Machine Learning Engineer Nanodegree

Capstone Project

Kevin Mann
September 19th, 2017

Definition

Project Overview

Participation in successful stock trading for the general public, with only limited exceptions, involves future price predictions. These predictions may stand alone as a single decision factor or play into a larger and more complex trading strategy. This project focuses on the price prediction of publicly listed stocks using a neural net architecture trained on historical data.

Despite the massive increase over the past two decades in computing resources dedicated to trading and the contextual time frame for many computer trading decisions shrinking into microseconds, there remain two major approaches that human practitioners follow when forming securities price predictions:

- Fundamental Analysis: The use of financial and economic data at the company, sector, and market level to create price forecasts.
- Technical Analysis: The use of asset price and volume history to create price forecasts.

Fundamental analysts spend time researching a company, sector, and the overall market to determine a price estimate. There is usually a lot of research and estimation of major factors such as revenues, costs, and interest rates that are input into a financial model. They are usually familiar with accounting, finance, and economics, and will often do much of their work with spreadsheets that are custom built with financial models for inputting performance expectations. This process takes time to do well and so a single fundamental analyst may have current models built for only a few stocks at a time.

Technical analysts study the price and volume history of stocks and use these metrics to quantitatively interpret the underlying forces driving the stock's performance. They rely in part on repeating psychological tendencies of traders which cause asset prices to form recognizable patterns over time. For example, they are familiar with formations that signify tops, bottoms, and consolidations that will give clues to a stock's future direction. They will do much of their work analyzing stock price graphs by using drawing tools (digital tools in the present era), and there are technical indicators (calculations based on price and volume history) that they will add to the chart to give more clues to the future. A

technical analyst can analyze stocks comparatively quickly because there is essentially no background research involved. They can create well-formed opinions on a very large number of stocks within a short period of time and also use computing power to perform parts of the analysis.

Problem Statement

This project takes a regressive, technical approach to stock price prediction, using a combined model architecture of Convolutional Neural Net (CNN) and pooling layers, followed by Recurrent Neural Net (RNN) layers of the Long Short Term Memory (LSTM) variety, and finalized with fully connected neural net layers that results in a stock price prediction.

The script will be executed from the command line, and will prompt the user for any number of tickers, a start date and end date to set the training bounds, and a number of days into the future that the model will predict. The program will do the following with the user-provided information:

1. Retrieve stock price and volume information from a publicly available source.
2. Remove unnecessary columns, leaving only Adjusted Open, Adjusted High, Adjusted Low, Adjusted Close, and Adjusted Volume in the data.
3. Add calculated indicators that will help the model forecast the future price more effectively.
4. Frame the data into time slices that serve as “images” to be fed into the model.
5. Create a combined neural net model and fit it to the data.
6. Print the accuracy of the analysis and provide a future stock price prediction as requested by the user.

Metrics

The metrics used to determine the quality of the program and fitted model are mean error of prediction, mean absolute error of prediction, and standard deviation of prediction errors. The calculation for each are as follows:

$$\text{Mean Error of Predictions: } \frac{\sum \textit{Prediction Errors as Percent}}{\textit{Count of Prediction Errors}}$$

$$\text{Mean Absolute Error of Predictions: } \frac{\sum |\textit{Prediction Errors as Percent}|}{\textit{Count of Prediction Errors}}$$

$$\text{Standard Deviation of Prediction Errors: } \textit{StdDev}\left(\textit{Prediction Errors as Percent}\right)$$

Having a low Mean Error of Prediction means that the prediction errors are centered around the actual value. A low Mean Absolute Error of Prediction means that errors on average are low. A low Standard Deviation of Prediction Errors means that the predictions are generally close to the actual value.

Each of these is appropriate for a regression predicting a value on a continuous scale. The price predictor will produce several hundred predictions for test and validation data and these statistical aggregates will provide a high-level assessment of the characteristics of the result errors.

