$stock Forecasting using Machine Learning

Greg Colvin, Garrett Hemann, and Simon Kalouche

***Abstract*— We present an implementation of 3 different machine learning algorithms — gradient descent, support vector machine, and a deep neural network — which attempt to model and forecast individual stock prices at some future date based on a set of 11 company related features. We vary internal parameters of each algorithm (optimization step size, epoch number, loss function, hypothesis functions, number of hidden layers, etc.) as well as more intuitive feature-based parameters (forecast horizon, window length, etc.). We summarize and present the performance of several experiments and indicate which physical parameters are most influential to the model in producing the model with the best 1-day, 10-day, and 20-day stock price horizon.**

I. INTRODUCTION

The US stock market is said to be semi-strong efficient meaning at any given time, prices fully reflect all available information on a particular stock and/or market. Therefore, no investor has a legal advantage in predicting a return on a stock because no individual investor has access to information not already available to the public. The goal of this study is to determine if an advantage could be gained by utilizing the available information more effectively than the individual investor. Existing mathematical financial market models include Black-Scholes and Brownian models [1][2]. However, in this study, artificial intelligence, specifically machine learning techniques, were implemented with the goal of predicting future stock prices based on data from past and present metrics which are commonly deemed influential stock value indicators. Machine learning using gradient descent to minimize a soft-max loss, a support vector machine, and a deep neural network were used to predict performance based on 25 years data for 11 different criteria across 6 different companies from different industries. Three technology companies (Apple, Microsoft, and HP), two retail companies (Kohls and Macys), and one manufacturing company (3M), were evaluated. Daily values for the following criteria spanning 25 years were collected for each of the companies from the Bloomberg Database.

*This work was done in partial fulfillment of CMU's Graduate Artificial Intelligence Course (15-780) in the Spring of 2016.

G. Colvin, G. Hemann, and S. Kalouche are with the Robotics Institute at Carnegie Mellon University's School of Computer Science.

II. TECHNICAL APPROACH

Three artificial intelligence techniques were used to attempt prediction of stock performance. The three algorithms are a traditional machine learning approach which uses a softmax loss with gradient descent, a support vector machine which minimizes uses a linear kernel (squared hinge loss function), and a neural network which minimizes a logistic loss.

The input data parsed from the Bloomberg Database is assembled in the form

$$F = [f_1, f_2, f_3, ..., f_n] \tag{1}$$

where F is the complete feature set and n is the number of features ($n = 11$ for Bloomberg Database features, $n = 15$ for additional Google Trends data, see II-D).

Each machine learning algorithm used a supervised learning approach which requires a trained data set, Y. Y is chosen based on the metric we are attempting to predict. For instance, if we wish to predict 2-day net price change on day $i+1 = 50$ with a 10-day window, the Y for that prediction will be the actual 2-day net price change that occurred on day 51 and the X will be the feature set data from day 40 to 50. The feature set data is shaped by the number of features used n, as well as the window length w which indicates how many past days of data ($i - w$) are used to predict the desired feature on day $i + 1$.

The data fed into the algorithm is $X \in \mathbb{R}^{mxn}$, and the trained data is $Y \in \mathbb{R}^{mxq}$, where m is the number of days (data points per feature) and q is the number of classes in the classification problem. In the case of 2-day net price change, $q = 2$ where $y_i = [0, 1]$ if the 2-day net price change was positive and $y_i = [1, 0]$ if the 2-day net price change was negative. We took a classification approach where two classes indicate a feature's sign. Thus, if $argmax(\hat{y}) = 0$ the ML model predicts a negative 2-day net price change and if $argmax(\hat{y}) = 1$, then the model predicts a positive 2-day net price change.

We generate our input data set X based on window length by

$$X_i = [F_i, F_{i-1}, F_{i-2}, ...F_{i-w}] \tag{2}$$

where i indexes the day and F_i is the full feature set, F, data for day i. Y is formulated to be

$$Y_i = [sign(f_3)_{i+1}] \tag{3}$$

where in the case of predicting 2-day net price change (i.e. f_3 in the feature set) we look one day into the future and we use day $(i+1)$'s 2-day net price change as the trained data point for X_i.

We then implement our 3 different ML algorithms and feed these algorithms with X and Y of this form to train the model weights, $\Theta \in \mathbb{R}^{qxk}$, where $k = n*w$.

In order to compare each algorithm's performance, the stock price was predicted for each stock while varying the amount of input data (i.e. window length w) and the forecast horizon (1-day, 2-day, 10-day, or 20-day). There were approximately 6200 time steps covering the 25 year period, and this was split using the standard 70/30 convention where 70% becomes training data and 30% is testing data.

