connection and data download code (c.f. *get_twitter_data.py*)

Appendix B – Market Price Pre-processor

The purpose of the below piece of code is to load the market data price files and merge them into a unique file. As each market data file contains data for the past 3 days, the merging process ensures the date/time continuity, and remove any duplicated records between two consecutive files. All rows are order in ascending order.

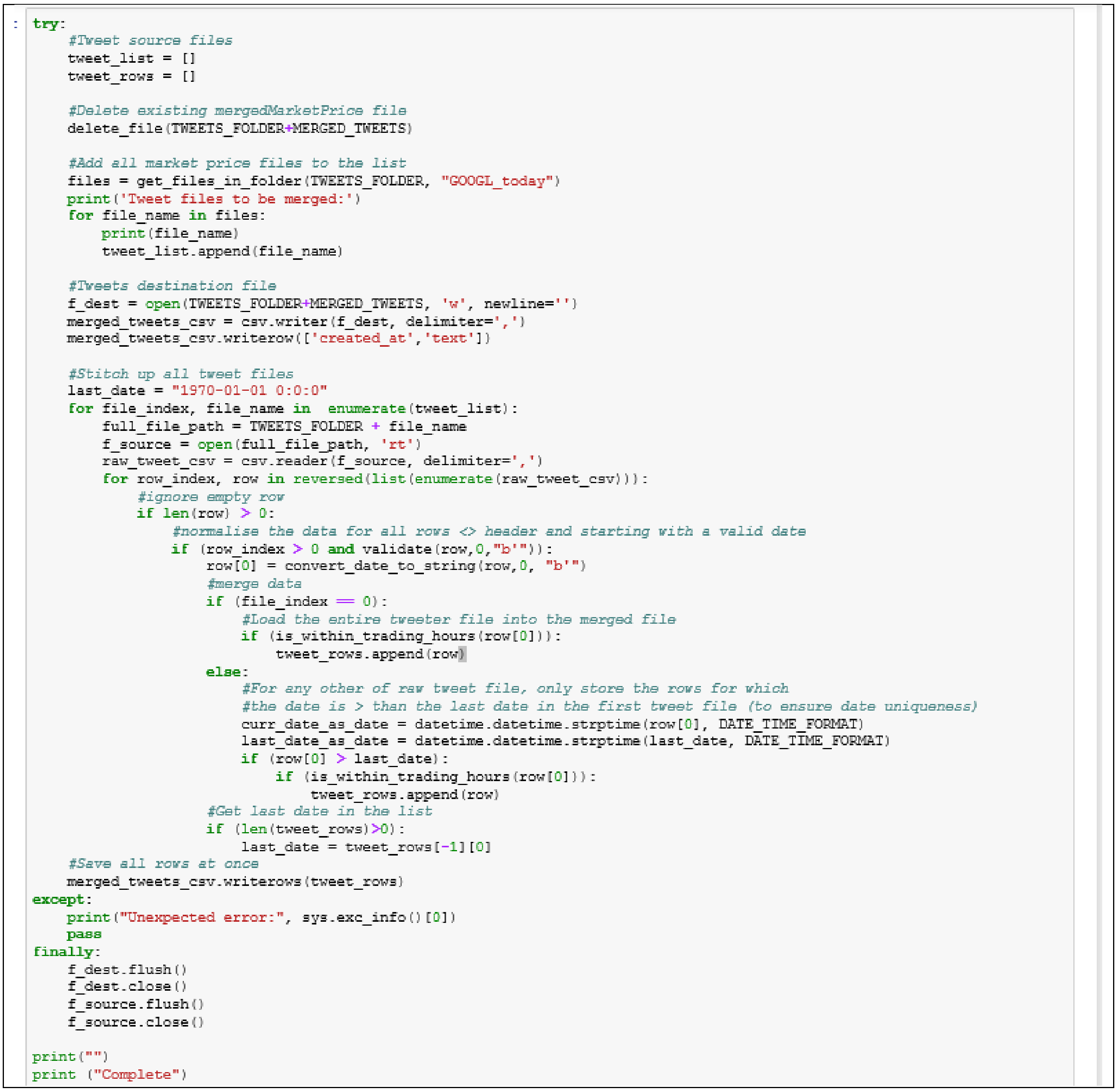


```
try:
    #Market price source files
    market_price_list = []
    market_price_rows = []

    #Delete existing mergedMarketPrice file
    delete_file(MKT_PRICE_FOLDER+MERGED_MKT_PRICES)

    #Add all market price files to the list
    files = get_files_in_folder(MKT_PRICE_FOLDER, "GOOGL_2017")
    print('Market Price files to be merged:')
    for file_name in files:
        print(file_name)
        market_price_list.append(file_name)

    #Market price destination file
    f_dest = open(MKT_PRICE_FOLDER+MERGED_MKT_PRICES, 'w', newline='')
    merged_market_price_csv = csv.writer(f_dest, delimiter=',')
    merged_market_price_csv.writerow(['Date','Open','High','Low','Close','Volume'])

    #Stich up all market price files
    last_date = 0
    for file_index, file_name in  enumerate(market_price_list):
        full_file_path = MKT_PRICE_FOLDER + file_name
        f_source = open(full_file_path, 'rt')
        raw_market_price_csv = csv.reader(f_source, delimiter=',')
        for row_index, row in enumerate(raw_market_price_csv):
            #convert dates as int types into string readible dates
            row[0] = datetime.datetime.fromtimestamp(int(row[0])).strftime(DATE_TIME_FORMAT)
            if (file_index == 0):
                #Load the first raw market file into the list
                if (is_within_trading_hours(row[0])):
                    market_price_rows.append(row)
            else:
                #For any other of raw market price file, only store the rows for which
                #the date is > than the last date in the first market price file (to ensure date uniqueness)
                if (row[0] > last_date):
                    if (is_within_trading_hours(row[0])):
                        market_price_rows.append(row)
        #Get last date in the list
        last_date = market_price_rows[-1][0]

    #Save all rows at once
    merged_market_price_csv.writerows(market_price_rows)
except:
    print("Unexpected error:", sys.exc_info()[0])
    pass
finally:
    f_dest.flush()
    f_dest.close()
    f_source.flush()
    f_source.close()

print("")
print ("Complete")
```

Code Snippet 10 – The market price pre-processor (c.f. *sentiment_analysis.ipynb*)

Appendix C – Market Price Pre-processor

The purpose of the below piece of code is each twitter file and merge them into one unique file. The process ensures the removal of empty messages, duplicated messages and order the tweets in date ascending order.

