Optimised Prediction of Stock Prices with Newspaper Articles

Bryan Cheong, Stanford University, Mathematical and Computational Science, `bcheong@stanford.edu`
Christopher Yuan, Stanford University, Computer Science, `cqyuan@stanford.edu`
Stephen Ou, Stanford University, Computer Science, `sdou@stanford.edu`

1. ABSTRACT

For this project, we investigated the predictive power of newspaper articles on the stock prices of various companies. Using supervised learning, we were able to obtain a testing accuracy of up to 61%. We then performed reinforcement learning on this predictor, feeding its predictions into a Markov Decision Process (MDP) which bought and sold shares on a simulation that we programmed. We compared the MDP's performance on a year's stock prices to that of a random guesser. Overall, we conclude that the MDP with our predictor generally outperforms a random guesser.

2. INTRODUCTION

We treated data from newspaper articles with machine learning tools to predict up or down changes in the stock prices, i.e. our response was binary. In particular, we examined four algorithms, the Naive Bayes algorithm, SVM, the Perceptron, and Boosting with weak learners on a Bag-of-Words model of newspaper articles. In addition, we optimised these algorithms by stemming the Bag-of-Words, introducing Term Frequency-Inverse Document Frequency (TF-IDF) to improve our algorithms' grasp of the relevance of words, and finally with cross-validation for the best parameters.

After obtaining a base predictor that predicts using newspaper articles whether the stock of a company would go up or go down, we also wrote a Markov Decision Process (MDP) learner that builds on the base predictor using reinforcement learning that is able to buy and sell shares of a company. In order to do so, we needed to make the assumption that this approximates to a Markov process. Our final simulation tests if there may be credence in such an assumption, despite its limitations.

3. PREVIOUS WORKS

Our literature survey found five works that also used news articles for classifying stock prices. However, almost all of these previous works used a Naive Bayes classifier on a simple Bag-Of-Words model, which generally performed poorly.

Gidfalvi [1] implemented a naive Bayes text classifier on financial news to track very short-term (on the scale of minutes) fluctuations in stock prices, and concluded that there is a strong correlation between news articles and the movement of stock prices in a window between 20 minutes before and 20 minutes after the news articles are published.

Two such previous works were former CS 224N projects from Stanford University, Ma [2], and Lee and Timmons [3]. Both projects used two approaches, a naive Bayes classifier and a logistic regression classifier (which both papers referred to as a maximum entropy classifier), and both projects generally concluded that the Naive Bayes classifier was relatively ineffective, and the logistic regression classifier had slightly better accuracy.

Chen et al. [4] are an undergraduate group at UC Berkeley that used a Naive Bayes classifier and sense-labelling of words in news articles (whether they are positively or negatively connotated) to categorise stock movements in the companies mentioned in these articles, once again to test the efficient market hypothesis. However, they concluded that the sense-labelling of news articles have no impact on the movement of stock prices.

The last work is a Masters thesis from the Norwegian University of Science and Technology by Aase [5]. In his work, Aase used an SVM text classifier with an RBF kernel and sense labelling of words to make predictions on stock prices, but limited his scope only to Norwegian oil companies.

We observe that none of these previous works employed TF-IDF or cross-validation to improve their algorithms, which may explain why they mostly concluded that newspaper articles have weak predictive power on changes in stock prices. In addition, none of these previous works used reinforcement learning to improve on their base predictor.

4. DATA

We needed to obtain articles that mention specific companies from reputable newspapers in order to compile our data set and set up our Bag-of-Words models. We decided to use 'reputable sources' instead of publications like Twitter, Buzzfeed, etc. because we hypothesize that reputable articles based on fundamental company research and understanding will generally have more predictive power than less reputable articles. Unfortunately, the newspapers that we wish to examine, such as the New York Times and the Wall street Journal, have paywalls that block their websites and make automatic scraping difficult. Instead, we relied on the Proquest NewsStand database to obtain our data.

