6.867 Final Project: Predicting Stock Trends Using Financial News Articles

Keren Gu KGU@MIT.EDU
Ernest Zeidman ZEIDMAN@MIT.EDU

1. Introduction

There have been many efforts to apply machine learning to the forecasting of financial time series data. For example it is possible to apply support vector machines (SVMs) to historical prices to forecast price movement; In fact, SVMs perform this task better than back propagated neural networks [1]. There have also been more varied approached, including the application of microblogging signals. It is possible to predict the direction of the NASDAQ by analyzing 1% of all tweets [2]. Alternatively its possible to make weak predictions about a particular stock using the feed of tweets mentioning the company name or ticker [3].

Our aim is to investigate how news articles can be used to predict the direction of stock movement. We investigate the predictive power of one article at a time, asking whether an article about company X released today can be used to forecast whether stock X will go up or down during the next day. In spirit, the problem is about sentiment analysis. It is based on the idea that investors will make trading decisions based on the tone and content of the news they read. However, the sentiment analysis is implicit because our labels (stock price changes) don't directly describe the article sentiment. The main goal of the project is to find a support vector machine, which performs this classification task. We also look into two alternative approaches, using regression and using a naive Bayes classifier.

It is worth noting that predictive power is not the full story in financial forecasting. Consider the trading strategy which buys when the classifier forecasts a positive movement and shorts when the classifier forecasts a negative movement. Such a strategy can have a negative expected value even if it makes money most days (e.g. if when it is wrong, it is terribly wrong).

All scripts, data-files, experiments, and results of this project can be found at this repository: `http://bit.ly/1eTPikL` .

2. Approaches

2.1. Data Collection

We collected two types of data for this project: articles and corresponding labels.

We first selected a set of 31 largest technology companies from NASDAQ and collected all news article by scraping each company's 'news' section on Google Finance. The available set of news articles span exactly 2 years into the past with majority of the articles published from the past month. We selected all of our articles from Reuter's Key Development in order to keep our articles uniformly distributed over the past 2 years. In total, we collected 1737 articles for 31 companies over 2 years.

In order to generate labels for the obtained articles, we looked at the prices for the company of interest around the time the article is published, under the assumption that the sentiment of the article is reflected in the stock's price in the near future. While only daily prices are available on the web, the set of prices that we collected contains the closing price on the day of publishing, the opening price on the next day, and the closing price of the next day. Together, these daily prices form 3 sets of labels: (i) after-day-label, defined as the after-hours price change; (ii) next-day-label, defined as the price change on the day after the article was published; and (iii) combined-label, defined as the sum of the after-day-label and next-day-label.

In the case where multiple articles are collected on the same day for the same company, we merge the feature vectors of these articles into one. Because with labels representing the price-change over a day, we have no way of distinguishing the fraction of price change that is caused by one article over another. After merging, we have 1477 articles with distinct company and date combination.

Lastly, the type of content collected is mainly announcements, summaries of news reports on company earning or dividends. All articles are very formal and highly objective. The following is an excerpt from a

random article:

"International Business Machines Corp announced that Australian fruit and vegetable retailer Harris Farm Markets has selected an IBM Flex System solution to help support its business growth and reduce IT costs and complexity."

2.2. SVM

The main approach to the classification problem is the application of a support vector machine. A SVM finds a hyperplane which bisects the input space into two halves corresponding with two classes. This hyperplane has the maximal margin, meaning it is as far as possible from all training points. If the sample points aren't linearly separable, it may be necessary to project them into a feature space. The radial basis kernel, which we employ, generates a feature space which is always linearly separable.

Stock price movements are characteristically noisy and erratic, meaning that our labels have a fair measure of noise. Basic SVMs can be very fragile and thrown off by a single mislabeled point. To mitigate this we employ slack variables, which allow some of the training points to slide into margin or even to be misclassified. Consequently our SVM has two parameters which need to be tuned: C governs the relative penalty given to the use of slack variables for each point, ξ_i, which is apparent in the primal form of the optimization problem:

$$\min \frac{1}{2}\|\theta\|^2 + C\sum_i \xi_i$$

The other parameter, β, comes into play when using the radial basis kernel. β governs the relative "distance" between points. For the major part of our project, we employed the radial basis kernel:

$$K(x, x') = \exp\big(-\frac{\beta}{2}||x - x'||^2\big),\ \beta > 0$$

Larger β means that even adjacent points are treated as if they are unrelated.

