Jeffrey Levesque

Ist-565 | Final project

Can Social Media + Wikipedia Predict Stock Market?

IST-565

Jeff Levesque

Professor Gates

IST 565

**Introduction**

The stock market often intrigues those from fields such as economics, to other areas including social sciences. Questions that have been asked many times: can the stock market be predicted? These questions are many times too encompassing, leading some to lean in the direction of superstition. An amusing study in 1973 by a professor from Princeton University, found that monkeys can outperform seasoned financial analysts[[1]](#footnote-1). However, with the rise of data centers containing petabytes of customer tendencies, server logs, and social media, are analysts in a better position to answer such a question?

Data is now the new currency, and acquiring the right set, is often one of the bigger challenges for analysis. Without special partnerships accessing into financial databases, can available API’s with public data, be processed for modeling, then prediction? These are often regular challenges faced by analysts today.

A study by Derek Tsui, found that social media data from StockTwits coupled with the Yahoo financial api could generate between 50-60% accuracy[[2]](#footnote-2). However, the study is analogous to asking a single person to predict, versus larger group. In this study, a more unique dataset will be used for modelling. Specifically, can a set of keywords from twitter coupled with preference of categorical articles from Wikipedia, predict stock prices? Ultimately, the goal is to determine whether more unique sources of data, and API’s can be leveraged, to produce classification models.

**Analysis**

Data Preparation (Twitter):

The twitterscraper package from python was implemented to scrape keywords[[3]](#footnote-3):

* Apple
* Walmart
* amazon
* TheDemocrats
* GOP
* NFL
* NBA
* MLB
* NHL
* shakira
* Eminem
* rihanna
* justinbieber
* vindiesel
* WillSmithNews
* TheRock
* realjstrathamEminem

The selection process was two-folds. First, an attempt to cover influential businesses, politics, sports and music industry, as well as celebrities. The second intention was to have a close relation to the facebook api. However, due to time constraint, only the twitterscraper non-api package was used:

for tweet in query\_tweets(query, int(quantity)):

tweets.append({

'text': tweet.text,

'likes': tweet.likes,

'retweets': tweet.retweets,

'replies': tweet.replies,

'user': tweet.user,

'timestamp': str(tweet.timestamp)

})

if len(tweets):

with open(outfile, 'w') as file:

json.dump(tweets, file, indent=4)

The result of this process was 17 json files, for each keyword. Specifically, the scraper was assigned to scrape all related tweets for a three-year span.

Data Preparation (Wikipedia):

The wikipedia package from python was implemented[[4]](#footnote-4) to scrape the contents of the top 1000 articles for the span of three years:

r = requests.get(

'{}/metrics/pageviews/top/{}/all-access/{}'.format(

rest\_v1,

project,

date

)

)

top1000 = r.json()

articles = json.loads(r.text)['items'][0]['articles']

# report top 1000 article

with open(outfile, 'w') as jsonfile:

json.dump(top1000, jsonfile, indent=4)

This generated a folder with many json files, each representing the top 1000 wikipedia article for a given month. To build the train dataset, only the first json file, from the latter json.dump was implemented:

if use\_sample:

with open('data/2016-08-01--sample-train.json', 'r') as f:

articles = json.load(f)['items'][0]['articles']

articles = [a for a in articles if a['category'] != 'other']

for item in articles:

article = item['article']

repls = {':': '--colon--', '/': '--fslash--'}

filename = reduce(lambda a, kv: a.replace(\*kv), repls.items(), article)

filepath = 'data/wikipedia/articles/{}.txt'.format(filename)

try:

if not path.isfile(filepath):

search\_count[filename] = {}

with open(filepath, 'w') as txtfile:

# article content

summary = wikipedia.WikipediaPage(title=article).summary

Once again, a directory was created containing many text files, each containing only the content for the top 1000 wikipedia articles[[5]](#footnote-5). A manual process was required, to assign an article a corresponding category. Due time constraint, only the first 500 articles were manually inserted a category attribute:

"articles": [

{

"rank": 1,

"views": 65843652,

"article": "Main\_Page",

"category": "other"

},

{

"rank": 2,

"views": 2132787,

"article": "Special:Search",

"category": "other"

},

{

"rank": 3,

"views": 366380,

"article": "Harry\_Potter\_and\_the\_Cursed\_Child",

"category": "other"

}

]

The predetermined categories:

* tv\_movie
* celebrity
* music
* sports
* politics
* science + technology
* geography
* other

Feature Reduction:

An attempt was made to use the chi.squared method from the FSelector package as a means for feature reduction, and selection:

feature.set = chi.squared(

X.category ~ .,

df.merged

)

However, the results only removed one feature from the original df.merged dataframe, containing 15492 different words. Since the computing time for this process was significant, this implementation was omitted.