Analysis

Data Exploration

The data source used in the program is Quandl, but many public data sources for stock prices exist. Some of the premium data sources require a fee, but daily price and volume data for publicly listed companies can be found without charge. A production system should use a professional data source because flaws are reputedly common in freely available data, but for prototyping a predictor that is not used in actual trading, this data source is quite sufficient.

Quandl provides both unaltered daily Open, High, Low, Close, and Volume metrics and adjusted amounts for each, which adjusts for the effects of dividends, splits, etc., and Ex-Dividend and Split Ratio columns. The Adjusted Close price is the value is that is to be predicted. Every active trading day should have a single entry for each of these columns; weekends and holidays are excluded.

Date	Open	High	Low	Close	Volume	Ex-Dividend	Split Ratio	Adj. Open	Adj. High	Adj. Low	Adj. Close	Adj. Volume
2011-01-03	325.6400	330.2600	324.8365	329.570	15897800.0	0.00	1.0	41.849279	42.443013	41.746018	42.354338	111284600.0
2011-01-04	332.4400	332.5000	328.1500	331.290	11038600.0	0.00	1.0	42.723173	42.730884	42.171849	42.575382	77270200.0
2011-01-05	329.5500	334.3400	329.5000	334.000	9125700.0	0.00	1.0	42.351768	42.967350	42.345342	42.923655	63879900.0
2011-01-06	334.7194	335.2500	332.9000	333.730	10729600.0	0.00	1.0	43.016108	43.084298	42.782290	42.888956	75107200.0
2011-01-07	333.9900	336.3500	331.9000	336.120	11140400.0	0.00	1.0	42.922370	43.225663	42.653776	43.196105	77982800.0
2011-01-10	338.8300	343.2300	337.1700	342.455	16020000.0	0.00	1.0	43.544377	44.109839	43.331044	44.010241	112140000.0
2011-01-11	344.8800	344.9600	339.4700	341.640	15861000.0	0.00	1.0	44.321887	44.332168	43.626626	43.905502	111027000.0
2011-01-12	343.2500	344.4300	342.0000	344.420	10806800.0	0.00	1.0	44.112409	44.264056	43.951767	44.262770	75647600.0
2011-01-13	345.1600	346.6400	343.8500	345.680	10599300.0	0.00	1.0	44.357871	44.548071	44.189517	44.424698	74195100.0
2011-01-14	345.8900	348.4800	344.4400	348.480	11030000.0	0.00	1.0	44.451686	44.784537	44.265341	44.784537	77210000.0
2011-01-18	329.5200	344.7625	326.0000	340.650	67178500.0	0.00	1.0	42.347913	44.306786	41.895544	43.778273	470249500.0

Illustration 1: Raw data from Quandl for Apple Inc. (Ticker: AAPL)

Price data represents the amount of US dollars that are paid in exchange for a single share of stock. Most stock prices are less than $500, but there is no upper limit. The open price is the first price paid

for the day, the high price is the highest price paid during the day, the low price is the lowest price paid during the day, and the close price is the final price paid for the day.

Adjusted values result from making corrections to prior data after events that cause shifts in the stock price that are independent of the underlying value. For example, if a stock has 1 million shares outstanding at a price of $10 per share and performs a 2:1 split so that there are now 2 million shares trading at a price of $5 per share (the total value, market capitalization, of $10 million staying the same), the unadjusted price would show a drop of 50% that should be adjusted out to show the true price movement. The adjustment will divide all pre-split prices by the scale of the increase in number of shares, in this case dividing all pre-split prices by 2.

The price data can be considered continuous and is always positive. Prices move in upward and downward trends, usually with a rising tendency over time. Training periods of 7 years or less will exclude the large drop in fall 2008 through spring 2009 and every price column for most stocks will tend to have an upward slope with only short periods of correction during that time. The recent data for most stocks will have them at the upper bound of the range, since the stock market is currently near all time highs.