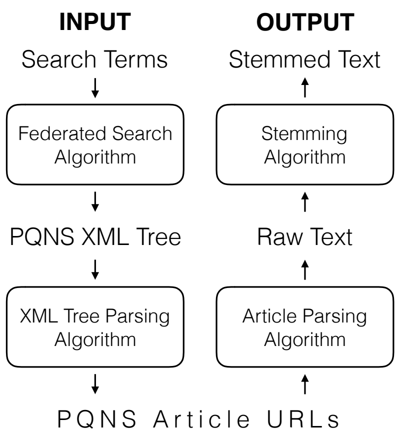

Fig. 1 Webscraping pipeline

Proquest NewsStand archives all articles from these newspapers in a searchable database. By writing an algorithm to generate Federated Search URLs, we were able to automatically obtain the PQMS XML Tree which contains the URLs of the full texts of articles that mention the company in interest, from the newspapers that we want the articles from, and the year for which we wish to generate the data. We then wrote an XML tree parser in order to automatically compile a list of all the PQNS URLs wherein the article full texts are contained.

Unfortunately, Proquest does not allow these articles to be accessed without a validating URL. Hence, we also had to write a web scraper specifically tailored to work around the limitations of the Proquest database by generating cURLs so that our automatic article gatherer behaved like a real user's browser. We then used regular expressions to parse only the full texts and the article publication dates from the webpages to finally obtain our raw text data, which could then be treated with the Python stemming library for pre-processing. The schematic in Fig.1 summarize the webscrapping pipeline that we used in order to obtain our text data.

Next, we modeled our input data (i.e. news articles) as a Bag-of-Words model. For the input matrix X, each of the n columns represents a word in our dictionary, and each of the m rows represents a news article. Each X_{ij} entry contains the frequency for which the word j appears in the news article i.

Our response variables were the stock movements of individual companies. We obtained the end-of-day stock prices for the last twenty years of six companies:

(1) Boeing
(2) General Motors
(3) McDonald's
(4) Tesla
(5) Twitter
(6) Valeant

We chose these particular companies either because their stock prices are volatile or mainstream news publications frequently write about them.

The stock prices for these six companies were obtained from the Quandl database using their API. For our base predictor, we only took into account whether the stock prices moved up or down to obtain a binary response. Our response (i.e. stock prices) is a binary label where +1 indicates the stock price has gone up and -1 indicates the stock price has gone down. We computed the label by subtracting the closing price on the day the article came out from the closing price before the day the article came out. If the article published date falls on a weekend, when the markets are closed, we used the difference between the Monday closing price and the Friday closing price.

We separated our data into a training and testing set, where the training set was twice the size of the testing set. In total, we processed 6,776 articles for our dataset over six companies.

5. METHODOLOGY

5.1 SUPERVISED LEARNING ALGORITHMS

We used four supervised learning algorithms to treat our data. Since previous literature mostly used the Naive Bayes algorithm, we included it in our investigation to compare with other algorithms. For Naive Bayes, we made what's called the Naive Bayes assumption - we assumed that all the x_i's are conditionally independent given y. Therefore,

$$\begin{aligned}&p(x_1,\dots,x_n)\\ =&p(x_1|y)\cdot p(x_2|x_1,y)\dots p(x_n|x_1,\dots,x_{n-1},y)\\ =&p(x_1|y)\cdot p(x_2|y)\dots p(x_n|y)\\ =&\prod_{i=1}^{n}p(x_i|y)\end{aligned}$$

Now, to figure out whether the stock price will go up ($y = 1$) or go down ($y = -1$), we will calculate the probability

$$\begin{aligned}&p(y=1|x)\\ =&\frac{p(x|y=1)p(y=1)}{p(x)}\\ =&\frac{(\prod_{i=1}^{n}p(x_i|y=1))p(y=1)}{\prod_{i=1}^{n}p(x_i|y=1))p(y=1)+\prod_{i=1}^{n}p(x_i|y=-1))p(y=-1)}\end{aligned}$$

Then, we will calculate the same probability for stock price going down, i.e. $p(y = -1|x)$. Now, we can classify whether $y = 1$ or $y = -1$ based on which one has a higher probability.