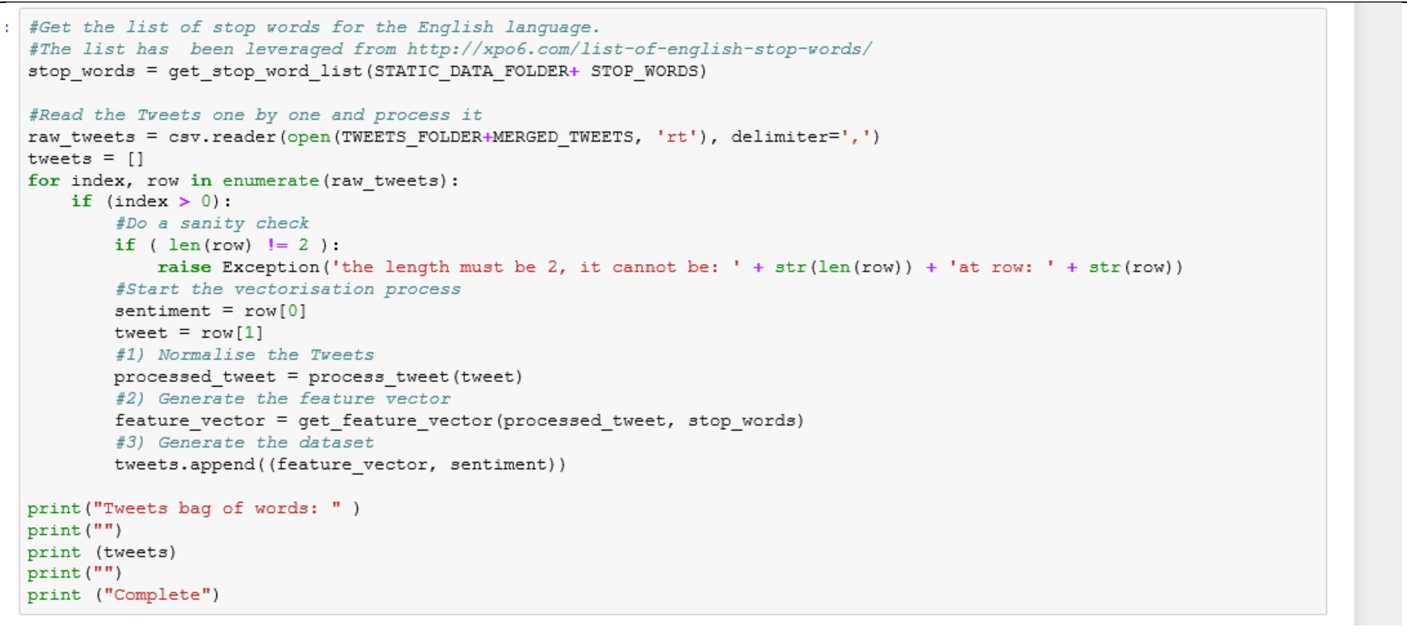


```
try:
    #Tweet source files
    tweet_list = []
    tweet_rows = []

    #Delete existing mergedMarketPrice file
    delete_file(TWEETS_FOLDER+MERGED_TWEETS)

    #Add all market price files to the list
    files = get_files_in_folder(TWEETS_FOLDER, "GOOGL_today")
    print('Tweet files to be merged:')
    for file_name in files:
        print(file_name)
        tweet_list.append(file_name)

    #Tweets destination file
    f_dest = open(TWEETS_FOLDER+MERGED_TWEETS, 'w', newline='')
    merged_tweets_csv = csv.writer(f_dest, delimiter=',')
    merged_tweets_csv.writerow(['created_at','text'])

    #Stitch up all tweet files
    last_date = "1970-01-01 0:0:0"
    for file_index, file_name in  enumerate(tweet_list):
        full_file_path = TWEETS_FOLDER + file_name
        f_source = open(full_file_path, 'rt')
        raw_tweet_csv = csv.reader(f_source, delimiter=',')
        for row_index, row in reversed(list(enumerate(raw_tweet_csv))):
            #ignore empty row
            if len(row) > 0:
                #normalise the data for all rows <> header and starting with a valid date
                if (row_index > 0 and validate(row,0,"b'")):
                    row[0] = convert_date_to_string(row,0, "b'")
                    #merge data
                    if (file_index == 0):
                        #Load the entire tweeter file into the merged file
                        if (is_within_trading_hours(row[0])):
                            tweet_rows.append(row)
                    else:
                        #For any other of raw tweet file, only store the rows for which
                        #the date is > than the last date in the first tweet file (to ensure date uniqueness)
                        curr_date_as_date = datetime.datetime.strptime(row[0], DATE_TIME_FORMAT)
                        last_date_as_date = datetime.datetime.strptime(last_date, DATE_TIME_FORMAT)
                        if (row[0] > last_date):
                            if (is_within_trading_hours(row[0])):
                                tweet_rows.append(row)
                    #Get last date in the list
                    if (len(tweet_rows)>0):
                        last_date = tweet_rows[-1][0]
    #Save all rows at once
    merged_tweets_csv.writerows(tweet_rows)
except:
    print("Unexpected error:", sys.exc_info()[0])
    pass
finally:
    f_dest.flush()
    f_dest.close()
    f_source.flush()
    f_source.close()

print("")
print ("Complete")
```