Volume represents the total number of shares traded within a single day. Volume data is comparatively flat over time but also will move in trends and with spikes. Spikes in volume are a more frequent occurrence than in price data. Volume typically has values in the millions of shares for large companies.

The data arriving from Quandl is assumed to be clean without need for alteration, unless problems are encountered during the model training. There is no need for isolated corrections to the data, and doing so would compromise its integrity.

Exploratory View

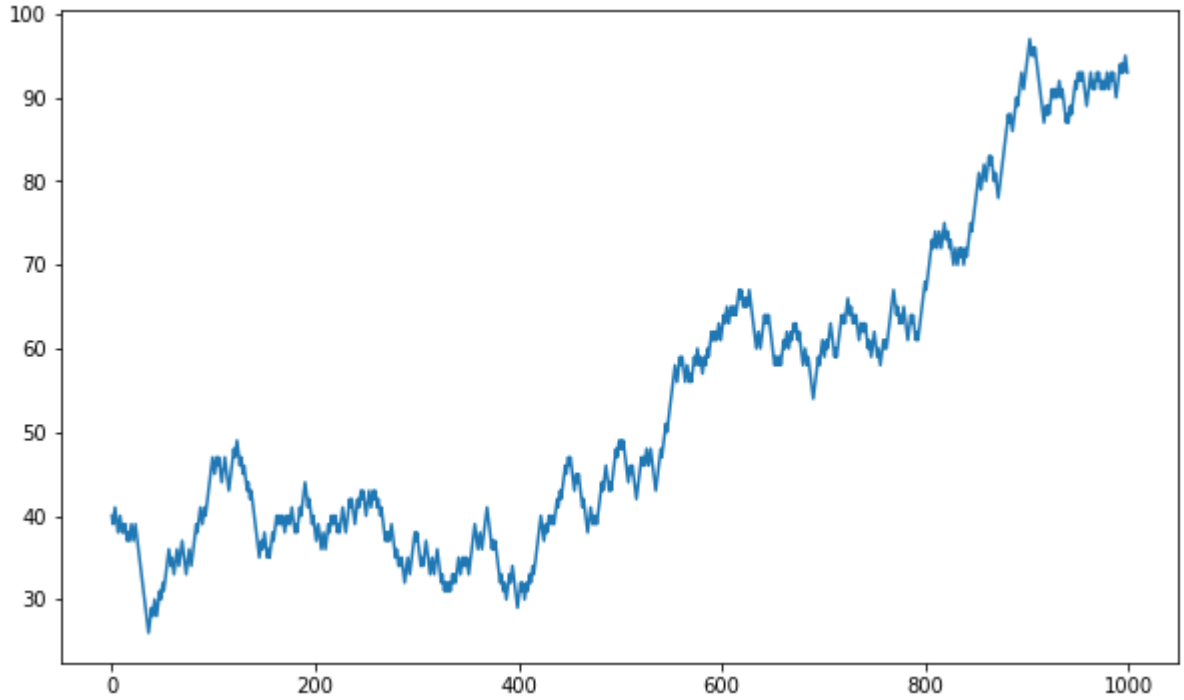

Illustration 2: A random walk starting at a value of 40, with equal probability of increment or decrement at each step

The price movements of many stocks tend to resemble a random walk, since a stock's price movements has intermingled elements of both trend-following and mean-reversion, but have a much more complex explanation, which is the thousands of interactions between buyers and sellers (both humans and machine agents) that determine transaction prices.

The Random Walk Hypothesis states that stock prices change according to a random walk and thus cannot be predicted and any profits are a result of chance. This is a widely known but extreme and pessimistic view that is generally disproved by investors that consistently outperform the market and trading strategies and techniques that have been shown to be reliable over many decades.

Algorithms and Techniques

The predictor uses a multi-layer neural network composed of two initial Convolutional Neural Networks with Max Pooling layers that feed into two Long Short Term Memory (LSTM) Recurrent Neural Network (RNN) layers, and finally has two fully connected neural network layers.