We used MATLAB's implementation of the `trainsvm` and `svmclassify`. MATLAB's `trainsvm` uses the Sequential Minimal Optimization (SMO) method as its default method to identify the support vectors s_i, weights α_i, and bias b that are used to classify a vector x according to the following equation:

$$\text{sign}\big\{\sum_i \alpha_i K(s_i, x) + b\big\}$$

where $K(x, x')$ is the kernel function of choice. We also experimented with the Quadratic Programming method, which is documented to give more precise solutions, though slower in performance.

`trainsvm` also allows us to set the tolerance with which the Karush-Kuhn-Tucker (KKT) condition is checked when using the SMO method. This parameter also serves as a restraint on the minimal value of C that can be set. The default value is set as 10^{-3}. We experimented a little with this parameter when trying out the linear kernel on one of our feature variants. In addition, `trainsvm` also allows us to set the percentage of dataset that is allowed to violate the KKT condition. We kept this parameter as 0.

To compare and verify the setup of MATLAB's `trainsvm`, we trained and compared results against the implementation from Problem Set 6.

2.3. Regression

We applied our regression implementation from Problem Set 3 to our dataset to investigate whether we can extract useful information from the magnitude of price changes (the SVM only considers the sign of the price changes). Since the feature space is so large, we employ regression with regularization, optimizing:

$$\frac{\lambda}{2}\|\theta\|^2 + \frac{1}{2}\sum_i (y_i - \theta \cdot \phi(x_i) - \theta_0)^2$$

The regression model gives more fine-grained predictions, allowing more complex trading strategies. For example consider the strategy which buys or shorts an amount which is proportional to the magnitude of the prediction. The idea here is that since regression intrinsically penalizes large deviations, it may induce risk-mitigating trading strategies.

2.4. Naive Bayes

We applied a basic naive Bayes classifier (NBC) to our dataset to give a benchmark against which we can compare our SVM. A naive Bayes classifier works by finding the class which has the highest probability conditioning on the observed inputs. The NBC assumes that every document is produced from a mixture model and that given a document and its class, the words in the document are generated independently from each other. To compute the probability of a document belonging to a class, the NBC uses the Bayes rule as follows:

$$P(c_j|D) = \frac{P(c_j)\Pi_i P(w_i|c)}{P(D)}$$

, and makes a classification by computing

$$\text{argmax}_{c_j} P(c_j)\Pi_i P(w_i|c)$$

. The naive Bayes method is a very fast way to generate a decent linear classifier.

3. Features

We investigated several variations on feature representations of articles. The primary representation of an article is a "bag of words", represented by a vector of word counts. We investigated whether it would be beneficial to discard the most frequent or the most infrequent words; or to include a couple other derived features. For comparison, we tried a feature representation based on historical prices. The specific feature representations are given below.

3.1. Standard Feature Representation

We consider all the 7358 English words that occur in our corpus of articles. Each article is represented by a vector v, where v_i gives the number of occurrences of word i in the title and body.

3.2. Feature Variant 1 (V1)

We eliminate the 2299 words which only occur once. The intuition behind this variant is that when we split our dataset into training and testing set, these words either do not contribute to the training or are not useful when being classified. Overall, these words do not contribute to lowering the generalization error.

3.3. Feature Variant 2 (V2)

We eliminate the 10% of words which are the most frequent. (These are words which occur 28 times or more). The intuition behind this variant is that the most frequent words are likely to be articles and neutral words that have no sentiment.

3.4. Feature Variant 3 (V3)

Each article is represented by the following features: (1) the time of the day the article is published, (2) the day of the week the article is published, (3) the length of the article in characters, (5) the number of words in the title, (6) the number of words in the body, (7) the number of numbers and symbols in the article, (8) the average word length of the article, (9) the sum of positive scores of all words in the article, and (10) the sum of negative scores over all words in the article, where the positivity and negativity values of each English word is obtained from SentiWordNet.

3.5. Feature Variant 4 (V4)

For each article, we include the opening, closing, high, and low prices, as well as the transaction volume for each of the 20 trading days before the day in which an article is released, a total of 100 features per article. The goal of this variant is to compare how a company's performance in the past month reflects the after-day price changes against how articles reflect the after-day price changes .