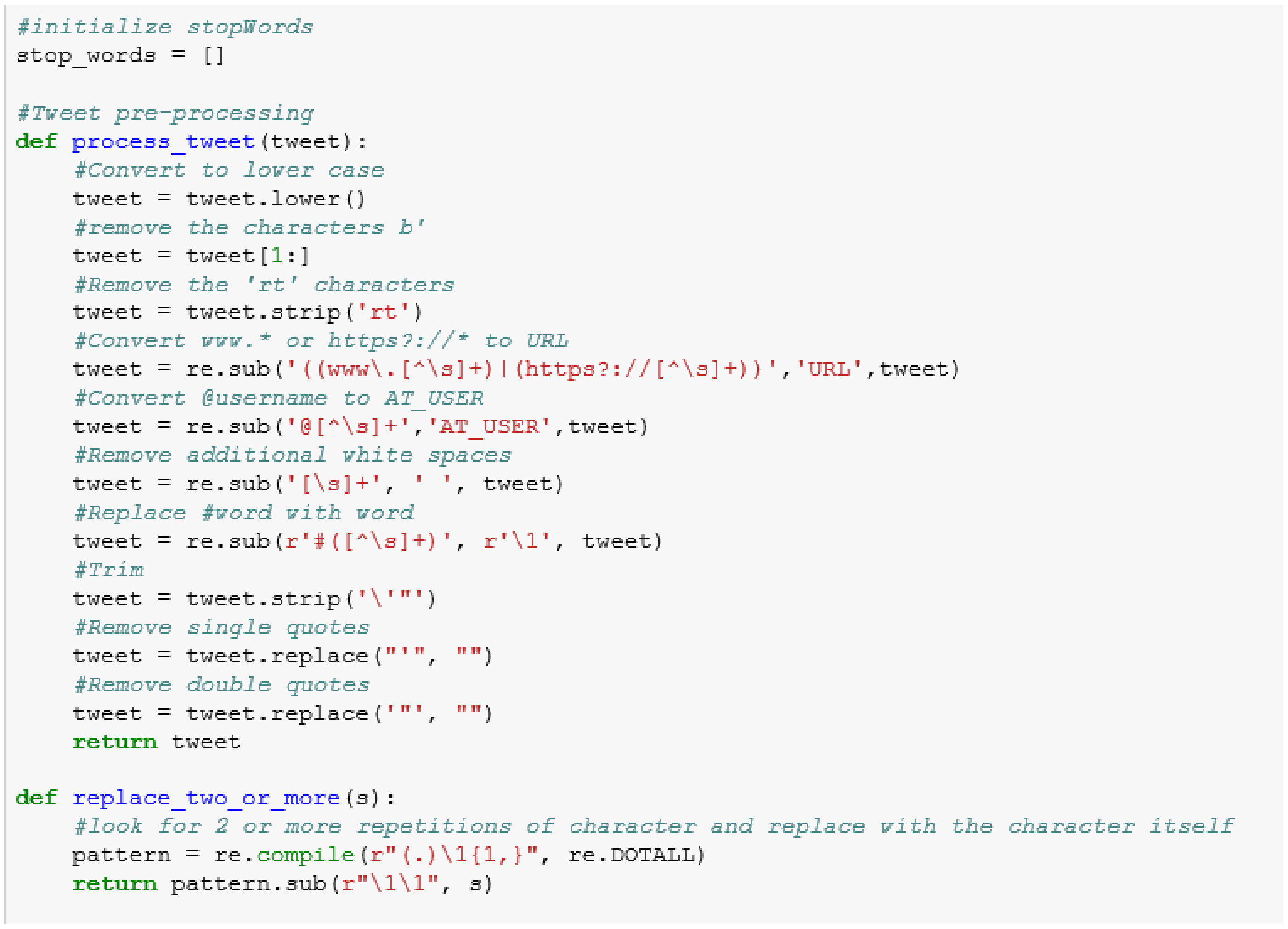
Code Snippet 11 – Twitter's messages pre-processor (c.f. *sentiment_analysis.ipynb*)

Appendix D – Generate Feature Vectors and Corresponding Sentiment

The purpose of the below piece of code is to normalise tweets and generate a bag of words.



```
#Get the list of stop words for the English language.
#The list has  been leveraged from http://xpo6.com/list-of-english-stop-words/
stop_words = get_stop_word_list(STATIC_DATA_FOLDER+ STOP_WORDS)

#Read the Tweets one by one and process it
raw_tweets = csv.reader(open(TWEETS_FOLDER+MERGED_TWEETS, 'rt'), delimiter=',')
tweets = []
for index, row in enumerate(raw_tweets):
    if (index > 0):
        #Do a sanity check
        if ( len(row) != 2 ):
            raise Exception('the length must be 2, it cannot be: ' + str(len(row)) + 'at row: ' + str(row))
        #Start the vectorisation process
        sentiment = row[0]
        tweet = row[1]
        #1) Normalise the Tweets
        processed_tweet = process_tweet(tweet)
        #2) Generate the feature vector
        feature_vector = get_feature_vector(processed_tweet, stop_words)
        #3) Generate the dataset
        tweets.append((feature_vector, sentiment))

print("Tweets bag of words: " )
print("")
print (tweets)
print("")
print ("Complete")
```