The CNN is the best performing neural network architecture for image recognition, by learning patterns in the data which define the characteristics. This is a good analog for human-based technical analysis, which involves pattern recognition but with a time component.

The LSTM layers add that time component, being good at time-series analysis by keeping a memory of former events. As an example, LSTM architectures are used in speech recognition where the timing of events tend to vary.

The combination of the two types of neural network layers provides both pattern recognition and memory components that correspond to human-based technical analysis.

Finally, two fully connected layers are appended, the penultimate layer for additional model learning flexibility and the output layer for the numerical prediction.

Data is converted to a form that is appropriate for the architecture, and the model processes the data in epochs attempting to minimize a loss function. The parameters of the data and model were set and tested in an iterative manner. After several iterations testing various combinations of values for the parameters and after the model has reached an acceptable accuracy on the training data, the model is tested on unseen test data to determine if it meets the benchmark goals.

Benchmark

The metric goal amounts for this project follow, each for the case where the price is being predicted 21 trading days into the future (one trading month, on average), with training dates of 10 years ago to present, and using the average results from the top 10 components of the S&P 500.

- Mean Error of Test Predictions: < 1%
- Mean Absolute Error of Test Predictions: < 5%
- Standard Deviation of Test Prediction Errors: < 5%

Methodology

Data Preprocessing

All columns are removed from the Quandl data except the following:

- Adjusted Open
- Adjusted High
- Adjusted Low

- Adjusted Close
- Adjusted Volume

Following this initial step, the following technical indicators, calculated on the Adjusted Close price, are added to the data prior to reshaping the data as frames:

- Price change over periods of time
- Exponential moving averages with varying spans
- Percent price change over periods of time

The periods of time and spans for the indicators are the Fibonacci numbers between 13 and 144 (inclusive), chosen because the sequence frequently occurs in market patterns and technical analysis, and provides a good variation of time frames. Smaller and larger numbers from the sequence were eliminated during testing.

Having the data in the form of a single row entry per date, where each date is associated with the above columns and indicators, is not adequate for the designed architecture. CNNs learn patterns in images, and LSTMs learn changes over time. To get the data into a form that would be used effectively by both, I split the input data into frames of time. Each date serves as the end of a frame, and each date with its corresponding data will repeat in a number of frames equal to the frame length (these statements exclude dates at the start within a frame length to the beginning, since a full frame would be impossible). All frames are of the same size and represent overlapping, rectangular “images” of stock data.

In the process of creating each frame of input data, the data was scaled using Scikit-Learn’s Standard Scaler, which scales each column to have a mean of 0 and a standard deviation of 1.

The target variable is the ratio of adjusted close between the target date and the last date in the frame. Two values are stored as a row in the dependent variable array: the first entry is the change multiplier (ratio of target price over current price, current price being the last entry for each frame) and is the target value for the model to learn, and the second entry is the scaling factor (the original divisor, current price) to convert the predictions back into stock price.

Implementation

A usable script includes user input, data retrieval, data processing, modeling, prediction, and performance reporting. These functions provide the user interface aspect of the script:

- get_ticker(): Requests the user to enter one ticker at a time. A ticker is not tested for validity but will be skipped if Quandl is not able to return data. Multiple tickers can be entered, but only one at a time and after confirming a prompt.
- get_start_date(): Requests the start date for training. Defaults to 8 years ago. The date entered here may be moved forward to the earliest trading day for the stock by Quandl when retrieving the data. The start date is the same for all tickers entered.
- get_end_date(): Requests the end date for training. Defaults to None, which will be interpreted by Quandl as the most recently completed trading day. The end date is the same for all tickers entered.
- get_advance(): The number of days to predict in advance. The advance is the same for all tickers entered.

The data retrieval and processing takes place next. The data is queried from Quandl, trimmed of unused columns, indicators are added, and the data is framed and scaled. These steps are implemented in:

- get_data(ticker, start_date, end_date, use_columns, predict_column, frame_length, advance): Returns independent (x) data in a 3D array and dependent (y) data in 2D array.