The test and training prediction error was calculated from the ratio of correctly predicted sign change in metric Y (1-day, 10-day, or 20-day net price change) to the total number of training or testing data samples. The precise equation used for error is

$$error = \frac{\sum \mathbf{1}\{sign(\hat{y}) = sign(y)\}}{n} \tag{4}$$

where n is the number of testing or training samples and the **1-function** returns 1 when the expression in the brackets is satisfied and 0 otherwise.

A. Machine Learning Using Gradient Descent

A traditional machine learning algorithm using gradient descent to minimize a softmax loss was first implemented. The soft-max loss is defined by

$$l(y, \hat{y}) = \log(\sum \exp(y')) - y' \cdot y \tag{5}$$

where $\hat{y} = h_\theta(x)$. The gradient of the softmax loss is defined by

$$\nabla l(y, \hat{y}) = \frac{\exp(y')}{\sum \exp(y') - y} \tag{6}$$

The algorithm was tuned with alpha = 0.025 and was shown to approach steady state after 150 epochs.

TABLE I
FEATURE SET

Index	Feature	Description
f_1	1-Day Net Price Change	Difference in Today's Last Price and Yesterdays Last Price
f_2	Last Price	End of Day Market Price
f_3	2-Day Net Price Change	Difference in Today's Last Price and Two Days Previous Last Price
f_4	10-Day Volatility	Standard Deviation of Price Change Over 10 Days
f_5	50-Day Moving Average	Mean Market Price Over 50 Days
f_6	Price to Earnings Ratio	Market Price per Share/Earnings per Share Volume
f_7	Volume	Shares of Stock Outstanding
f_8	Enterprise Value	Companys Total Value
f_9	Overridable Alpha	Stock Performance against S&P 500 Sector Index
f_{10}	Overridable Beta	Measure of Stocks Price Volatility Compared to Sector Index
f_{11}	Alpha for Beta Plus Minus	Alpha Over Beta
f_{12*}	Google Trends	Trends data indicating search popularity of certain keywords related to a company

B. Support Vector Machine

A generalization of the traditional gradient descent forms our second algorithm which is the support vector machine (SVM), where hyperplanes classify different segments of data. SVM's can use linear kernels like hinge loss to solve the gradient descent problem, but can also use higher order polynomial kernels for more complex data. In our formulation, we found the linear kernel performs better, which uses a squared hinge loss on a binary class. The squared hinge function is

$$l(y, \hat{y}) = \sum max\{1 - y_i \cdot x_i^T \Theta, 0\}^2 + \lambda \sum \Theta_i^2 \tag{7}$$

where the second term $\lambda \sum \Theta_i^2$ is the L2 penalty regularizing the loss. The SVM was implemented using Python's SK-learn library.

C. Deep Neural Network

The third algorithm that was implemented was a deep convolutional neural network (CNN). Although recurrent neural networks (RNNs) are typically used for time-series data (as is the case here) we employ a CNN to predict stock prices movement from a representative 'picture' of a time-series of past price fluctuations.

The neural network implementation differs from the previous two algorithms in that a logistic loss was used

to yield a single prediction value $\hat{y}$ thus making the neural network a regression algorithm as opposed to the previous two classifiers. The CNN implemented a logistic loss (l) and a loss gradient (∇l) defined as

$$l(y, h_\theta(x)) = l(y, \hat{y}) = \log(1 + \exp(-\hat{y} \cdot y)) \quad (8)$$

$$\nabla l(y, \hat{y}) = \frac{-y \cdot \exp(-\hat{y} \cdot y)}{1 + \exp(= \hat{y} \cdot y} \quad (9)$$

Before the optimization occurs a non-linear function fu_j is applied to the transformation linear hypothesis function $h_\theta(x)$. Therefore, the hypothesis function now takes the form

$$h_\theta(x) = fu_{j+1}\Big(W_{j+1} fu_j(W_j x + b_j) + b_{j+1}\Big) \quad (10)$$

Here, fu is a non-linear activation function where common non-linear functions used are sigmoid, hyperbolic tangent, and the rectified linear unit (ReLU). In our study we found that the ReLU activation function produced the best experimental results for all layers except the last layer which was set to be a linear activation function. The ReLU function is applied element-wise and defined by

$$fu(x)_{ReLU} = max\{0, x\} \quad (11)$$

where the function returns zero for values where x is negative and x for values of x that are positive.