Preprocessing Twitter:

Twitter data was loaded into R using a custom load\_data function[[6]](#footnote-6). Specifically, each json file, containing tweets for a given keyword, was merged into a single dataframe. Then, each tweet within its corresponding dataframe column was converted to lowercase. This allowed for exploratory analysis:

tweets.unnested = df.twitter %>% unnest\_tokens(word, text)

tweets = tweets.unnested %>%

group\_by(timestamp) %>%

mutate(word\_count = 1:n()) %>%

inner\_join(get\_sentiments('bing')) %>%

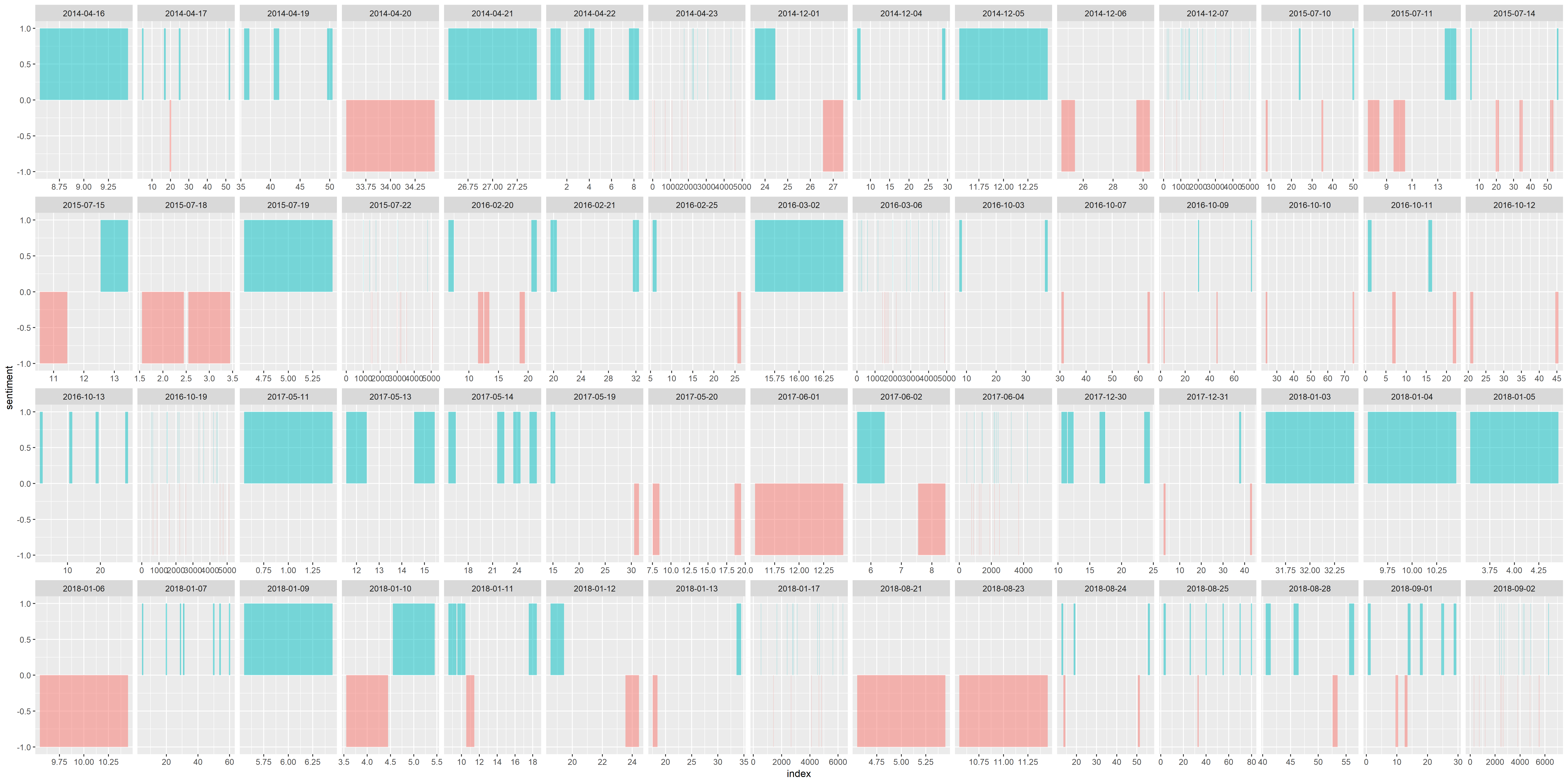
count(timestamp, index = word\_count, sentiment) %>%