The processed dataset is then split into training, testing, and validation sets. Training, validation and testing datasets are chosen at random from the user-specified period in the following proportions:

- Training: 60%
- Validation: 20%
- Testing: 20%

Splitting the data at random was chosen because stock behavior changes over time, and a training set that intentionally excludes the most recent data would not be a reliable predictor for future values. The trained model should be exposed to as much data as possible within the entire training period. For back testing a trading strategy based on this model, only sections of data prior to the date of prediction could be used for training without introducing a look-forward bias.

The first few attempts at fitting a model that met benchmarks involved non-neural net implementations available in Scikit-Learn. When none of these algorithms produced results sufficient to meet benchmarks, a neural net approach was taken. A feed-forward neural net was built and tested but inadequate, and adding a CNN layer did not meet benchmarks. By researching I found suggestions that LSTMs are effective at learning time series data, and by combining CNNs with LSTMs the model would possess both pattern recognition and time series memory capabilities. The neural net architecture was implemented with various layers and parameters being tested. Keras was used to build the architecture, because the implementation details are well encapsulated by the library and modifications are made easily.

Model checkpoints are used to save and retrieve the best model from training, and early stopping is used to prevent excessively long training times. For each ticker specified by the user, the model is fit over a maximum 256 epochs and batch size of 2, and when finished returns both the best fit model as determined by the mean square error of the validation data and the loss history for validation data.

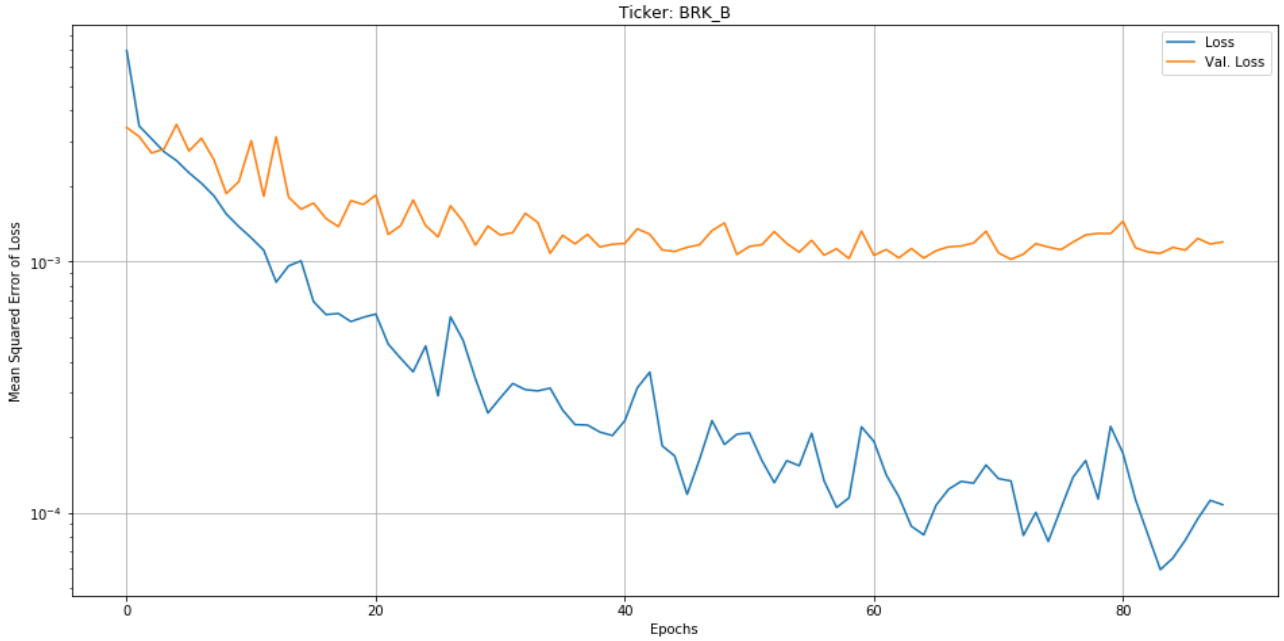

Illustration 3: Model fitting for Berkshire Hathaway, Inc. Class B (Ticker: BRK_B), stopped early after 88 epochs