In addition, we also treated our data using Support Vector Machines (SVM) in the hope to capture the nonlinear relationship in our data. The Support Vector Machine model was initialised without smoothing on a Gaussian kernel,

$$K(x,z)=\exp(-\frac{1}{2\tau^2}||x-z||_2^2)$$

Then, we used the Perceptron algorithm, a simple linear algorithm that decides whether an input belongs to one class or another. We first tried the algorithm with a bias term, so we made a prediction according to

$$h_\theta(x)=g(\theta^T x+b)$$

where

$$g(z)=\begin{cases}1 & \text{if } z\geq 0\\ -1 & \text{if } z<0\end{cases}.$$

Finally, we applied a Boosting algorithm that automatically chooses its feature representation. In this case, our Boosting algorithm was initialized using simple decision stumps as weak learners.

Our motivation for using these four algorithms was to compare their relative performance in testing accuracy as we applied the various optimisations in subsequent steps.

5.2 OPTIMIZATIONS

Our first step in dealing with the raw text data was to pre-treat it with a stemming library. This is to ensure that reflects the various forms of a word (e.g. "diversify," "diversifying," "diversified") as the same token. Our reasoning for this is that the various forms of the word do not differ significantly in semantic content, and indicate roughly the same meaning when they occur in newspaper prose, so the stemmed forms of the words is a better representation. We will compare the test accuracy of the various algorithms using stemmed and unstemmed tokens.

Our second step in optimising the four algorithms was to apply Term Frequency - Inverse Document Frequency (TF-IDF) weighting on the stemmed text data. TF-IDF places the following weights on the tokens as they appear:

$$\mathrm{tfidf(t,d,D)=tf(t,d)\cdot idf(t,D)},$$

$$\mathrm{tf(t,d)} = 1 + \log \mathrm{tf_{t,d}},$$

$$\mathrm{idf(t,D)} = \log \frac{\mathrm{size(D)}}{\mathrm{df_t}},$$

where t refers to the term index, d refers to the document index, and D the corpus of all documents. Additionally, $\mathrm{tf_{t,d}}$ refers to the number of times the term t appears in the document d and $\mathrm{df_t}$ refers to the number of documents D the term t appears in. Finally, $\mathrm{tfidf}(t,d,D)$ refers to the final weight placed on the term t. This weighting places less weight on words that occur in more articles, and that are therefore less able to distinguish between articles.

Our motivation for applying TF-IDF was because we observed that a naive weighting on token frequencies put heavy weights on very common words such as "the," "for," etc. which dominate other words in terms of absolute frequency. We hypothesized that while such words are, on an absolute level, more common words, they are less able to distinguish between articles than rarer words that are able to distinguish more between articles, or better reflect an article's semantic content.

Finally, we applied parameter optimisation via cross-validation of these four algorithms, drawing on the training data for a cross-validation set. The final optimised parameters are: Naive Bayes with a Laplace smoothing of 1 on the observations, SVM with a linear kernel without smoothing, Perceptron without regularisation, and Boosting on decision trees with arbitrary depth on 100 estimators. We used the final parameter-optimised algorithms to be used on our MDP reinforcement learning simulation.

5.3 REINFORCEMENT LEARNING

Having obtained a base predictor for stock price movements based on newspaper articles, we applied reinforcement by using it to build a Markov Decision Process (MDP) that simulated the buying and selling of shares. The base predictor we used to train the MDP is the optimised Naive Bayes predictor. We placed our MDP learner in a simulation run using real data from the stock price of Tesla in the year 2015, and using real newspaper articles from the New York Times and the Wall Street Journal, all of which were not in the training set used to train the base predictor. Our design of the MDP learner is as follows.

