Seyed Reza Mir Ghaderi
Nima Soltani

Forecasting Stock Market Behavior Based on Public Sentiment

Project Overview

It is believed that public sentiment is correlated with the behavior of the stock market [1]. The general objective of the project is to characterize this correlation and use it to predict the future behavior of the market. We assume that people express their mood in their Twitter posts (tweets) and approach the problem by performing large-scale analysis on these tweets. The ultimate goal of this project is to predict how the market will behave tomorrow given a large set of tweets over the past few days.

Previous work by Bollen et al. [1] uses sentiment analysis tools in conjunction with a non-linear model (Self-organizing Fuzzy Neural Network) to predict the changes in Dow Jones Industrial Average (DJIA). Here we design and build the learning and prediction system making only basic assumptions about the relationship between tweets and the market. We do not assume that the moods of the tweets are their only important feature, and instead look for informative features that might not be specifically mood-related. The reason for doing so is to give enough freedom to the algorithm itself to determine which words are most pertinent to the stock values.

Data

When starting this project, we had data for 6 months in 2009. This included data obtained from Twitter over the course of the 6 months as well as daily stock values.

Tweets

We had training data in the form of ~60GB of Twitter posts [2] over the timespan of June to December of 2009. Given the large volume of data, we needed to format the data into a consistent format. We first tried using Stanford's NLP Java application to parse the documents and get the data in a readable format. The advantages of this approach was that it was already written and debugged, as well as containing features like grouping words together by their root word. The main disadvantage of this approach, however, was that it took much more time and space to run than we could allow in our large-scale problem.

Thus we decided to write our own parser/tokenizer. By use of regular expressions in Python, we filtered out tweets with non-English letters, tokenized URLs, numbers, twitter usernames, and emoticons, converted everything to lowercase and removed all punctuation. We then ran a Python stemming tool (stemming 1.0) to remove the suffixes and attempt to find the root word of the words in the tweets. Also, since each tweet has a 140 character maximum, we decided to make each tweet 50 words long, where we truncated the tweet if it contained more than 50 words and padded the tweet with NULL words if it contained less. We ran this script on all our data, training and test data, so we can read words with consistent formatting.

Stocks

For all the learning algorithms except linear regression, we used daily open/close values of Dow Jones Industrial Average (DJIA) from June 12 2009-Dec 30 2009. For the linear regression we used DJIA hourly values for the same period (obtained from Price-Data). We tried different labeling definitions to form the classification problems. In particular for one bit representation of the state of the stock market following day t we tried

$$y = 1\{Openning(t+1) > Closing(t)\}$$

$$y = 1\{Closing(t+1) > Closing(t)\}$$

$$y = 1\{Closing(t+1) > Openning(t+1)\}$$

Similar approach was tried for the growth computation in Match/Score algorithm.

Feature Selection

One of the main differences between our work and the original work [1] is that we try to generalize the feature set. The original feature set used 7 features from two sentiment analysis tools: a general positive/negative classification, and mood states in 6 dimensions of calm, alert, sure, vital, kind and happy. Also, they took out tweets that did not explicitly express the author's mood. Our goal in this project, however, was to remove these constraints and allow the algorithm to select the most informative content in the day.

Clustering

To select the features, we found the statistics of the words appearing every hour. Despite having tens of gigabytes worth of tweets, we realistically had only 100 training data points, so we could not confidently train a hypothesis with more than 5-10 input features. In order to make good use of the data, we then used a k-means clustering algorithm to find words that varied consistently with each other so as to maximize the number of useful words to use as features.

For the clustering, we tried clustering based two different sets of features: (1) the ratio of the hourly word count to the total hourly word count, or (2) the ratio of the hourly word count to the total count of the word across all time. Running the clustering algorithm on these two sets produced fundamentally different features. The clusters using (1) lumped words of nearly equal frequency together. This did a good job of clustering words that are extremely common and have seemingly no correlation with the market, such as "the", "of", "and", etc. The clusters using (2) lumped words that fluctuated similarly together, regardless of the number of times they appeared each hour. The common words were still mainly lumped together, but there were some mixed in other sets. It was also interesting to see that in both cases, the algorithm assigned Spanish words to their own cluster. After coming up with 100 clusters, for each cluster C_i we then calculated the mutual information as $I(1\{\#C_i/n > E(\#C_i/n)\}; y_i)$, where n is the number of words that day. We then sorted them according to mutual information and took the 8 most informative clusters.

Surprisingly, this method of calculation only made a 1% change in the performance of our algorithm. In retrospect, the clustering strategy should have somehow considered the mutual information more explicitly. Also, one problem with this method was that it was almost chaotic. Any small perturbation in the algorithm parameters changed the resulting clusters. Clustering based on a more robust feature of the word also could have helped.

Market Behavior Learning/Inference Models

The inputs to the algorithms were counters of clusters with daily and hourly resolutions X_{day}, X_{hour}, and the stock values/indicators. In all the models we used the tweets of one day to predict the stock behavior on the next day. In all the algorithms, we used 70% of data (tweets and DJIA up/down indicator) for training and the rest for testing. We tried 4 different algorithms: Naïve Bayes, SVM, Linear Regression and our own heuristic algorithm Score/Match.