ungroup() %>%

spread(sentiment, n, fill = 0) %>%

mutate(sentiment = positive - negative)

The associated visualization was busy:



However, is both informative on which days received more tweets compared to other reported days, as well as an indication that some days are missing. Specifically, some days contained units of thousand of tweets, other days only had single digit quantities. To further investigate why many days were missing from the result, additional keywords were appended, and the corresponding python twitterscraper was reimplemented:

* CNN
* FoxNews
* MSNBC

However, the generated visualization was similar, except days provided on the previous results, were missing. Likewise, some previous missing days were displayed. Results computed by R also indicated these results. This suggests that the corresponding python package did not return all tweets within the designated time. Due to time constraint, an alternative package was not investigated.

Preprocessing Wikipedia:

All analysis related to clustering and classification required each article text file to be loaded into memory. This was done through a custom module function load\_corpus[[7]](#footnote-7). This function generated a list of nonempty files, given a path parameter:

files = list.files(

source,

pattern = '.txt',

full.names = TRUE,

all.files = TRUE

)

f = file.info(files)

nonempty\_files = rownames(f[which(f$size > 0),])

Then, using the text2vec package, an iterator over the corresponding files, allowed a tfidf document matrix to be created:

it\_files = ifiles(nonempty\_files)

it\_tokens = itoken(

it\_files,

preprocessor = tolower,

tokenizer = word\_tokenizer,

progressbar = FALSE

)

vocab = create\_vocabulary(it\_tokens, stopwords = tm::stopwords('english'))

vectorizer = vocab\_vectorizer(vocab)

dtm = create\_dtm(it\_tokens, vectorizer)

model.tfidf = TfIdf$new()

dtm.tfidf = model.tfidf$fit\_transform(dtm)

return(dtm.tfidf)

If a corresponding article contained multiple rows of content, the corresponding document term matrix, would also have multiple rows of representation. This was reduced by aggregation:

df.merged = aggregate(

df.merged[, -c(

which(names(df.merged)=='article\_name')

)],

by = c(df.merged$article\_name),

na.rm = TRUE,

na.action = 0,

FUN = sum

)

Finally, the category was appended to the structure:

X.category = lapply(articles.nodupe, FUN = function(x) {

wikipedia.list[

which(wikipedia.list$article == gsub('.txt$', '', x)),

grep('^category$', colnames(wikipedia.list))

]

})

df.merged$X.category = X.category

The completed Wikipedia data representation allowed the sparse tfidf to be clustered in the absence of the category label, while allowing for classification techniques in its presence.

Support Vector Machines:

The support vector machine was trained against 2/3 of the collected Wikipedia article content. However, due to the limitation of the classifier requiring a comparison of many one versus all, this implementation was executed via python[[8]](#footnote-8):

clf = SVC(probability=True)

if test:

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X,

y,

test\_size=test\_size,

random\_state=42

)

clf.fit(X\_train, y\_train)

pred = clf.predict(X\_test)

Specifically, the python variant has the built-in ability to perform multi-classifiers. In the above implementation, the chi-squared determined the number of features to be implemented via the tfidf document term matrix:

ch2 = SelectKBest(chi2, k=k)

corpus = ch2.fit\_transform(X, y)

myTfidf = TfidfVectorizer()

myTfs = myTfidf.fit\_transform(corpus)

Naïve Bayes:

The naïve bayes classifier was run against 2/3 of the collected Wikipedia article content. Only the laplace function was used for tuning:

fit.nb = naive\_bayes(

as.factor(X.category) ~ .,

data=df.train,

laplace = 1

)

The intention was to assist smoothing across the model, since the supplied data was a sparse matrix.