3.6. Combined Feature Representation 1 (C1)

This is the concatenation of feature variant 3 and feature variant 4.

3.7. Combined Feature Representation 2 (C2)

This is the concatenation of the standard representation and variant 4.

Another representation that we considered but did not pursue is bigram-representation of the article, where every two consecutive words is considered as a feature. The reason we did not pursue this is because of the already large feature space relative to the number of training points that we have. In addition, upon careful examination of the content of the articles that we have collected, as presented in section 2.1, the articles are direct and objective. There isn't much negation, nor many instances of complex tone (e.g. sarcasm).

4. Results

All the results reported below come from using after-day-labels. We also tried next-day-labels and combined-labels, but found across the board that articles have a diminished ability to forecast these. This makes sense, because next-day-labels and combined-labels extend further into the future.

4.1. K-Fold Cross Validation (KFCV)

Initially to test our model, we implemented `split_data` which randomly permutes the dataset and partitions it into training and test sets. We tested our model by training and testing on different splits of the input data and averaging over the error rates.

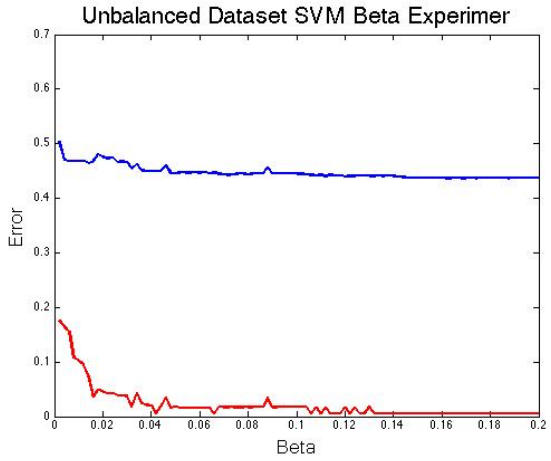

Figure 1. The training error (red) and testing error (blue) of SVM with fixed $C = 1$ and β varying from 0.001 to 0.2 using the standard features. For values of β greater than 0.2, the error rate converges to 43.47%.

However, in order to compare different models against each other and against different feature representations, we implemented `KFCV`, which is actually a K-fold cross validation, where we divide the entire dataset into K blocks where each block is left out during each iterations while the remaining K-1 blocks are used as the training set. We made this modification so that each experiment could finish in a reasonable amount of time. Throughout this project, K has been selected as 18.

4.2. SVM

In our first attempt at using a SVM, we used the standard feature representation of the entire 1737 articles that we collected, the radial basis kernel and, and default values of $C = 1$ and $\beta = 1$. We obtained the generalization error of 43.47% from our K-fold cross validation.

We varied our parameters in order to find the values of C and β that would lower the generalization error by fixing one value and varying the other. The result of fixing $C = 1$ and varying β over the interval of $[0.001, 0.2]$ is shown in Figure 1. We also experimented with β up to 10, and the result shows that the generalization error for $\beta \geq 0.1$ converges to 43.47%. In addition, we fixed β and varied C over the interval of $[0.01, 1000]$ and found that for any $C \geq 1$, the generalization error is 43.47%. We also see that the training error for when $\beta \geq 3$ and $C \geq 1$ is smaller than 0.003.

This result is contrary to what is expected, and is a clear example of overfitting. Upon further investigation, by looking at the distribution of positive and negative after-day-labels and by looking at the average number of false-positive and false-negative classifications, we found that the fraction of negative points in our dataset is 43.52%, and that the SVM is making 99% positive classifications. (Though the 1% negative classification has greater than 99% accuracy.) This led us to balance our dataset to obtain 1304 articles with balanced labelings.

Table 1. SVM Training and Generalization Error Over All Feature Representation Variant Performances

Variant	Train Error	Test Error	β^*	C^*
Standard	1.69%	48.06%	.0120	.0500
Variant 1	0.32%	48.29%	.0120	1.000
Variant 2	4.60%	48.29%	.0600	2.000
Variant 3	42.60%	**47.07**%	.0400	1.000
Variant 4	30.32%	47.37%	.0800	2.000
Combined1	43.85%	50.19%	.0020	1.000
Combined2	0.15%	49.20%	.0140	1.000