The approved project proposal for this capstone explained goal amounts in terms of percent return per year, specifically a goal amount of 9% per year for a regressive predictor. As I worked deeper into the project, I realized that this is an inappropriate measure for a stock price predictor, some stocks will easily exceed this with a low quality predictor, while other stocks could not produce this return amount with a high quality predictor and well executed long/short strategy.

Therefore, I decided to more closely follow the suggested project benchmark available in the capstone resources on Udacity, but with a more ambitious length of prediction. The capstone resources suggest a target of +/- 5% on average over a 7 day period (presumably 5 trading days). However, a 5%+ change in 1 week is a rare occurrence for most stocks. I decided instead that a one month time frame is a reasonable time period for a retail investor to reassess investment allocations according to expectations, and I kept the target average absolute error unchanged at 5%. In addition, I wanted the average error to be symmetrically distributed around 0% and the standard deviation to be commensurate, so I concluded the above benchmarks would suffice.

Refinement

An initial solution producing values sufficient to meet benchmarks involved the three types of neural network layers (CNN, LSTM, and Dense). The initial model architecture was the following:

1. Conv1D(filters=64, filter_size=2, activation='relu', padding='same')
2. MaxPool1D(pool_size=2, strides=2, padding='valid')
3. LSTM(neurons=32)
4. Dense(neurons=1)

To refine the final model architecture, additional layers were added and tested, with the parameters tuned based on trial performance on stocks chosen from the top 10 components of the S&P 500. Tuning was focused on those stocks that were most difficult to predict, as determined by the validation accuracy. The number and type of layers was determined by adding and removing layers with parameters that were determined to be most effective in the simple model. Afterward, a grid search technique was used to tune the layers' parameters.

In building a model that came close to meeting benchmarks, I used actual prices as the dependent variable until I noticed that the values being predicted would never exceed the high or low from the training data for that stock. I determined it was caused by the model being unable to predict any value higher or lower than had been seen in training. This meant the model was completely unable to predict new highs or lows. I therefore switched to predicting the percentage change rather than the actual price. This made an additional column necessary in the target variable, namely the scale amount as discussed under Data Preprocessing, to allow conversion back to prices.

In considering the issue with scaling the data to be able to predict new highs, scaling the independent data also became necessary. I tried scaling the independent data by dividing all framed data by the last closing price within each frame, but the model did not learn well. I found that by scaling each frame with Scikit-Learn's Standard Scaler, the model learning was faster and the results were better. Each frame is fit and transformed by the scaler, and since no inverse transformation is necessary, the scaler parameters do not need to be retained.

The final model is built with these layers (non-listed parameters are left as Keras defaults):

1. Conv1D (filters=256, kernel_size=3, activation='relu', padding='same')
2. MaxPooling1D (pool_size=2, strides=2, padding='same')
3. Conv1D (filters=128, kernel_size=3, activation='relu', padding='same')
4. MaxPooling1D (pool_size=2, strides=2, padding='same')

5. LSTM (neurons=64)
6. LSTM (neurons=32)
7. Dense (neurons=4)
8. Dense (neurons=1)

The model is compiled with these parameters:

- model.compile (loss='mean_squared_error', optimizer='adam')

This loss calculation is appropriate for a regressive predictor, and the optimizer was chosen by analyzing results. Leaving the default optimizer parameters gave the best validation results. The 'rmsprop' optimizer also worked for most training sessions, but would occasionally stop training beyond a certain validation accuracy.