These are the parameters we used to write our Markov Decision Process:

(1) **States**: Percent return from the starting amount
(2) **Rewards**: Same as the state. A higher percent return equates to a higher reward
(3) **Discount factor**: 0.995
(4) **Actions**: For the most part, our algorithm was allowed to buy or sell up to 3 stocks, or take no action. However, we placed restrictions on the actions based on the following conditions:
    —If we had negative cash, the algorithm was only allowed to sell stocks (up to 3)
    —If there was no article for a particular day, the only allowable action was to do nothing (0)
(5) **Transition Probabilities**:
    —Before discussing transition probabilities, we must first consider the observations we recorded. We observed every transition from one state to another under an action. For example, if we moved from 3% return to 4% return by buying 3 shares, we would record it.
    —Now, let the prediction we get from an article on a given day be represented by p, and the test accuracy of that prediction be η. Let the number of states we have be represented by N. Finally, let $O_{sa}(s')$ be the number of observations from state s to s' under the action a. Then the transition probability is given by the following:

$$P_{sa}(s';p,\eta,N) = \begin{cases} \left(\frac{1}{\eta N}\right) O_{sa}(s') & \text{if } p(s'-s) \geq 0 \\ \left(1-\frac{1}{\eta N}\right) O_{sa}(s') & \text{otherwise} \end{cases}$$

In order to implement MDP, however, we needed to fundamentally assume that the day-to-day transitions of a stock buy-and-sell portfolio satisfy the Markov property. Such an assumption is, we acknowledge, possibly contentious. While changes in the price of a stock on a day-to-day basis and the actions of buying and selling a stock are arguably independent, because our MDP is a trader, it necessarily needs to hold different amounts of stocks in its portfolio from day to day. This different level of stocks being held or shorted is not reflected in the state, which therefore dissatisfies the Markov property. Hence, we implemented a penalty on holding or shorting too much stock, such that the amount of stock held by our MDP's portfolio is usually within -3 to 3 shares. In that manner, it is able to buy or get rid of stocks within a single action, and so the transitions between states and actions approximates the Markov property. In other words, we are able to implement MDP by making the percent changes in the value of the portfolio and the actions of buying and selling stock a pseudo-Markov decision process.

Then, we implemented the Value Iteration algorithm to compute an optimal MDP policy and its value. For each day that the stock market is open, we compute the maximum Q values that a state can obtain by taking all the possible actions, where the Q value equals the sum of Q_i for each of the new states s reachable from the current state s, and

$$Q_i = \mathrm{transProb}(s,s') \cdot (\mathrm{reward}(s') + \mathrm{discount} \cdot \mathrm{value}(s')).$$

Please see the appendix A for a pseudocode summary of our Markov Decision Process and Value Iteration implementation.

6. RESULTS

Generally, the four algorithms showed incremental improvement with each optimisation that we applied, as demonstrated in the upward trend in test accuracy shown in Fig. 2, which plots the average test accuracy of the algorithms across the six companies for which we used in our dataset.

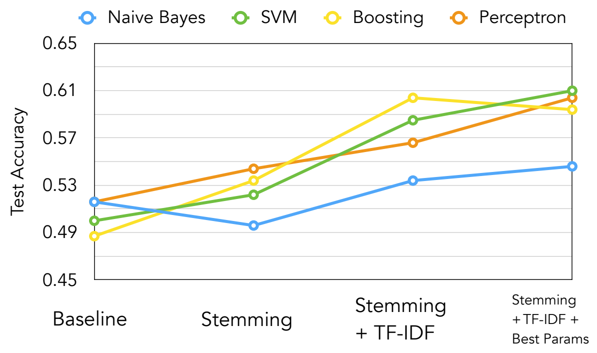

Fig. 2 Test accuracy with various learning algorithms

The baseline algorithms, without any optimisations, generally were no better than chance, with test accuracy around 0.5. However, after applying all our optimisations, the algorithms generally improved in test accuracy to around 0.6, except for the Naive Bayes algorithm which was favoured by previous literature, which was never able to obtain a test accuracy above 0.55. We summarise the test accuracy of the incrementally optimised algorithms in Fig.3.