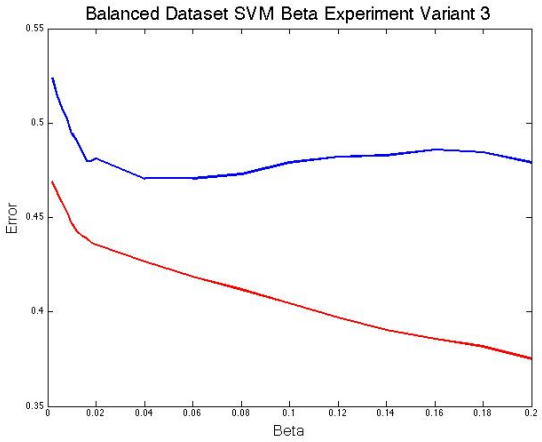

Figure 2. The training error (red) and testing error (blue) of SVM with fixed $C = 1$ and β varying from 0.001 to 0.2 using Feature Variant 3. For values of β greater than 0.2, the training errors continues to decrease and the generalization error increases to 51.11% at $\beta = 10$.

After balancing our dataset, we ran two experiments over every feature mentioned in the previous section in order to find the best values of β and C. In each of our experiments, we fixed one and varied the other, and used KFCV to assess our training and testing errors. Specifically, to find the optimal value of β for each feature variant, we fixed $C = 1$ and iterated over

37 values of β ranging from 0.001 to 20. To find the optimal value of C, we fixed β for each feature variant as the optimal value we found in the previous experiment and iterated over 28 values of C ranging from 0.01 to 10. The resulting best values of β and C and their corresponding test and training error for each feature variant is shown in Table 1.

Unlike the result from using the imbalanced dataset, the generalization error as a function of varying β with a fixed $C = 1$ shows a healthy global minimum at $\beta = \beta^*$, and the training error decreases as we increase β. A sample plot of the training and test error for feature variant 3 is shown in Figure 2.

In addition, we also considered linear kernels for a number of feature variants (S, V3, V4, and C1). We explored C values ranging from 0.001 up to 1. For small values of C, we needed to adjust the tolerance at which the KKT condition was checked. For large values of $C \geq 0.8$, when using the standard feature variant, we found it necessary to increase the maximum number of iterations allowed while using the Quadratic programing method. The best C value for each variant and their corresponding training and K-fold cross validation errors are shown in Table 2.

When working with the standard feature set, the number of features is the number of distinct English words in the entire dataset, which was 7358. Since the VC-dimension of a k-dimension linear classifier is greater than k and our entire dataset after balancing and merging consists of 1304 articles, we know it is possible to generate a model with zero training error. However, we are using slack variables so we get slightly increased error of 0.07% . Feature variants 3 and 4, on the other hand, have much smaller feature spaces, which explains the large training error from using the linear kernel.

Among all the feature variants, we expected feature variants 3 and 4 to do well. Specifically, the positive and negatives scores in feature variant 3 should be a good indicator of the sentiment of the article. And if our assumptions are correct, the sentiment of an article should be a good indicator of the stock's price change. Feature variant 4 was expected to do well based on previous research mentioned in the Introduction section. In addition, we expected the combined feature 1 to perform the best, combining two of the best performing feature variants can, at worst, give a equally performing classifier. However in practice, the result showed a worse performance of 50.19% error. Since the lowest test errors for variants 3 and 4 are found with different values of β and C, we can not make this a strict comparison with the combined feature. We also consider the possibility that SVM optimized for a greater margin, but did not generalize well when tested with our K-fold cross validation.

Table 2. Linear SVM using Selected Feature Representation Variants

Variant	Train Error	Test Error	C^*
Standard	0.44%	49.17%	1
Variant 3	46.71%	50.33%	.5000
Variant 4	46.28%	49.25%	.0050
Combined1	42.70%	49.00%	1

4.3. Regression

We performed two experiments to find optimal parameterization for β and λ for all seven feature variants. First we fixed λ at 1, while varying β over 37 values. Then using the optimal β values, we iterated over 28 values of λ. In both cases we pick the parameterization that leads to the smallest KFCV test error when using the regression model to make directional predictions. Variant 3 performs best as is evident in Table 3.

We expected the regression model to have similar results to the SVM because their optimization problems are so similar. In fact, the results are generally comparable in terms of testing error. The exception seems to be feature variant 4, which performs much worse with a regression model. At the same time, the regression model for V3 achieves our overall best performance with a test error of 46.54%, beating the best SVM by a small margin. One explanation is that a regression model gets more information from real-valued labels, than SVM models get from single bit labels.