Decision Tree:

The decision tree was executed on 2/3 of the collected Wikipedia article content:

fit.tree = rpart(

as.factor(X.category) ~ .,

data = df.train,

method = 'class'

)

Random Forest:

The random forest was executed on 2/3 of the collected Wikipedia article content:

fit.rf = randomForest(

as.factor(X.category) ~ .,

data = df.train,

ntree = 2

)

Originally, the number of trees was set to ntree = 30. However, the local machine was not able to allocate roughly 915MB of memory. Since the computing time was costly, this was reduced to ntree=2.

**Results**

During the early stages of collecting twitter data, it was found that the client based twitterscraper package was unable to fully scrape all tweets. Having this problem, along with the limitation of time to research alternative implementations, eliminated the social media aspect of the study. Instead, the problem adjusted, and attempted to predict whether web traffic for aggregated article categories could predict whether the stock market would fall or rise.

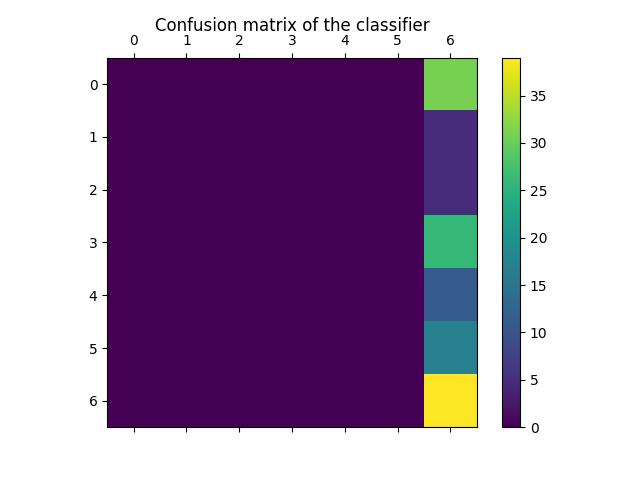
With the adjustment to the study, the first goal was to determine whether a classification method could model with enough accuracy an article category. If an adequate model could be generated, then it would be used to predict successive article categories for the full 3-year duration. This was the original duration of the collected twitter data.

Multiple classification algorithms were attempted. First, the support vector machine (svm) was implemented. Since the dimensionality of the given problem was extremely high, performing many one versus all classification would be difficult, even if streamlined into a loop. Specifically, the amount of memory required to generate thousands of svm models against a sparse matrix would not be possible on a local machine. As mentioned during the analysis, attempts to reduce dimensionality did not work with chi.squared. Therefore, an alternative was the use the sklearn package from python. Additionally, five different values were used with chi-squared – 5, 25, 50, 300, all features. Therefore, the feature selection returns the top n latter features. Then, the corresponding features were used for prediction:

clf.fit(X\_train, y\_train)

pred = clf.predict(X\_test)

Both the error rate, and confusion matrix were identical. Therefore, the selection of different k values, did not yield any improvements in modeling. The corresponding generated confusion matrix indicates that the model did not train well:



The second implementation used naïve bayes. This returned a 0.92 error. By chance, a category should predict at about 14%, if 7 different categories were used. Therefore, the error rate was roughly double random error. When assessing what could potentially have been improved, a proper dimension reduction technique should have been applied. However, an earlier attempt only reduced the dimensional space by one, given 15492 features. Since the runtime outweighed the benefits, this implementation was omitted for all classification techniques. The gain.ratio is an alternative to the chi.squared and could be tested.

Next, the decision tree had the longest runtime.

Finally, the random forest was used, with hopes of improved results, compared to the single decision tree. However, attempting to build a random forest of 30 trees, produced memory allocation error:

> ##

> ## random forest: instead of using 'method=class' like rpart, 'as.factor'

> ## is implemented directly on the formula.