	Naive Bayes	SVM	Boosting	Perceptron
Baseline	0.516	0.500	0.487	0.516
+ stemming	0.496	0.522	0.534	0.544
+ TF-IDF	0.534	0.585	0.604	0.566
+ optimised params	0.546	0.610	0.594	0.604

Fig. 3: Tabulated test accuracies from incrementally optimised algorithms

In addition, we also observed that after TF-IDF, the weights that the Naive Bayes algorithm placed on tokens seemed to be intuitively meaningful. We extract some of the top positive and negative weighted terms in Fig. 4 below. Seeing how the test accuracy for the algorithms improved all around after the application of TF-IDF, it bears out our hypothesis that such rarer words, which are better able to distinguish between different articles, should be given more weight. These rarer words, as listed Fig. 4., are what we would ordinarily expect to connote ”positivity” or ”negativity” surrounding a description of a company in a newspaper article.

Term	Weight
suggest	0.54
rumor	0.51
high	0.51
report	0.48
rise	0.45
express	0.44
staff	0.43
soar	0.42
diversified	0.40
reorganization	0.40

Term	Weight
wary	-0.51
discourage	-0.48
misinform	-0.48
reprehensible	-0.46
abusive	-0.45
dispute	-0.45
denounce	-0.45
cheapen	-0.43
rough	-0.41
takeover	-0.40

Fig. 4. Tokens with positive or negative weight from Naive Bayes predictor. In unstemmed form for readability.

Finally, for our results from the MDP simulation, we find that the learner is generally able to outperform a random guesser when buying and selling stock using the base predictor. Fig.5 below illustrates the percentage of return of our learner (in red) v.s. the percentage return of the random guesser (in blue). This figure is based on Tesla stock and articles from January 1, 2015 to December 31, 2015. After running simulation of 250 trading days (a full year has only around 250 working days, excluding weekends and holidays, when the stock markets are closed), our learner produces a return of 1.95%, while the random guesser produces a return of -2.88%.

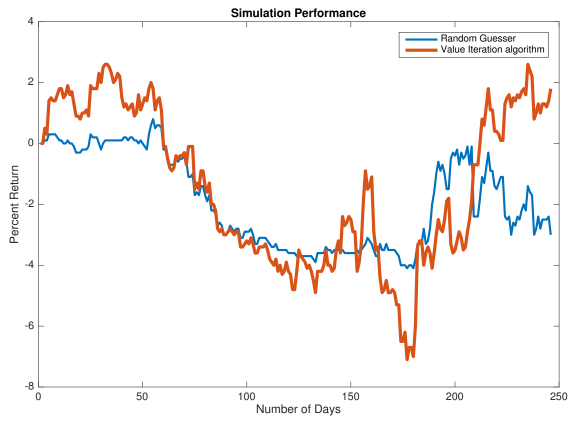

Fig. 5 MDP returns vs. Random guesser for Tesla, Jan. 01, 2015 to Dec. 31, 2015.

There may be multiple sources of error for our models. In particular, we noted that the testing accuracy of our Boosting algorithm decreased when we used cross-validation to optimise the parameters from using boosted decision stumps to boosted decision tree of arbitrary depth. This is probably a case of over-fitting.

In addition, since there were insufficient days in the year to allow the learner to converge, the MDP is still not making optimal decisions. It is unclear, however, if extending the learning period and increasing the amount of data the learner has access to will lead to convergence, since the if we extend the period to more than one year, the assumption of the Markov property may begin to break down, and the transition probabilities arrived at by the MDP for one year may not be useful in the next year, or in the long run in general. Whether this is a fundamental limitation of using MDP for such a use would require further experimentation.

7. CONCLUSIONS

In summary, the four algorithms that we used (Naive Bayes, Support Vector Machine, Perceptron, and Boosting) without any optimisations only produced results no better than random, as expected. However, adding several optimisations were able to bring the test accuracy up to 61%.