The number of layers (s) was also varied to determine if the prediction improved with a deeper or shallower layer. The total number of layers varied from 5 (3 hidden) to 25 (23 hidden) where each layer size was linearly spaced from layer 1 (L_1) having a size of the input data ($\mathbb{R}^{mxk}$) to the last layer (L_s) having a size of ($\mathbb{R}^1$) because the last layers should yield the predicted value of only one feature. It was determined that more layers did not produce a more accurate model and so the network was kept to 5 layers.

The weights matrix **W** and bias vector **b**, which make up the are initialized randomly and then fed into a stochastic gradient descent optimization algorithm which searched for the parameters $\theta = [\mathbf{Wb}]$ which minimized the logistic loss function. The optimization used a constant step size of $\alpha = .005$ and 10 epochs (i.e. 10 runs through the entire input data set.

To appropriately compare the error of the three algorithms the predicted values of the CNN are converted into classes based on $sign(\hat{y})$, where a positive prediction corresponds to $\hat{y} = [0, 1]$ and a negative prediction value corresponds to $\hat{y} = [1, 0]$. The error of the neural network is then calculated according to eq. 4.

As done in the previous two algorithms, the forecast horizon and window length parameters were varied from 1 to 6 weeks and 1 to 20 days, respectively.

D. External Features using Google Trends [1]

In addition to the Bloomberg Database stock criteria, we evaluated the effects of external data pulled from Google Trends as seen in fig. 4. For the Apple stocks, we used the search terms 'apple', 'ipad', 'iphone', and 'ipod' to see if the magnitude of searches overtime would impact stock predictability. Magnitude is normalized per-topic by the maximum of searches-per-day over the time window (12 years).

III. RESULTS

We first compare the results of the three algorithms shown in Figs. 1 and 2 with varying forecast horizons. The vertical axes shows the testing error of the algorithm and the horizontal axes show the price change at various distances in the future we were trying to predict: 1, 10, & 20 days. In addition to varying the forward distance of the predication, each bar represents how much of the past data that we based our prediction on: 1, 2, 3, 4, 5, & 6 weeks of data. The plots show that the algorithm preformed the best when predicting the short term price change. It also demonstrates that no algorithm had significant improvement over another.

The traditional machine learning algorithm using gradient descent was run on the 6 test companies and compared against a baseline of an always-increase predictive strategy (see Fig. 3). This is admissible because on average, stock prices can be expected to rise steadily at a rate of 2-5% per year. From Fig. 3 it is evident that the model's prediction accuracy varies across companies.

Our results demonstrate that external effects (features not included in our algorithms) have greater impact on Apple stock than they do on 3M stock. This may occur due to Apple's volatility is influenced on a weekly/monthly basis by the release, reviews, and issues with their consumer products. On the contrary, 3M is a much less volatile stock whose stock value is less dependent on product releases and thus varies much less on a day-to-day basis.

[1] https://www.google.com/trends/

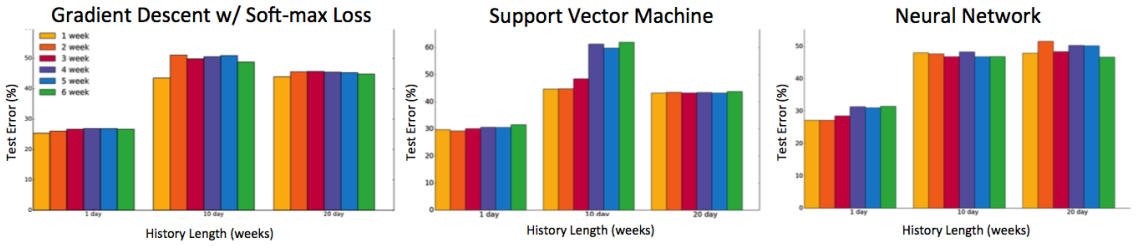

Fig. 1. Comparison of ML algorithms for 3M Stock

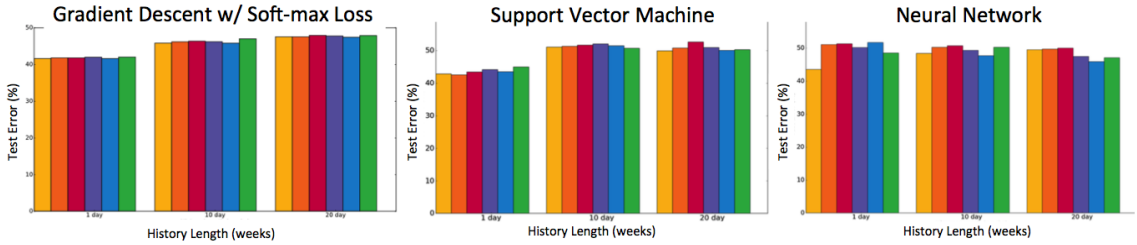

Fig. 2. Comparison of ML algorithms for Apple stock

In an attempt improve Apple stock prediction, we added Google Trends data of apple search terms to the feature set. These additional features showed no prediction improvement, which we believe is the result of Google Trends data not publishing a connotation associated with the keywords. For example, if the keyword 'iPhone' has a high search frequency on an arbitrary day, its inconclusive whether that spike is a result of the new iPhone being released (good connotation) or the result of an issue or recall with the iPhone (bad connotation). Having more richly-annotated data would likely provide more informative results.