> ##

> rf.start = Sys.time()

> fit.rf = randomForest(

+ as.factor(X.category) ~ .,

+ data = df.train,

+ ntree = 30

+ )

Error: cannot allocate vector of size 915.4 Mb

Since there was not enough memory, the ntree=2 was implemented:

> ##

> ## random forest: instead of using 'method=class' like rpart, 'as.factor'

> ## is implemented directly on the formula.

> ##

> rf.start = Sys.time()

> fit.rf = randomForest(

+ as.factor(X.category) ~ .,

+ data = df.train,

+ ntree = 2

+ )

Error in randomForest.default(m, y, ...) :

Can not handle categorical predictors with more than 53 categories.

> rf.end = Sys.time()

Unfortunately, since the chi.squared failed to reduce the dimension, the random forest could not be applied to the sparse document term matrix. Therefore, a result could not be generated.

When analyzing the results, the size of the dataset was not ideal for several reasons. First, having 500 labeled articles is not sufficient. For example, the 20 newsgroups dataset[[9]](#footnote-9) comprised of 20,000 documents. This is more than 40x the data used in this study. More generally, some of the collected articles, were not Wikipedia articles. Instead they were Wikipedia endpoints, including Main\_Page. Additionally, the distribution of articles into the predefined categories were very unbalanced. Some categories contained roughly 50 articles, while other would have 7 articles. Having 50 sparse records to train a category is prone to significant error. Therefore, any category containing less than 10 articles, will not be useful for the overall model.

Additionally, a consistent methodology should have been implemented when labelling the 500 records of the train data. Some discrepancies include classifying movie actors and actresses as celebrities, while musical artists were labeled as music. Other problems include classifying previous political family figures as political, whereas current figures are instead labeled as celebrities. These inconsistencies, coupled with the sparse limited data, can quickly lead to large errors found in this study.

**Conclusions**

The intention of this study was to discover novel markers to help predict whether the stock market would fall or rise. However, problems in data collection, proved the task more difficult than anticipated. This greatly reduced the capacity to implement algorithms. The question whether twitter, and Wikipedia traffic could predict stock market is indeed a novel problem. However, it was easily determined that a better data collection strategy is required.

Without having appropriate data, forcing data preprocessing can often yield empty, or an unbalanced distribution of features against potential labels. In the case of this study, only two different algorithms succeeded, each having poor results. Therefore, the next steps should not focus on how to produce better results. Instead, finding alternative scraping methods for twitter, as well as expanding social media sources, could help smooth results. For example, when considering the study performed by Derek Tsui, having a baseline of 50% using StockTwits is a good first approach. Then, the goal could be an attempt to append additional data, with ensembling techniques.

Overall, this study was difficult given such a broad general goal. Focusing on a smaller problem, then potentially build up to a larger general case, might be a better endeavor. For example, can an aggregated set of tweets, and Wikipedia articles on the top 100 nasdaq stocks, predict its corresponding stock values? Simplified problems may be a better initial approach, instead of trying to tackle such a large problem.

1. <https://www.forbes.com/sites/rickferri/2012/12/20/any-monkey-can-beat-the-market/#42e9b9ee630a> [↑](#footnote-ref-1)
2. Tsui, Derek. Predicting Stock Price Movement Using Social Media Analysis. Stanford University, 2016. [↑](#footnote-ref-2)
3. <https://github.com/jeff1evesque/ist-652/blob/master/utility/twitter_scraper.py> [↑](#footnote-ref-3)
4. <https://github.com/jeff1evesque/ist-652/blob/master/utility/wikipedia_scraper.py> [↑](#footnote-ref-4)
5. <https://github.com/jeff1evesque/ist-565/tree/master/data/wikipedia> [↑](#footnote-ref-5)
6. <https://github.com/jeff1evesque/ist-565/blob/master/packages/customUtility/R/load_package.R> [↑](#footnote-ref-6)
7. <https://github.com/jeff1evesque/ist-565/blob/master/packages/customUtility/R/load_corpus.R> [↑](#footnote-ref-7)
8. <https://github.com/jeff1evesque/ist-652/blob/master/run.py> [↑](#footnote-ref-8)
9. <http://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html#loading-the-20-newsgroups-dataset> [↑](#footnote-ref-9)