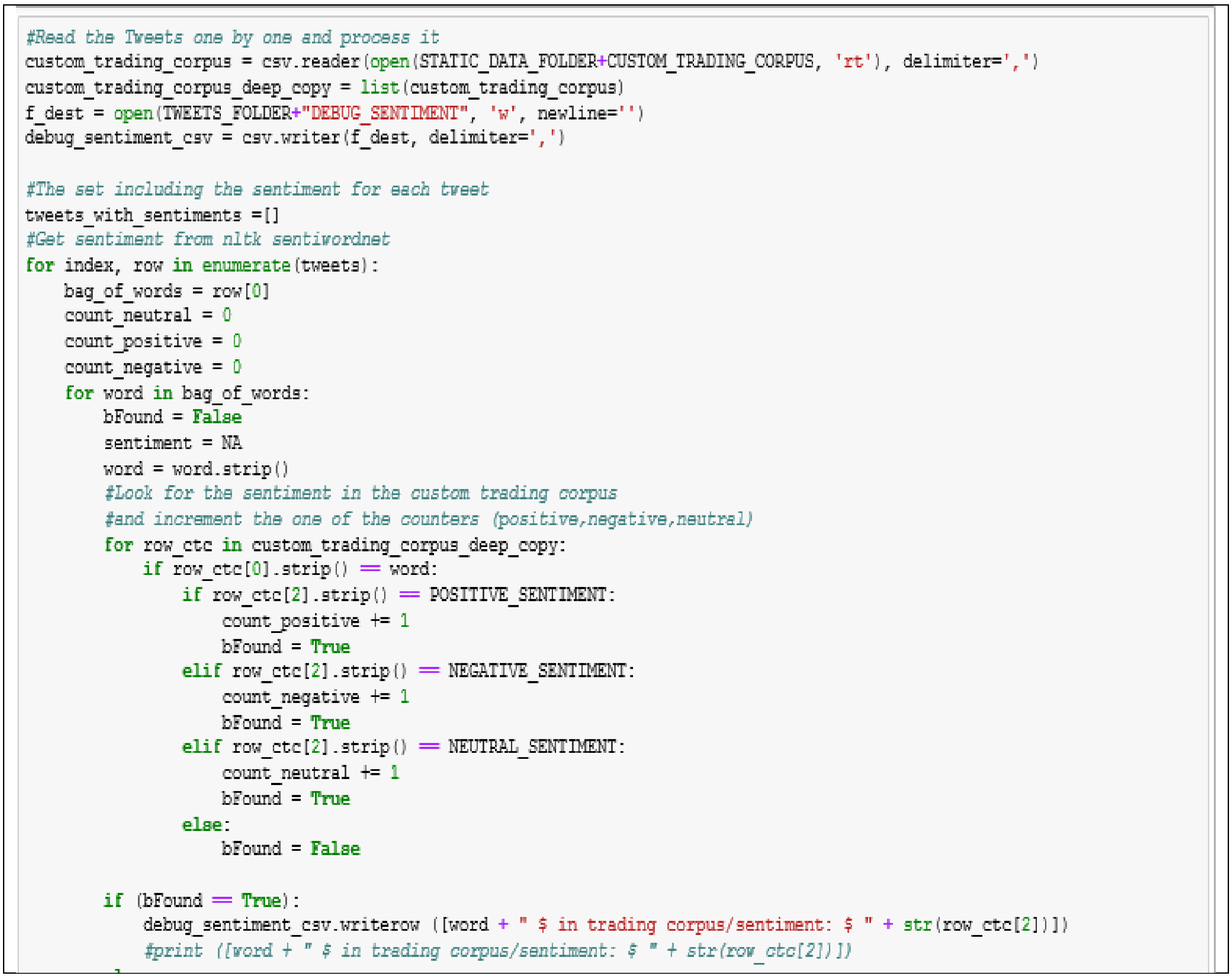
```
#initialize stopWords
stop_words = []

#Tweet pre-processing
def process_tweet(tweet):
    #Convert to lower case
    tweet = tweet.lower()
    #remove the characters b'
    tweet = tweet[1:]
    #Remove the 'rt' characters
    tweet = tweet.strip('rt')
    #Convert www.* or https?://* to URL
    tweet = re.sub('((www\.[^\s]+)|(https?://[^\s]+))','URL',tweet)
    #Convert @username to AT_USER
    tweet = re.sub('@[^\s]+','AT_USER',tweet)
    #Remove additional white spaces
    tweet = re.sub('[\s]+', ' ', tweet)
    #Replace #word with word
    tweet = re.sub(r'#([^\s]+)', r'\1', tweet)
    #Trim
    tweet = tweet.strip('\'"')
    #Remove single quotes
    tweet = tweet.replace("'", "")
    #Remove double quotes
    tweet = tweet.replace('"', "")
    return tweet

def replace_two_or_more(s):
    #look for 2 or more repetitions of character and replace with the character itself
    pattern = re.compile(r"(.)\1{1,}", re.DOTALL)
    return pattern.sub(r"\1\1", s)
```

```python
def get_stop_word_list(stop_word_list_fileName):
    #read the stopwords file and build a list
    stop_words = []
    stop_words.append('AT_USER')
    stop_words.append('URL')

    fp = open(stop_word_list_fileName, 'r')
    line = fp.readline()
    while line:
        word = line.strip()
        stop_words.append(word)
        line = fp.readline()
    fp.close()
    return stop_words

def get_feature_vector(tweet, stop_words):
    feature_vector = []
    #split tweet into words
    words = tweet.split()
    for w in words:
        #replace two or more with two occurrences
        w = replace_two_or_more(w)
        #strip punctuation
        w = w.strip('\'"?,.')
        #check if the word stats with an alphabet
        val = re.search(r"^[a-zA-Z][a-zA-Z0-9]*$", w)
        #ignore if it is a stop word
        if(w in stop_words or val is None):
            continue
        else:
            feature_vector.append(w.lower())
    return feature_vector
```