Table 3. Regression Training and Generalization Error Over All Feature Representation Variant Performances

Variant	Train Error	Test Error	β^*	λ^*
Standard	0.07 %	47.98%	0.18	0.01
Variant 1	0.07 %	47.98%	0.18	0.07
Variant 2	4.56 %	49.09%	0.18	0.005
Variant 3	1.02 %	**46.53%**	0.004	0.005
Variant 4	0 %	54.14%	0.002	0.005
Combined1	0 %	54.14%	0.002	0.005
Combined2	0.31 %	49.95%	0.12	0.01

Table 4. NBC Feature Representation Variant Performances

Variant	Train Error	Test Error
Standard	17.11%	50.02%
Variant 1	18.61%	50.09%
Variant 2	14.40%	52.00%
Variant 3	49.08%	53.54%
Variant 4	49.79%	49.80%
Combined1	49.41%	51.01%
Combined2	17.10%	50.31%

4.4. Naive Bayes

We tested the effectiveness of the naive Bayes classifier by using K-fold cross validation on all seven of the feature variants. It performs worse than the support vector machine across the board. The generated models reveals that "billing", "increases", and "proposes" are very positive words; that "extreme" and "disclosing" are very negative. However the predictive strength of these words is uncertain given the high observed testing error. The results are in Table 4.

5. Discussion

The ideation of this project is based on the assumption that the price change of a company's stock is caused by the news and announcements made about the company on the day of or day before. Realistically, many causal factors impacting asset movements. Also, even if news articles have a large role in investor decisions, so do expectations. For instance, an article with a positive announcement may discourage an investor if its not as positive as expected. In light of this complicated environment, we think the 47.07% achieved by variant 3 is pretty good considering two major limitations: the lack of training data and extremely noisy labels. (As a comparison, testing our SVM on a random labeling gave us 51%.) Furthermore, we think it is very noteworthy that the the derived article features (V3) have as much predictive power as the historical prices (V4). This suggests that news articles can be an important signal in financial decision making.

Finding news articles going 2 years back was a limiting factor mainly due to the fact that news becomes archived after a short amount of time. This introduced a problem for us: when the number of features is greater than the size of the training set, we cannot say anything about the test error because the bound on the generalization error (with probability at least $1-\delta$),

$$R(h) \leq R_n(h) + \sqrt{\frac{h(\log(\frac{2N}{h})+1) - log(\frac{\delta}{4})}{2N}}$$

where $h = \text{VC}(\mathcal{H})$ only holds when $h \ll N$. The same reason explains that when the linear kernel is used to classify the standard feature variant, the separater did no learning yet it was able to separate more than 99% of the training point correctly.

To possibly reduce the difference between the size of our training set and the size of our feature space, in feature variants 1 and 2, we eliminated 30% and 10% of our features respectively and found no improvement in the generalization error. This lack of improvement could still be due to our limited number of articles. In order to obtain much larger number of financial news articles related to a set of companies, we would imitate some existing research and collect our data over longer period of time, scraping news on a daily or weekly basis.

Our second limitation was due to the lack of accurate labels. While we collected after-day and next-day price changes, a number of other factors could be better indicators of the price change. For instance, if the article is released earlier in the trading day, the effect on the market may have been taking into account 30min after the release; or the price change could have been caused by a more global news event. A possible solution is to gain access to intra-day price change. Then, we could label each article by the price change 30 minutes or an hour after the release of the article. This will allow us to generate labels that are more finely tuned, and that are better representations of the articles.

6. References

[1] Kyoung-jae Kim, Financial time series forecasting using support vector machines, Neurocomputing, Volume 55, Issues 12, September 2003, Pages 307-319, ISSN 0925-2312.

[2] Xue Zhang, Hauke Fuehres, Peter A. Gloor, Predicting Stock Market Indicators Through Twitter I hope it is not as bad as I fear, Procedia - Social and Behavioral Sciences, Volume 26, 2011, Pages 55-62, ISSN 1877-0428.

[3] E. J. Ruiz, V. Hristidis, C. Castillo, A. Gionis, and A. Jaimes. Correlating nancial time series with microblogging activity. In Proc. of the 5th ACM International Conference on Web Search and Data Mining, WSDM 12, pages 513522, Seattle, WA, 2012.