Stemming helps unify the words that are represented in different forms in our training and test data but have the same semantic meanings. TF-IDF helps increase the weights of rare words that are strong indicators of a particular class and decrease the weights of common words that are weak indicators of a particular class. Parameter tuning helps us to figure out what are the best parameters to use in the four algorithms in order to produce the best results.

There are few interesting takeaways from our work in modeling the stock simulation as a Markov Decision Process and running the Value Iteration algorithm on it. First, we found that modeling the discretised percentage returns as states helps the Value Iteration algorithm converge quickly. Second, we found that using past observations to determine the transitional probabilities gave us an accurate measure of the *magnitude* with which the stock would change in the future. Third, adding a penalty for buying too much stock helps prevent the case where it would take too long to sell the stocks during a negative run of a particular stock. Fourth, this penalty also helped us approximate the Markov process even when some assumptions about the model have been violated. Lastly, we acknowledge that in order to conclusively determine whether or not the MDP simulator consistently outperforms a random guesser, more analysis must be performed.

8. FUTURE WORK

We tested the four algorithms of our base predictor on individual companies' stock prices separately. It remains an open question as to whether such supervised learning methods would be effective in predicting stock price movements in general, i.e. if there is a general model that is able to be tested on companies in general. This would be especially interesting if there is significant predictive power when testing on companies that were never included in the training dataset in the first place.

It was clear that even after running a year's worth of data on our MDP algorithm, it still had not reached convergence after the end of the year. While we may suggest that the MDP algorithm be simulated over a longer time period than one year, there is an open question as to whether a company's stock price, over a longer period of time will still allow the MDP learner to approximate a Markov decision process. It is generally acknowledged that stock prices are more variable in the long run, hence the short term forces that underlie the MDP learner's transition probabilities may change over the long run. While investigating the long-run future accuracy of the base predictor, and the long run soundness of modelling stock purchase and sale on an MDP are interesting, they are unfortunately outside the scope of our project, wherefore we leave these questions to future work.

9. REFERENCES

(1) Gidofalvi, Gyozo. "Using news articles to predict stock price movements." *Department of Computer Science and Engineering, University of California, San Diego* (2001).

(2) Ma, Qicheng. "Stock Price Prediction Using News Articles." *CS224N, Final Report* (2008).

(3) Timmons, Ryan, and Kari Lee. "Predicting the stock market with news articles." *CS224N, Final Report* (2007).

(4) Chen, Jerry, and Aaron Chai, Madhav Goel, Donovan Lieu, Faazilah Mohamed, David Nahm, Bonnie Wu. Predicting Stock Prices from News Articles. (2015).

(5) Aase, Kim-Georg. "Text mining of news articles for stock price predictions." (2011).

10. APPENDIX

Algorithm 1 Reinforcement Learning Algorithm

```
date ← Jan 09, 2015
cash ← 2000
numStocks ← 0
observations ← [ ]
values ← [ ]
totalValue ← 2000
predictions ← getLearningPredictions()
prices ← getStockPrices()
state ← getStartState()
epsilon ← 0.01

procedure SIMULATOR
    for all dates do
        action ← getActionWithMaxValue()
        newState ← getNewState(state, action)
        $observations[state][action][newState] ++$
        runValueIteration()
        state ← $newState$

procedure GETACTIONWITHMAXVALUE
    values ← [ ]
    for actions reachable from state do
        for all reachable newState with transProb, reward do
            values.append(getValue(newState,
            transProb, reward))
    return argmax(values)

procedure GETMAXVALUE
    values ← [ ]
    for actions reachable from state do
        for all reachable newState with transProb, reward do
            values.append(getValue(newState,
            transProb, reward))
    return max(values)

procedure GETNEWSTATE
    $cash ← cash − prices[date] × action$
    $numStocks ← numStocks + action$
    date ← getNextDate()
    $totalValue ← cash + numStocks × prices[date]$
    state ← getReturn(totalValue)

procedure RUNVALUEITERATION
    while true do
        oldValues ← values
        for all states in MDP do
            values[state] ← getMaxValue(state)
        if max(oldValues - values) < epsilon then break
```