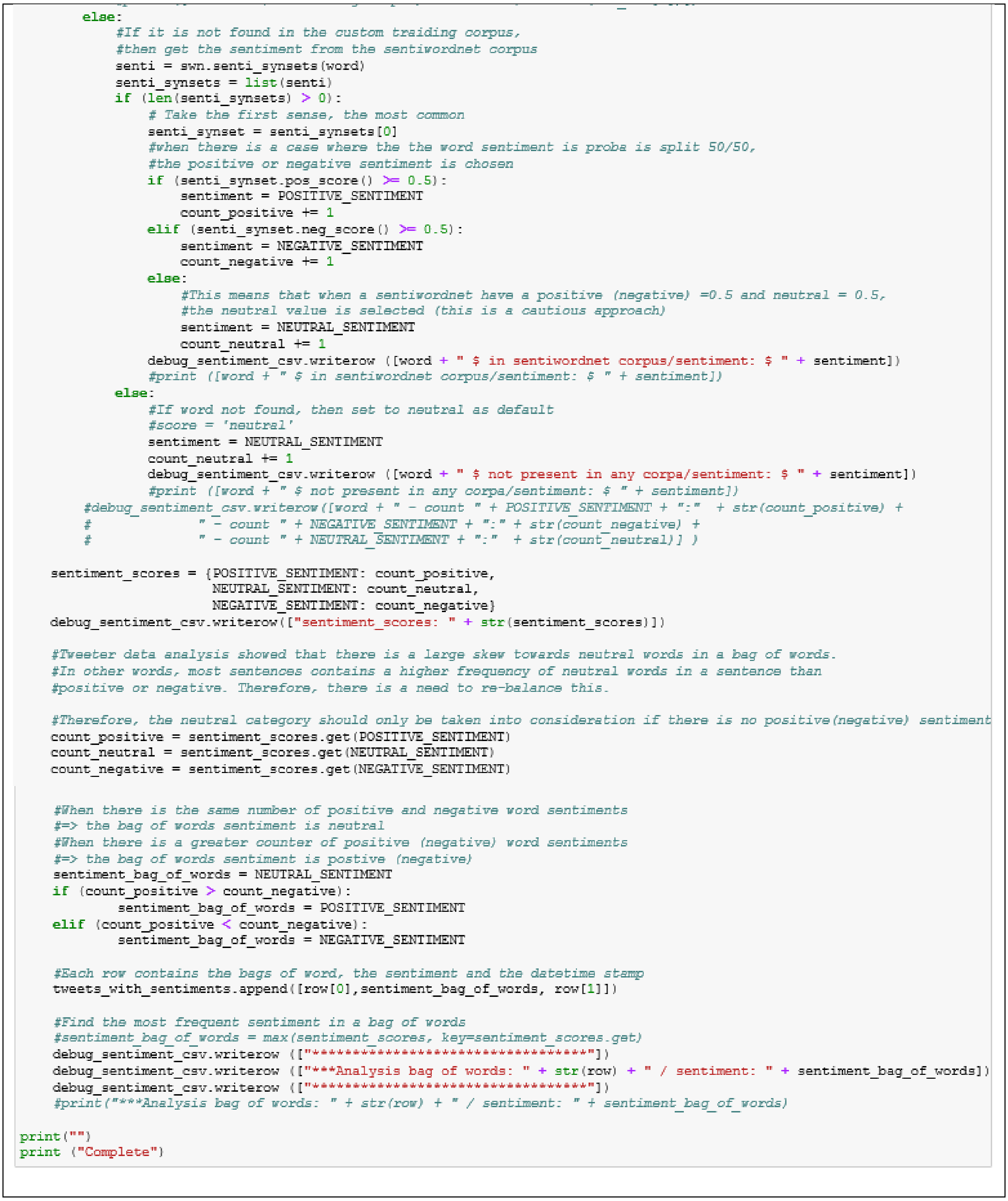
Code Snippet 12 – Twitter’s messages normalisation and feature creation (c.f. *sentiment_analysis.ipynb*)

Appendix E – Generate Feature Vectors and Corresponding Sentiment

The purpose of the below piece of code is to normalise tweets and generate a bag of words.