Results

Model Evaluation and Validation

The final model predicts prices with a good degree of accuracy. After settling on a final model, the robustness was tested by using the top 10 components of the S&P 500. All performed within the benchmark range, and visually, the price predictor is doing a good job to fit the data.

Illustration 4: Comparison of actual versus predicted for Johnson & Johnson (Ticker: JNJ)

The time taken to fit the model to each stock is longer than I would like for a good user experience, but considering this model would be expected to run at most once per day per stock, it is not a terrible amount of time to wait.

The model produces future predictions that appear reasonable. However, when looking closely at the predictions when compared to actuals, the model predictions move more suddenly. In general, the model has elements of trend-following and mean-reversion which are typical characteristics of stock price movements and therefore good to have in a predictor.

Illustration 5: Alphabet Inc. Class C (Ticker: GOOG) predicted into the future.

This model might serve well as one component of a trading system and could be improved in time, but as it stands, I would not suggest basing any real money decisions on it.

Justification

The final average metrics for the solution as tested on the top 10 components of the S&P 500, predicting with an advance of 21 days and training period of the most recent 10 years are:

- Mean Error of Predictions: 0.00% (Benchmark: < 1%)
- Absolute Mean Error of Predictions: 2.67% (Benchmark: < 5%)
- Standard Deviation of Prediction Errors: 3.72% (Benchmark: < 5%)

Conclusion

Free-Form Visualization

When comparing the fitted models to a naive predictor (one that flatly predicts the current price into the future) it can be seen that the model both greatly reduces variability of prediction errors and more closely centers the average around zero.

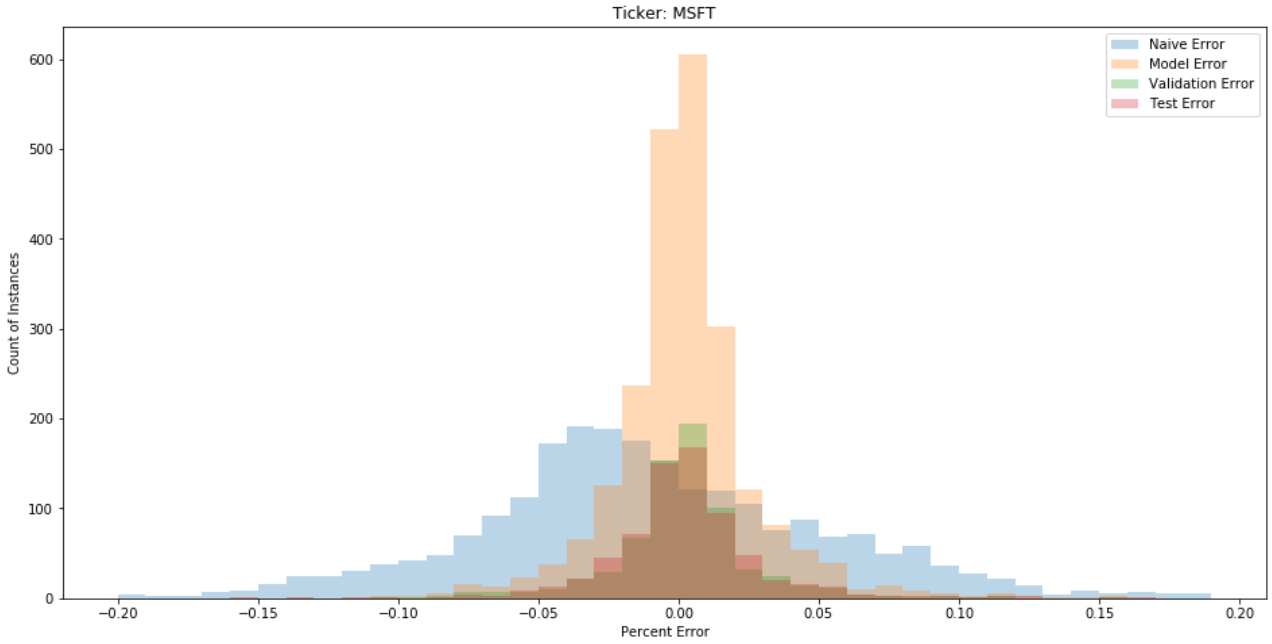

Illustration 6: Histogram comparing error percentages for Microsoft Corporation (Ticker: MSFT).

