Applying Machine Learning to Stock Market Trading Bryce Taylor

Abstract: In an effort to emulate human investors who read publicly available materials in order to make decisions about their investments, I write a machine learning algorithm to read headlines from financial news magazines and make predictions on the directional change of stock prices after a moderate-length time interval. Using techniques that do not attempt to parse actual meaning from headlines, I am able to approximate the overall market trends to a reasonable degree, but not well enough to make money in times of small growth / decline. Analysis indicates that features are present in the data to make use of headlines for an algorithm, but due to an abundance of extra noisy features we have not yet been able to determine precisely what these features are. I tried using natural language processing to produce better features, but the best processors available were not able to interpret the densely worded headlines well enough to be of use.

Introduction / Motivation:

Twitter provides a standard source of data to analyze the sentiment of the public and has been used by many authors to attempt to predict stock market changes. I believe that a down-fall of these methods is that they do not directly attempt to evaluate the value of a stock and can only be applied to large, well-known companies and wish to instead develop an algorithm that can be used to determine how well a company will do based on what the company actually does. Motivating examples of this are stock price reactions to events such as changes management and major acquisitions. This would allow our algorithm to mimic the thought process of successful investors. I thus analyze news headlines related to the company to determine whether the headlines indicate positive news for the company. Using news sources from more informed writers should provide much more dense information than sources that read from public opinion like twitter. Since news articles are released intermittently, I wish to design an algorithm that can make a prediction based on a single headline since if it has to wait for multiple headlines, the market will have already reacted to the first headline.

Data Sources:

I used two types of data to implement this algorithm: headlines from financial analysts and historic stock prices. Headlines were manually collected from the Seeking Alpha website by performing a search for the stock symbols of each of the companies, then parsing the results with a custom Java program. Historic stock prices were taken from the official NASDAQ website. In total, over 12000 headlines were available taken from the past 7 years (the furthest back the Seeking Alpha website provided articles) from 7 different companies. The companies were:

The headlines were then tokenized using the Porter Stemming process (with the exception of having a token reserved for the stock ticker symbol of each company used) to produce a set of features corresponding to the list of symbols that appeared in the headline and the company about which the headline was written. Feature vectors for headlines were condensed into matrix form and written to matlab-readable files for portions of the project involving SVMs and PCA.

In order to simulate having to make decisions on a real life stock market, the data was divided 70 / 30 into training and testing data,

sorted by the time at which the articles were published so that all testing data occurred chronologically after all training data. I initially used randomized selection of the testing / training data, but found that this was an unrealistic model because any algorithm would then know how well each company did on average over the testing period since it was the same as the training period, giving it unreasonable hindsight.

Research Questions:

For each of the algorithms, the goal was to answer a question of the form:

"Given a headline released today about some company X, will the stock price of X rise by more than P percent over the next time period T?"

As baselines, we compared our algorithm to two simple algorithms: always responding yes and always responding no. A third algorithm was also used for comparison that took the best of these two results, retrospectively choosing the better of those two algorithms as its algorithm (note that this algorithm can never have an error

rate above 50%). This algorithm represents following the general market trend, buying in an up time and shorting in a down time.

Based on research by Eugene F. Fama, Lars Peter Hansen and Robert J. Shiller for their Nobel prize-winning paper in Economics, I chose T to be 3 months, in particular much longer than a week (information taken from Nobel Prize press release:http://www.nobelprize.org/nobel_prizes/ec onomic-sciences/laureates/2013/press.html). This is because their paper indicates it is impossible to predict prices based on public information in the short term. Longer time periods were considered, but due to only having access to 7 years of data and making predictions based on a single headline at a time, I decided to use T=3months. P was varied over different trials; a large value of P indicates looking for the most extreme increases in price (i.e. a chance to make the most money).

Bayesian Classifier:

For my initial tests, I chose to use a simple multinomial Bayesian classifier that analyzed

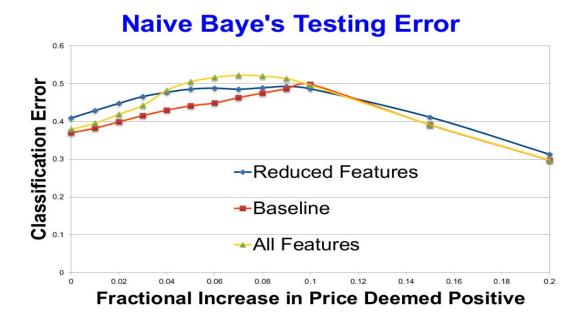


Figure 1

headlines based on the presence of each token in the headline. Since there were a total of 51,208 tokens, this resulted in a lot of tokens being used infrequently (and thus having inaccurate measurements of the probabilities used in the Bayesian model), even after simple Laplace smoothing. As a second test, I removed any token that did not occur at least 10 times in the headlines to create a second set of 693 features for use in the algorithm. The results of running Naive Baye's are shown as P ranges from 0 to 0.2 in Figure 1 on the previous page. Using all of the features, the error was sometimes above 0.5, which is unacceptable for most trading algorithms and was generally outperformed by the algorithm with reduced features since that algorithm never had testing error above 0.5. Neither algorithm was able to beat the hardest baseline, "best of two" algorithms, but the reduced feature algorithm followed it reasonably well, possibly well enough to use on real markets.