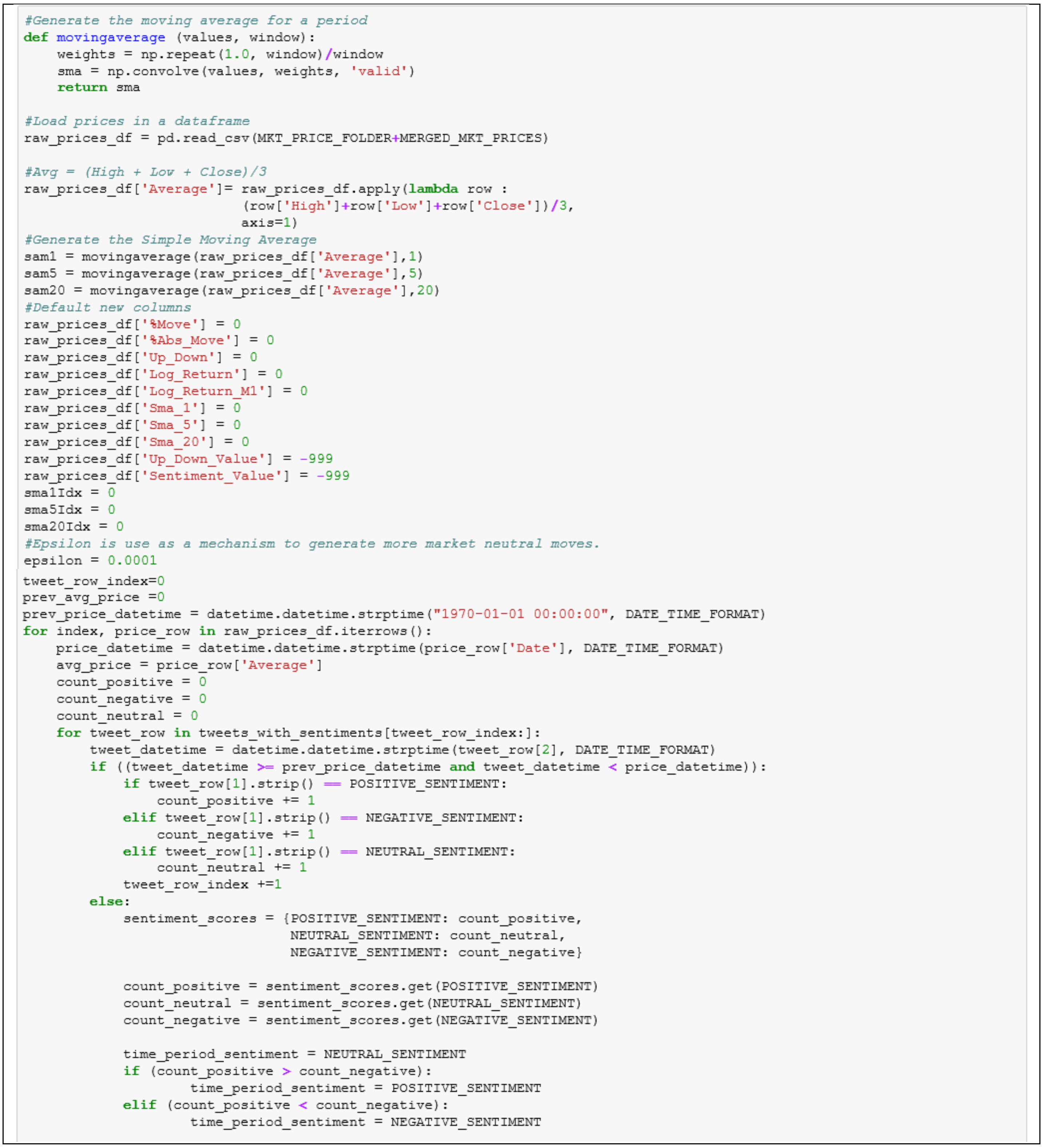
```python
#Read the Tweets one by one and process it
custom_trading_corpus = csv.reader(open(STATIC_DATA_FOLDER+CUSTOM_TRADING_CORPUS, 'rt'), delimiter=',')
custom_trading_corpus_deep_copy = list(custom_trading_corpus)
f_dest = open(TWEETS_FOLDER+"DEBUG_SENTIMENT", 'w', newline='')
debug_sentiment_csv = csv.writer(f_dest, delimiter=',')

#The set including the sentiment for each tweet
tweets_with_sentiments =[]
#Get sentiment from nltk sentiwordnet
for index, row in enumerate(tweets):
    bag_of_words = row[0]
    count_neutral = 0
    count_positive = 0
    count_negative = 0
    for word in bag_of_words:
        bFound = False
        sentiment = NA
        word = word.strip()
        #Look for the sentiment in the custom trading corpus
        #and increment the one of the counters (positive,negative,neutral)
        for row_ctc in custom_trading_corpus_deep_copy:
            if row_ctc[0].strip() == word:
                if row_ctc[2].strip() == POSITIVE_SENTIMENT:
                    count_positive += 1
                    bFound = True
                elif row_ctc[2].strip() == NEGATIVE_SENTIMENT:
                    count_negative += 1
                    bFound = True
                elif row_ctc[2].strip() == NEUTRAL_SENTIMENT:
                    count_neutral += 1
                    bFound = True
                else:
                    bFound = False

        if (bFound == True):
            debug_sentiment_csv.writerow ([word + " $ in trading corpus/sentiment: $ " + str(row_ctc[2])])
            #print ([word + " $ in trading corpus/sentiment: $ " + str(row_ctc[2])])
```