In Illustration 6, a comparison of a naive predictor is compared with model results for Microsoft Corporation (Ticker: MSFT). The aggregate accuracies are reported here:

Naive Accuracy ========== Mean: -0.8547% Abs. Mean: 5.0633% Std. Dev.: 6.4523%	Validation Accuracy ========== Mean: 0.1872% Abs. Mean: 1.7192% Std. Dev.: 2.6294%
Model Accuracy ========== Mean: 0.2388% Abs. Mean: 1.7346% Std. Dev.: 2.6942%	Test Accuracy ========== Mean: 0.2346% Abs. Mean: 1.8711% Std. Dev.: 2.9442%

The naive predictor consistently underestimates the gains, and has a wide dispersal of errors. Even with a constant prediction of a certain percentage increase per period to offset the displaced mean, the width of the naive predictor would stay unchanged because there is no variability being anticipated. The model, on the other hand, has an average error very close to centered, and greatly reduces the variability of prediction errors.

Reflection

The overall process involved researching possible algorithms, performing data preprocessing to fit the expected input shapes, and testing the algorithms to see if appropriate results were obtained. It was surprisingly difficult to find an algorithm that was able to come close to the benchmarks, though when looking back it makes sense that most or all simple approaches have probably already been exploited by players in the markets and thereby have been priced out of the market. Once I located an algorithm that came close to meeting goals, an organized tuning approach was the last step to maximize the accuracy. To finalize the usability, it was necessary to add functions allowing user input and preparing and formatting output to the terminal.

Keras offers a powerful encapsulation of neural network approaches and I was able to try different arrangements quickly in order to locate a superior model. Building the components of a neural net from scratch is educational, but unnecessary and wasteful with an offering such as Keras. However, I probably did gain the most intuition from building a feed-forward neural network on my own after I decided that Scikit-Learn’s algorithms would require to much time spent in feature engineering. I also learned a lot about technical analysis in the time I spent investigating technical indicators to add to the dataset as features.

The final model falls a bit short of my expectations for the project, but after putting in the work, I know this has been a good way to gain experience with the more advanced machine learning algorithms and the difficulty of predicting stock prices accurately. I would not recommend using this predictor with real money, but I believe the underlying architecture is sound and with some additional tuning, testing, and feature engineering, the program could be a useful addition to a trading system.

Improvements

In order to feel confident incorporating this model into an investment program, additional accuracy would need to be achieved. Additional tuning of hyperparameters and architecture should produce enhanced results. Running the model on a GPU-based system should both lessen training time and time

needed to produce results for the user, which would allow more epochs to be run, perhaps finding more accuracy.

I believe it likely that intensive feature engineering would be needed to enhance the accuracy to production-ready results. The difficulty in finding a reliably accurate predictive model highlights the challenge of stock market prediction.

Data augmentation for the convolutional layers may produce enhanced results, considering the flexible nature of stock market events, and the unique nature of each stock's behavior. As a similar idea, it is possible that tuned hyperparameters for each stock would be appropriate, instead of tuning for the best general performance. The time necessary to train the model as it is prevented such an approach on my machine.

The program itself needs some application of software engineering principles to ensure it works correctly without issue. It being a prototype, I did not focus on the peripheral details, rather focusing on the architecture. Error handling, proper documentation, general cleaning of the code, etc. would be necessary to put this into production.

From a user perspective, this model would benefit from being incorporated into a larger program with enhanced user interface. I can imagine a model like this being used on a web service that might have enough demand to charge a subscription fee, or possibly being used in an investment publication. The future of the markets will likely continue to become more technology focused and automated for many investors.