In addition to predicting the future stock net change, we trained models to predict the other stock characteristics as well. The same data set previously discussed was used and the prediction variable was changed. Fig. 5 shows that only 2-day net price change and 50-day moving average price could be predicted with better than 50% performance. This result is expected since these two metrics have a moving window that filters noise and therefore easier to predict.

IV. CONCLUSIONS

As expected, machine learning algorithms could only predict slightly better than 50% error, but further evaluation is necessary to determine whether following this prediction is profitable when compared to an index fund. Shorter time horizons were easier to predict but less valuable financially. Google Trends had negligible impact on the performance of the algorithms which may

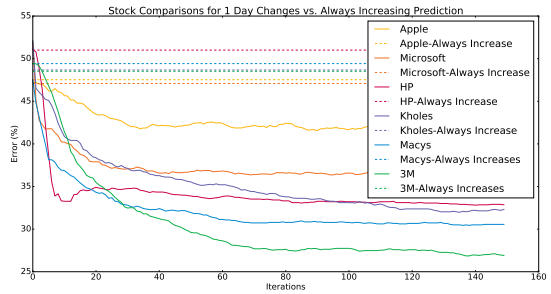

Fig. 3. The lower and upper limit companies are 3M and Apple respectively. The dotted lines are the always increase baseline predictive strategy.

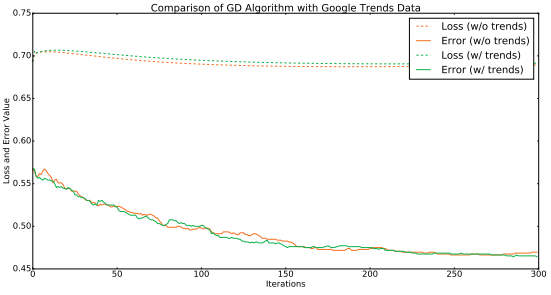

Fig. 4. Google Trend data for Apple search terms did not show any improved results for stock prediction. Here, we use the search-per-day value of 'apple', 'ipad', 'iphone', and 'ipod'.

be attributed to the lack of information on associated connotation for a given keyword search. Each algorithm

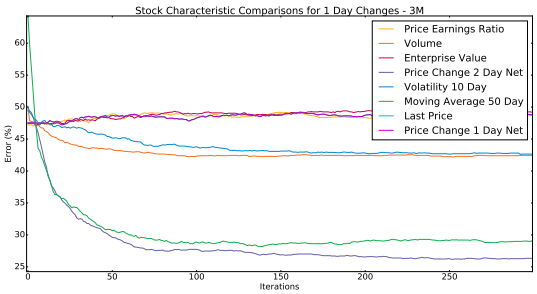

Fig. 5. A comparison of models trained to predict different stock characteristics. Only 2-day net price change and 50-day moving average price could be predicted better than 50%.

produced comparable results and the 2-day net and the 50-day moving average were the only two categories that predicted above a 65% success rate.

V. Future Work

Using external data such as Google Trends could help make a more informed and higher accuracy prediction of stocks. More thorough analysis is required as to what parameters have impact that do not overfit to any one company. Additional work can be done to determine profitability of short-term stock prediction as our results indicate some degree of success in comparison to our baseline. To determine profitability we must take into account the value of the loss in dollars when our prediction is incorrect and make judgments on thresholds of when to buy, sell, or hold a stock.

APPENDIX

ACKNOWLEDGMENT

The authors would like to acknowledge Professor Zico Kolter, Professor Tuomas Sandholm and the Spring 2016 15-780 course TAs.

References

[1] ”BlackScholes model.” Wikipedia.
[2] ”Brownian model of financial markets.” Wikipedia.
[3] Y. Dai, Y. Zhang. ”Machine Learning in Stock Price Trend Forecasting.” Stanford School of Computer Science Course Project, 2013.
[4] A. Siripurapu. ”Convolutional Networks for Stock Trading.” Stanford School of Computer Science Course Project, 2014.