**Stock Trend Prediction \_Sentimental Analysis**

**ABSTRACT**

Stock trend prediction is an important and thriving topic in financial engineering especially since new techniques and approaches on this matter are gaining ground constantly.In the contemporary era, the ceaseless use of social media has reached unprecedented levels, which has led to the belief that the expressed public sentiment could be correlated with the behavior of stock prices. The idea is to recognize

Patterns which confirm this correlation and use them to predict the future behavior of the various stock prices. With no doubt, though uninteresting individually, tweets can provide a satisfactory reflection of public sentiment when taken in aggregate. In this paper, we develop a system which collects past tweets, processes them further, and examines the effectiveness of various machine learning techniques such as Naive Bayes Bernoulli classification and Support Vector Machine (SVM), for providing a positive or negative sentiment on the tweet corpus. Subsequently, we employ the same machine learning algorithms to analyze how tweets correlate with stock market price behavior. Finally, we examine our prediction's error by comparing our algorithm's outcome with next day's actual close price. Overall, the ultimate goal of this project is to forecast how the market will behave in the future via sentiment analysis on a set of tweets over

the past few days. The final results seem to be promising as we found correlation between sentiment of tweets and stock prices. Our selected company is Apple Inc. (aapl). We choose this stock mainly because it is popular and there is a large amount of tweets available.

**Introduction**

Modern data mining techniques have led to the development of sentiment analysis, an algorithmic approach for detecting the predominant sentiment about a product or company using social media data. A prominent field for the use of sentiment analysis has been stock market forecasting, Now a days, a great volume of data, which contains information about numerous topics, is being transmitted online through various social media. An excellent example is Twitter, where over 400 million tweets are sent daily. Though each tweet may not be significant as a unit, a large collection of them can provide data with valuable insight about the common opinion on a particular subject. Gauging the public's sentiment by retrieving online information from Twitter, can be valuable in forming trading strategies. The correct prediction about the fluctuation of stock prices depends on many factors, and public sentiment is arguably included.

**2. OUR MODEL**

In this paper, we mined tweets using Twitter's Search API and subsequently processed them for further analysis, which included Natural Language Processing (NLP) and Sentiment Analysis. Thereafter, we applied Naive Bayes and

SVM to predict each tweet's sentiment. After predicting every tweet's sentiment, we mined historical stock data using Yahoo finance API. We then created a respective feature matrix for stock market prediction using sentiment score and stock price's change for each day and at the end we proposed our own trading strategy.

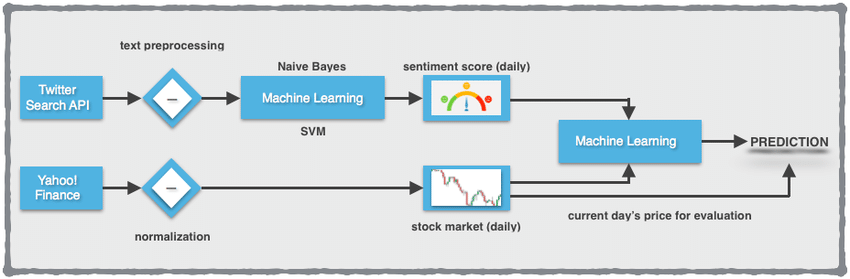
******

Figure 1: Flow diagram of Algorithmic model.

**4. DATASET**

We used two live fed datasets:

Stock Prices obtained using Yahoo! Finance API. This dataset consists of the Open, Close, High and Low values for each day. We also obtained a collection of tweets using Twitter's Search API. For each tweet these records provide a tweet id, the timestamp and tweet text, which is by design limited to 140 characters and needs to be altered from noise. Since we perform our prediction and analysis on a daily basis, we split the tweets by days using the timestamp as the main index of the dataframe.

**Twitter Data Preprocessing**

For the process of collecting tweets, Twitter provides two possible ways to gather Tweets: the Streaming API or the Search API. The Streaming API allows users to obtain real-time access to tweets from an input query. The user requests a connection to a stream of tweets from the server. Then, the server opens a streaming connection and tweets are streamed in as they occur, to the user. However, there are a few limitations of the Streaming API. First, language is not a special cable, resulting in a stream that contains Tweets of all languages, including a few non-Latin based alphabets, that complicates further analysis. Because of these issues, we decided to go with the Twitter Search API instead. The Search API is a REST API which allows users to request special queries of recent tweets. The Search API allows filtering based on language, region, geolocation and time. There is a rate limit associated with the query, but we handle it in the code.