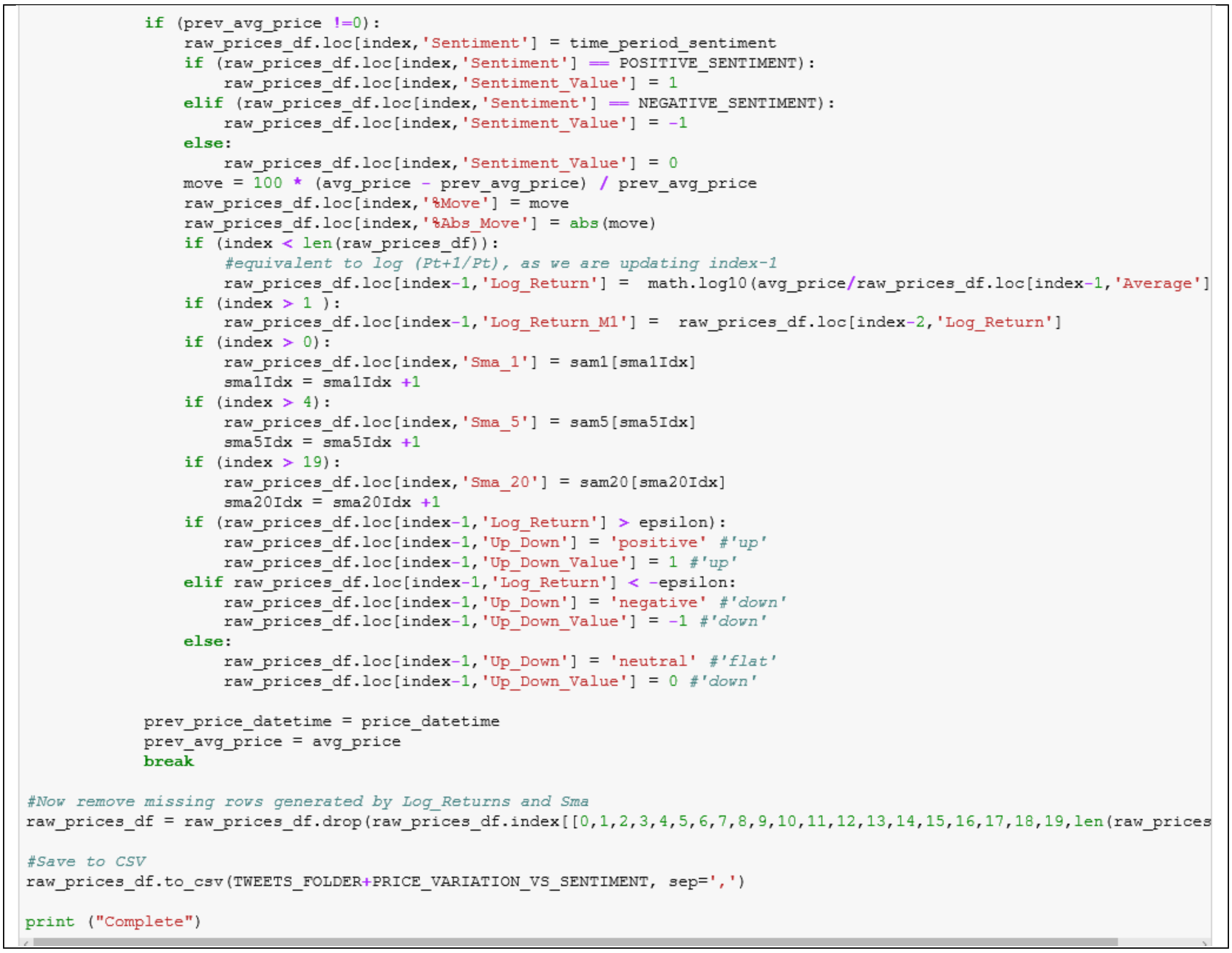
```
        else:
            #If it is not found in the custom traiding corpus,
            #then get the sentiment from the sentiwordnet corpus
            senti = swn.senti_synsets(word)
            senti_synsets = list(senti)
            if (len(senti_synsets) > 0):
                # Take the first sense, the most common
                senti_synset = senti_synsets[0]
                #when there is a case where the the word sentiment is proba is split 50/50,
                #the positive or negative sentiment is chosen
                if (senti_synset.pos_score() >= 0.5):
                    sentiment = POSITIVE_SENTIMENT
                    count_positive += 1
                elif (senti_synset.neg_score() >= 0.5):
                    sentiment = NEGATIVE_SENTIMENT
                    count_negative += 1
                else:
                    #This means that when a sentiwordnet have a positive (negative) =0.5 and neutral = 0.5,
                    #the neutral value is selected (this is a cautious approach)
                    sentiment = NEUTRAL_SENTIMENT
                    count_neutral += 1
                debug_sentiment_csv.writerow ([word + " $ in sentiwordnet corpus/sentiment: $ " + sentiment])
                #print ([word + " $ in sentiwordnet corpus/sentiment: $ " + sentiment])
            else:
                #If word not found, then set to neutral as default
                #score = 'neutral'
                sentiment = NEUTRAL_SENTIMENT
                count_neutral += 1
                debug_sentiment_csv.writerow ([word + " $ not present in any corpa/sentiment: $ " + sentiment])
                #print ([word + " $ not present in any corpa/sentiment: $ " + sentiment])
        #debug_sentiment_csv.writerow([word + " - count " + POSITIVE_SENTIMENT + ":"  + str(count_positive) +
        #              " - count " + NEGATIVE_SENTIMENT + ":" + str(count_negative) +
        #              " - count " + NEUTRAL_SENTIMENT + ":"  + str(count_neutral)] )

    sentiment_scores = {POSITIVE_SENTIMENT: count_positive,
                        NEUTRAL_SENTIMENT: count_neutral,
                        NEGATIVE_SENTIMENT: count_negative}
    debug_sentiment_csv.writerow(["sentiment_scores: " + str(sentiment_scores)])

    #Tweeter data analysis showed that there is a large skew towards neutral words in a bag of words.
    #In other words, most sentences contains a higher frequency of neutral words in a sentence than
    #positive or negative. Therefore, there is a need to re-balance this.

    #Therefore, the neutral category should only be taken into consideration if there is no positive(negative) sentiment
    count_positive = sentiment_scores.get(POSITIVE_SENTIMENT)
    count_neutral = sentiment_scores.get(NEUTRAL_SENTIMENT)
    count_negative = sentiment_scores.get(NEGATIVE_SENTIMENT)

    #When there is the same number of positive and negative word sentiments
    #=> the bag of words sentiment is neutral
    #When there is a greater counter of positive (negative) word sentiments
    #=> the bag of words sentiment is postive (negative)
    sentiment_bag_of_words = NEUTRAL_SENTIMENT
    if (count_positive > count_negative):
            sentiment_bag_of_words = POSITIVE_SENTIMENT
    elif (count_positive < count_negative):
            sentiment_bag_of_words = NEGATIVE_SENTIMENT

    #Each row contains the bags of word, the sentiment and the datetime stamp
    tweets_with_sentiments.append([row[0],sentiment_bag_of_words, row[1]])

    #Find the most frequent sentiment in a bag of words
    #sentiment_bag_of_words = max(sentiment_scores, key=sentiment_scores.get)
    debug_sentiment_csv.writerow (["*********************************"])
    debug_sentiment_csv.writerow (["***Analysis bag of words: " + str(row) + " / sentiment: " + sentiment_bag_of_words])
    debug_sentiment_csv.writerow (["*********************************"])
    #print("***Analysis bag of words: " + str(row) + " / sentiment: " + sentiment_bag_of_words)

print("")
print ("Complete")
```

Code Snippet 13 – Sentiment generation (c.f. *sentiment_analysis.ipynb*)