Naïve Bayes Learning

Given its fast running time, Naïve Bayes was our best choice as the baseline algorithm. We predicted the probability of each cluster given each label and the label prior as

$$\mathrm{P}(\text{Cluster } i \mid y=b) = \frac{\sum_{t=1}^{NumTrainingDays} 1\{y_t = b\} X_{day}(t,i)}{\sum_{t=1}^{NumTrainingDays} 1\{y_t = b\} \sum_{j=1}^{NumClusters} X_{day}(t,j)}$$

$$P(y=b) = \frac{\sum_{t=1}^{NumTrainingDays} 1\{y_t = b\}}{NumTrainingDays}$$

and used the following rule to predict the days in testing data

$$\hat{y}_t = 1\left\{ \frac{P(y=1) \prod_{i=1}^{NumClusters} P(\text{Cluster } i \mid y=1)^{X_{day}(t,i)}}{P(y=0) \prod_{i=1}^{NumClusters} P(\text{Cluster } i \mid y=0)^{X_{day}(t,i)}} \geq 1 \right\}$$

Given the large occurrence of each word and each class (up/down) we did not need ant smoothing. Given the relatively large level of error (see the error table) and the strong assumptions made in Naïve Bayes, we decided to apply a discriminative algorithm as the next step.

Support Vector Machine

The feature vectors we used for SVM were normalized to give the relative occurrence of each cluster in a day. We used cvx for convex optimization and set the regularization factor $C = 1$. To

avoid overfitting we turned the real frequency vector to a binary vector showing whether each frequency is higher or lower than its average value over all the training days. O can see slight improvement (see the error table) as compared to Naïve Bayes. However, the error was still large. We decided that problem may come from is the loss of valuable information by discretizing the stock value. Therefore we used a linear regression model.

Linear Regression

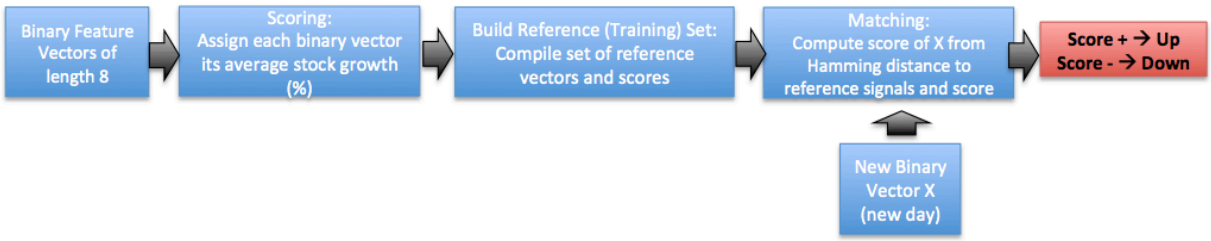
We tried to predict the value of the stock by running a linear regression on

- The stock value in the past few hours
- The average stock value in the past day
- The hourly count of clusters in the tweet data of the past 24 hours

We considered predicting the afternoon stock value (to have data for the past few hours). The result indicated that with the size of data we had, linear regression was prone to overfitting. While the training error was very low, we got orders of magnitude higher testing error.

Our Own Heuristic: Score/Match Algorithm

As explained in the SVM section, we can obtain a binary feature vector by discretizing the frequency vector. Doing so, we get binary vectors of length 8. To each vector we assigned a score equal to the stock growth following that vector (following that day). If one vector happened multiple times in our data we assigned the average of the stock growth (negative if decline) values. We then labeled the testing sequences by comparing their feature vector to the ones in the training set, finding the ones with minimum Hamming distance and finally comparing the average score values of the closest vectors to 0 (see the diagram). This algorithm (which was inspired by minimum Hamming distance decoding in communication) lead to an error which was lower than all the other classification algorithms.



Results

Naïve Bayes	Linear SVM	Linear Regression	Score/Match
Training Error38% Test Error 42%	Training Error 40% Test Error 40%	Training MSE 161 Test MSE 140000	Test Error 37.5%

Conclusions

Although we did not meet the performance of the original paper, we were solving a slightly different problem. The result of 37.5% accuracy is quite good considering the generality of the assumptions we were making and the limited training data that we had (the original paper had 9 months' worth of training data).

Future Improvements

There is room for improvement and we are interested in pursuing research on this topic. The following are potential methods of improvement that we plan to use:

Retrain with a sliding window

Due to the scarcity of data in our project, we tried to maximize the amount of data we could use for training, but one problem was that we did not update our training data once we tested it. Testing on a finite window of the past could provide a more realistic dataset.

Use pairs of words as features

The mutual information will increase even more with the joint distribution of the words in the cluster as opposed to the distribution of the union of the words. Intuitively, this should correspond more to a contextual clustering of the words, as opposed to purely coincidental.

Use a different classification structure

The current system only considers the change as an up/down quantity and does not distinguish a 1% change in the market from a 10% change. We could make a ternary system where we add an "insignificant change" level, corresponding to a -1% to 1% change. Alternatively we could still use a binary classification but only train using data points corresponding to large changes.

Acknowledgements

We would like to sincerely thank Ali Reza Sharafat, who helped with the initial bring-up of the project and provided advice on Python coding later on after he withdrew from the course. We would also like to thank Mihai Surdeanu and John Bauer for proposing the topic.

Works Cited

[1] J. Bollen, H. Maoa and X. Zeng, "Twitter mood predicts the stock market," *Journal of Computational Science,* pp. 1-8, 2011.

[2] J. Yang and J. Leskovec, "Patterns of Temporal Variation in Online Media," in *ACM International Conference on Web Search and Data Minig (WSDM)*, Hong Kong, 2011.