The request returns a list of JSON objects that contain the tweets and their metadata. This includes a variety of

information, including username, time, location, retweets, and more. For our purposes, we mainly focus on the time

and tweet text. We filter out the unnecessary metadata and store both the tweet text and its timestamp in a .text.

We query the ticker of the company in front of which we add the dollar sign to gather the most financial" tweets.

Both of these APIs require the user to have an API key for authentication. Once authenticated, we were able to easilyaccess the API through a python library called Tweepy", which is a wrapper for the Twitter API.

**5. SENTIMENT ANALYSIS**

**Text Processing**

The text of each tweet includes a lot of words that are irrelevant to its sentiment. For example, some tweets contain URLS, tags to other users, or symbols that have no meaning. In order to better determine a tweet's sentiment

score, before anything else we had to exclude the noise. For this to happen,we relied on a variety of techniques using the Natural Language Toolkit (NLTK) for Python.

**A.Lowercase Conversion**

Converting all your data to **lowercase** helps in the process of preprocessing and in later stages in the **NLP** application, when you are doing parsing

**B Tokenization**

Firstly, we divided the text by spaces, thus forming a list of individual words per tweet. We then used each word in the tweet as features to train a classifier.

**C.Stemming**

**Stemming** is the process of reducing a word to its word stem that affixes to suffixes and prefixes or to the roots of words known as a lemma

**D. Removing Stopwords**

Next, we removed stopwords from the list of words. Python's Natural Language ToolKit library contains a stopword dictionary, which is a list of words that have neutral meaning and are inappropriate for sentiment analysis. To remove the stopwords from each text, we simply checked each word in the list of words against the dictionary. If a word was in the list, we excluded it from the tweet's text. The list of stopwords contains articles, some prepositions, and other words that add no sentiment value (able, also, or, etc.)

**E. Twitter Symbols**

It is not uncommon that tweets may contain extra symbols such as \@" or \#" as well as URLs. On Twitter, the word

following an \@" (mentions) symbol is always a username, which we also exclude because it adds no value at all to the text. Words following \#" (hashtags) are not filtered out be- cause they may contain crucial information about the tweet's sentiment. They are also particularly useful for categorization since Twitter creates new databases that are collections of similar tweets, by using hashtags. URLs are filtered out entirely, as they add no sentiment meaning to the text and could also be spams.

**Training Set Collection**

To train a sentiment analyzer and obtain data, we were in need of a system that could gather tweets. Therefore, we

first collected a large amount of tweets that would serve as training data for our sentiment analyzer.

In the beginning, we considered manually tagging tweets with a \positive" or \negative" label. Thus we created a

list of 1000 hand classified tweets but because it was hard and time-consuming, we decided to look for a database with already sentiment-classified tweets. Surprisingly, we found that our search for other tweet corpuses returned no results, as Twitter had recently modified its terms of service to disallow public hosting of old tweets. Under these circumstances,

we turned to alternative methods in order to form a training set. Specifically, we had two main ideas on how to classify tweets as training data.According to the first idea we created a \positive" and a \negative" dataset for training, by using Twitter's Search API. Each dataset was created programmatically and was based on positive and negative queries on emoticons and keywords:

fi Positive sentiment query: ":) :-) =) :D <3 like love"

fi Negative sentiment query: ":( =( hate dislike"

Any tweet that included one or more of these keywords or emoticons was most likely to be of that corresponding

sentiment. This resulted in a training set of \positive" and \negative"tweets which was almost as good as tagging tweets by hand.

**Training the classifiers**

Once we had collected a large tweet corpus as training data, we were able to construct and train a classifier. Within this project we used two types of classifiers: Naive Bayes Bernoulli and Support Vector Machine. We chose to focus

on these algorithms.

**A. Feature Extraction**

A unigram is simply a N-gram of size one. For each unique tokenized word in a tweet, a unigram feature is created for the classifier. For example, if a negative tweet contains the word \bad", a feature for classification would be whether or not a tweet contains the word \bad". Since the feature came from a negative tweet, the classifier would be more likely to classify other tweets containing the word \bad" as negative.