Using data collected from the first run of my Bayesian classifier, Table 1 shows the top 5 most indicative symbols (that occurred at least 10 times in the headlines) on both ends of the spectrum for classifications run with P=0. Some of the symbols are not surprising; discussion of a "split" likely means the stock price will drop in half if the split happens which would be marked negative by our algorithm (even though it is not actually bad for investors); similarly tokens like tough and event are logical since they indicate difficult times for the company and the company being involved with events, which is usually deemed positive. This bodes well for the general problem of extracting investing advice from headlines.

In an attempt to improve accuracy even further, I tried selecting only the top 20 most

indicative tokens on both ends of the spectrum and rerunning the classifier. However, it

Symbol	Positive probability to negative probability ratio	
Buzz	0.08136	
Split	0.1914	
Mini	0.2169	
Gross	0.2169	
Tough	0.2169	
Carrier	3.905	
Touch	3.905	
Effort	3.905	
Event	5.857	
Affect	6.508	

Table 1

performed significantly worse than using 693 tokens or using all of the available tokens, so we likely removed important features from the headlines.

Precision / Recall Analysis

As a quick check, I computed the error on positive headlines and the error on negative headlines as P varied (varying P varies the proportion of headlines that are actually positive or negative). The positive error was strictly increasing (0.15 to 0.93) while the negative error was strictly decreasing (0.85 to 0.052, graphs omitted to save space). These numbers are better than the precision / recall which would be given by the "best of two" algorithm (always resulting in one of the errors being 1), but since they follow the proportion of headlines marked positive, it means that is one of the most indicative features in my model (as opposed to the headlines themselves).

Support Vector Machines:

After converting the data to Matlab format, I trained Support Vector Machines (SVMs) on the data for all of the same trials as the Bayesian classifier. Unfortunately, Matlab was unable to process the full data set (12K headlines with 50K features each), so I only tested it on the reduced feature data set and the minimal feature data set. The results were almost identical to the Bayesian classifier regardless as to which type of SVM was used (polynomial, linear, etc) and roughly approximated the best baseline solution, but did not match or beat it.

Principal Component Analysis:

In order to determine how useful the features I had actually were, I ran principal component analysis on the data and then tested linear SVMs on several of the top principal components. Matlab's built-in principal component function was used to identify the principal components and the data with all 693 features was then projected onto this space to create new training and testing data for the SVM. Table 2 on this page shows the error rates as a function of how many of the top principal components were used.

There are several things to take away from these results: first, using a small number of the top principal components performs nearly as well as the baseline and slightly better than Naive Baye's (given a testing size of ~3500, these are likely a statistically significant differences). Thus, some portion of the data does hold information significant to predicting stock prices. Furthermore, adding more features does not reduce the error rate further and in fact increases the error rate by a significant amount. This means that many of the "features" (those that contribute

to the non-top principal components) are effectively noise for the purposes of satisfying our hypotheses.

Number Components	Error
1	0.5127
2	0.4045
3	0.3982
4	0.3982
5	0.3982
6	0.3939
7	0.4172
8	0.4049
9	0.4038
10	0.4133
50	0.4175
200	0.4503
693 (MAX)	0.4767
BAYES	0.4091
BASELINE	0.3859

Table 2

Manual Keyword Selection

Another method I tried for feature selection was to manually select features human insight indicates should be indicative. The key words corresponding to these tokens are shown below in Table 3 with their frequency in positively and negatively classified headlines; few of these stand out as strong indicators (not stronger than for example the average number of times stocks rose on the training period). Training on these symbols alone meant that few of the headlines could be processed total (~1300) and resulted in an error of 0.38 when classifying with P=0, notably worse than the baseline error of 0.33 on the same set.

Symbol	Positive	Negative	Ratio
	Frequency	Frequency	
Buy	0.217	0.182	1.193
Sell	0.064	0.068	0.953
Good	0.079	0.074	1.065
Growth	0.158	0.165	0.956
Beat	0.068	0.072	0.940
New	0.253	0.254	0.997
Stop	0.018	0.008	2.179
Disappoint	0.004	0.013	0.323
Win	0.028	0.027	1.006
Bullish	0.048	0.042	1.138
Risk	0.028	0.057	0.484
Bad	0.035	0.036	0.968

Table 3

Natural Language Processing:

After failing to beat our baseline algorithm using any of the standard features selection techniques, I decided to try using Natural Language Processing to provide better features for the learning algorithms. In particular, the best feature would be an indication of whether or not the article headline spoke of the company positively or negatively. This is known as sentiment analysis in natural language processing. Stanford has a publicly available Natural Language Processing Toolkit that provides sentiment analysis to sentences with high accuracy (>80%). Unfortunately, when tested on the headlines (re-parsed into sentence-like format) , the processor was unable to achieve high success rates for non-neutral sentences when compared to my own reading of the headline. In a random

sample of 20 headlines the processor deemed non-neutral (i.e. positive or negative sentiment), I agreed with exactly half (10) of the assessments. This is likely due to the fact that headlines are very short, often requiring complex parsing in order to fit them into a small space and thus do not really resemble the sentences the processor was trained in (which used normal, full-text sentences). Natural language processors would need to be specifically tailored to processing headline-like data to be able to make a meaningful contribution towards answering my research questions.

Conclusion:

Overall, my work indicates that a sophisticated model able to beat overall market trends by reading financial news headlines cannot be easily found without fairly sophisticated human-like processing of the headlines. However, using my work, one would be able to run an algorithm that does notably better than random chance and can do better than not investing at all by roughly following both positive and negative market trends.

Future Work:

After working on this project all quarter, there are a couple of ways that it might be extended to better answer the question:

- Customize natural language processing tools to work well with headlines
- Make predictions based on more than one headline at a time; this may allow for better long-term analysis of a company's potential