Appendix F – Generation of the base dataset



```python
#Generate the moving average for a period
def movingaverage (values, window):
    weights = np.repeat(1.0, window)/window
    sma = np.convolve(values, weights, 'valid')
    return sma

#Load prices in a dataframe
raw_prices_df = pd.read_csv(MKT_PRICE_FOLDER+MERGED_MKT_PRICES)

#Avg = (High + Low + Close)/3
raw_prices_df['Average']= raw_prices_df.apply(lambda row :
                          (row['High']+row['Low']+row['Close'])/3,
                          axis=1)
#Generate the Simple Moving Average
sam1 = movingaverage(raw_prices_df['Average'],1)
sam5 = movingaverage(raw_prices_df['Average'],5)
sam20 = movingaverage(raw_prices_df['Average'],20)
#Default new columns
raw_prices_df['%Move'] = 0
raw_prices_df['%Abs_Move'] = 0
raw_prices_df['Up_Down'] = 0
raw_prices_df['Log_Return'] = 0
raw_prices_df['Log_Return_M1'] = 0
raw_prices_df['Sma_1'] = 0
raw_prices_df['Sma_5'] = 0
raw_prices_df['Sma_20'] = 0
raw_prices_df['Up_Down_Value'] = -999
raw_prices_df['Sentiment_Value'] = -999
sma1Idx = 0
sma5Idx = 0
sma20Idx = 0
#Epsilon is use as a mechanism to generate more market neutral moves.
epsilon = 0.0001
tweet_row_index=0
prev_avg_price =0
prev_price_datetime = datetime.datetime.strptime("1970-01-01 00:00:00", DATE_TIME_FORMAT)
for index, price_row in raw_prices_df.iterrows():
    price_datetime = datetime.datetime.strptime(price_row['Date'], DATE_TIME_FORMAT)
    avg_price = price_row['Average']
    count_positive = 0
    count_negative = 0
    count_neutral = 0
    for tweet_row in tweets_with_sentiments[tweet_row_index:]:
        tweet_datetime = datetime.datetime.strptime(tweet_row[2], DATE_TIME_FORMAT)
        if ((tweet_datetime >= prev_price_datetime and tweet_datetime < price_datetime)):
            if tweet_row[1].strip() == POSITIVE_SENTIMENT:
                count_positive += 1
            elif tweet_row[1].strip() == NEGATIVE_SENTIMENT:
                count_negative += 1
            elif tweet_row[1].strip() == NEUTRAL_SENTIMENT:
                count_neutral += 1
            tweet_row_index +=1
        else:
            sentiment_scores = {POSITIVE_SENTIMENT: count_positive,
                                NEUTRAL_SENTIMENT: count_neutral,
                                NEGATIVE_SENTIMENT: count_negative}

            count_positive = sentiment_scores.get(POSITIVE_SENTIMENT)
            count_neutral = sentiment_scores.get(NEUTRAL_SENTIMENT)
            count_negative = sentiment_scores.get(NEGATIVE_SENTIMENT)

            time_period_sentiment = NEUTRAL_SENTIMENT
            if (count_positive > count_negative):
                    time_period_sentiment = POSITIVE_SENTIMENT
            elif (count_positive < count_negative):
                    time_period_sentiment = NEGATIVE_SENTIMENT
```



```
            if (prev_avg_price !=0):
                raw_prices_df.loc[index,'Sentiment'] = time_period_sentiment
                if (raw_prices_df.loc[index,'Sentiment'] == POSITIVE_SENTIMENT):
                    raw_prices_df.loc[index,'Sentiment_Value'] = 1
                elif (raw_prices_df.loc[index,'Sentiment'] == NEGATIVE_SENTIMENT):
                    raw_prices_df.loc[index,'Sentiment_Value'] = -1
                else:
                    raw_prices_df.loc[index,'Sentiment_Value'] = 0
                move = 100 * (avg_price - prev_avg_price) / prev_avg_price
                raw_prices_df.loc[index,'%Move'] = move
                raw_prices_df.loc[index,'%Abs_Move'] = abs(move)
                if (index < len(raw_prices_df)):
                    #equivalent to log (Pt+1/Pt), as we are updating index-1
                    raw_prices_df.loc[index-1,'Log_Return'] =  math.log10(avg_price/raw_prices_df.loc[index-1,'Average']
                if (index > 1 ):
                    raw_prices_df.loc[index-1,'Log_Return_M1'] =  raw_prices_df.loc[index-2,'Log_Return']
                if (index > 0):
                    raw_prices_df.loc[index,'Sma_1'] = sam1[sma1Idx]
                    sma1Idx = sma1Idx +1
                if (index > 4):
                    raw_prices_df.loc[index,'Sma_5'] = sam5[sma5Idx]
                    sma5Idx = sma5Idx +1
                if (index > 19):
                    raw_prices_df.loc[index,'Sma_20'] = sam20[sma20Idx]
                    sma20Idx = sma20Idx +1
                if (raw_prices_df.loc[index-1,'Log_Return'] > epsilon):
                    raw_prices_df.loc[index-1,'Up_Down'] = 'positive' #'up'
                    raw_prices_df.loc[index-1,'Up_Down_Value'] = 1 #'up'
                elif raw_prices_df.loc[index-1,'Log_Return'] < -epsilon:
                    raw_prices_df.loc[index-1,'Up_Down'] = 'negative' #'down'
                    raw_prices_df.loc[index-1,'Up_Down_Value'] = -1 #'down'
                else:
                    raw_prices_df.loc[index-1,'Up_Down'] = 'neutral' #'flat'
                    raw_prices_df.loc[index-1,'Up_Down_Value'] = 0 #'down'

            prev_price_datetime = price_datetime
            prev_avg_price = avg_price
            break

#Now remove missing rows generated by Log_Returns and Sma
raw_prices_df = raw_prices_df.drop(raw_prices_df.index[[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,len(raw_prices

#Save to CSV
raw_prices_df.to_csv(TWEETS_FOLDER+PRICE_VARIATION_VS_SENTIMENT, sep=',')

print ("Complete")
```

Code Snippet 14 – Base dataset generation (c.f. *sentiment_analysis.ipynb*)-