Likewise, a bigram is a N-gram of size two and a trigram is a N-gram of size three. That means that in the case of bigrams the feature vector for the classifier is made of two-word combinations and in the case of trigrams is made of three-word combinations respectively. For example, if a negative tweet contains the combination \not perfect", in the case of the bigram feature extraction it would be classified as a negative tweet. Instead, if only unigram features were used, the tweet would have been classified as positive since the term \not" has a neutral sentiment and the term \perfect" a positive one.

**B. Feature Filtering**

With the method described above, the feature set grows larger and larger as the dataset increases leading to the point where it becomes difficult and unnecessary to use every single unigram, bigram, and trigram as a feature to train our classifier. So we decided to use only the n most significant features for training. We used a chi-squared test, Pearson's chi-squared test in particular, to score each unigram, bigram, and trigram in our training set. NLTK helped us to determine the frequency of each feature. Having, now, the features ordered by score, we selected the top-10000 to use for training and classification. This method undeniably speeded up our classifiers and reduced the amount of memory used.



**6. MACHINE LEARNING TECHNIQUES**

Accurate classification continues to be an engaging problem in machine learning and data mining. It is more than

often that we need to create a classifier with a set of training data and labels. In our case, we want to build a classifier that is trained on our \positive" and \negative" labeled tweet corpus.This way, the classifier will be able to label future tweets as either "positive" or "negative", according to each tweet's characteristics and features. In this project,

we examine two classifiers used for text classification: Naïve Bayes Bernoulli and Support Vector Machines (SVM).In the following examples, y represents the class label, which in our case is either "positive" or "negative", and xi represents a feature in the feature set F.

**Naive Bayes Bernoulli**

Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes' theorem with the "naive"

assumption of independence between every pair of features.Given a class variable y and a dependent feature vector x1 through xn, Bayes' theorem states the following relationship:

P(yjx1; :::; xn) = P(y)P(x1;:::;xnjy)

P(x1;:::;xn

Using the naive independence assumption that

P(xiji; x1; : : : ; xi􀀀1; xi+1; : : : ; xn) = P(xiji)

for all i, this relationship is simplified to:

P(y j x1; : : : ; xn) = P(y)

Qn

i=1 P(xiji)

P(x1;:::;xn)

Since P(x1; : : : ; xn) is constant given the input, we can use

the following classification rule:

P(y j x1; : : : ; xn) / P(y)

Qn

i=1 P(xi j y)

+

^y = arg maxy P(y)

Qn

i=1 P(xi j y)

and we can use Maximum A Posteriori (MAP) estimation to estimate P(y) and P(xi j y); the former is then the relative

frequency of class y in the training set. The different naive

Bayes classifiers differ mainly by the assumptions they make regarding the distribution of P(xi j y).

In our case, we use Naive Bayes' Bernoulli implementation which is distributed according to multivariate Bernoulli

distributions. The decision rule for Bernoulli naive Bayes is based on

P(xi j y) = P(i j y)xi + (1 􀀀 P(i j y))(1 􀀀 xi)

which differs from multinomial NB's rule in that it explicitly penalizes the non-occurrence of a feature i that is an

indicator for classifier y.

The Naive Bayes Bernoulli classifier is extremely simple,and its conditional independence assumptions are not realistic in the real world. However, applications of Naive Bayes classifiers have performed well, better than initially imagined.

**Support Vector Machine**

Support vector machines (SVMs) are a set of supervised learning methods used for classification, regression and outliers detection. Unlike the Naive Bayes classifier, the SVM is a large margin classifier, rather than probabilistic. In previous works, SVMs have been shown to be very efiective for text categorization. The SVM is a classifier that attempts to find a separation between a linearly separable set of data,with as wide of a gap as possible between them, called a margin. A SVM constructs a hyper-plane or set of hyper-planes in a high or infinite dimensional space, which can be used for classification, regression or other tasks. Intuitively, a good separation is achieved by the hyper-plane that has the largest distance to the nearest training data points of any class (so-called functional margin), since in general the larger the margin the lower the generalization error of the classifier. With our training set as an input, the SVM finds the hyperplane such that each point is correctly classified and the hyperplane is maximally far from the closest points.The name "support vector" comes from points on the margin between the hyperplane and the nearest data points, which are called support vectors. The SVM looks for a parameter vector that, again, maximizes the distance between the hyperplane and every training point. In essence, it is an optimization problem:

Minimize 1

2a fi a

y(a fi xi + b) >= 1 ,

where y is the class label (􀀀1; 1) for negative and positive.Once the SVM is built, classification of new tweets simply

involves determining which side of the hyperplane that they fall on. In our case, there are only two classes, so there is

no need to go to a non-linear classifier. Figure 3 displays the sentiment classification of all gathered tweets, concerning seven of the most popular tech companies, from 14/6 until 22/6 using the following formula to normalize their positive sentiment score over the total number of tweets per day:

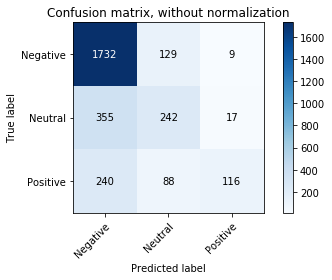
Normalized Positive Score = Positive Tweets

Total Tweets

**Classifier Evaluation**

In order to accomplish our ultimate goal, which was to utilize Twitter sentiment scores for predicting stock market's movements, we trained and tested each of our classifiers on a particular subset of our tweet corpus. So as to calculate accuracy, we used a confusion Matrix .

.



**STOCK MARKET PREDICTION**

After training our classifier, we decided that an interesting application would be to look at the correlation between

tweet sentiment and stock market prices. After we completed the Data Preprocessing, both in Twitter

data and in stock data, as described in above, we created a feature matrix which contains the following features:

- percentage positive sentiment score

- percentage negative sentiment score

- percentage neutral sentiment score

- close price

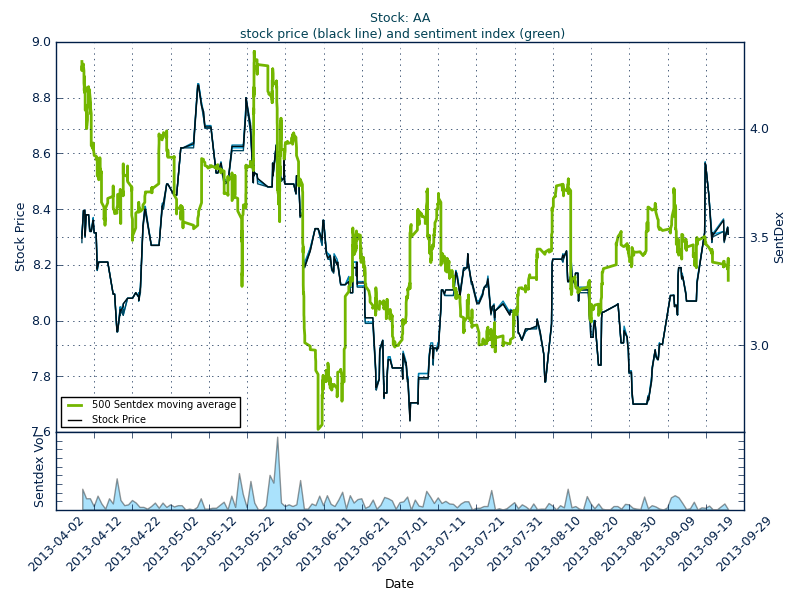
- HPLCT

- PCT fi change

- volume

Then we applied SVM in order to achieve a prediction about future stock movements. Afterwards, we compared

our results with the upcoming stock changes.



**CONCLUSIONS AND FUTURE WORK**

In this paper, we investigated whether public sentiment,as measured from tweets, is correlated or even predictive of stock values Our results show that changes in the public sentiment can affect the stock market. That means that we can indeed predict the stock market with high chances.

Furthermore, it is worth mentioning that our analysis does not take into consideration many factors. First of all, our dataset does not really extract the real public sentiment, it only considers twitter using, english speaking people. Secondly, the bigger the dataset is, the better the prediction but at the same time the problem gets more complicated.

There are many areas in which this work could be expanded in the future. With a longer period of time and more resources, there is much potential in the field. If it is possible, we would want to collect data over the course of a few years, both from Twitter and the stock market. In addition we could investigate intraday stock changes in order to make our prediction more accurate. Finally, in the future we could create a stock lexicon based on the most common words used.