Submission #1

Stock Price Predictor

Machine Learning Nanodegree Capstone Project

By: Faisal Al-Tameemi

Definition

1

In this section, we will define the problem, state the hypothesis behind our approach and the metrics and benchmarks to be used for evaluation.

Problem Overview

Investment firms, hedge funds and even individuals have been using financial models to better understand market behaviour and make profitable investments and trades. A wealth of information is available in the form of historical stock prices and company performance data, suitable for machine learning algorithms to process.

In this report, we will discuss the stock price predictor algorithm built; it takes daily trading data over a certain date range as input, and outputs projected estimates for given set of dates. Stock prices (inputs) will contain multiple metrics, such as opening price (Open), highest price the stock traded at (High), how many stocks were traded (Volume) and closing price adjusted for stock splits and dividends (Adjusted Close); our application will aim to predict the Adjusted Close Price. More on this in the next section (Analysis).

Data for all SP500 indices listed on Wikipedia will be downloaded from Quandl, a financial data service, and saved into a Postgres database.

We will be building a web app which will serve as a UI for interacting with the training and query interfaces handled by a separate python server (with Flask). We will dive into the stack of the project in more detail in Section 3 (Methodology) of this report.

Problem Statement

As we will predicting the numeric prices of each stock, we will need to use regression machine learning algorithms. If we were predicting a label such as the direction of the a stock price change (up or down), then it would be a classification problem.

Since the stock price data is in time-series format (continuous entries over time), we will have to apply certain transformations to make sure that it's in a format which our machine learning algorithm can accept.

This means we would have to train the algorithm on historic data (lags) or future data (horizons) with today's adjusted close prices.

Another possible problem is missing data. Since a given stock price can have zero activity trading days during the year (i.e. *there are days where no trading happens for certain stocks*). We will use **linear interpolation** and **mean interpolation** to fill in the gaps.

Metrics

In this implementation, our desired goal is to predict a price. As mentioned, this is a regression problem. The available regression metrics are:

- R^2 **score**
- **Mean Squared Error**

$$1/n \sum_{i=1}^{n} (prediction_i - actual_i)^2$$

- **Mean Absolute Percentage Error**

$$1/n \sum_{i=1}^{n} (|prediction_i - actual_i| \ / \ actual_i) * 100$$

Since our main goal to get as close as possible as the real adjusted close price for that day. We can measure how close our predictions are to the real price by using a **mean_absolute_percentage_error** (see formula above)**.**

This metric is useful since we're trying to have a clear understanding of how far off our prediction is. We can also use the **mean_squared_error** metric to find the dollar amount difference between our prediction and the actual price.

Another approach we can take is to convert our numerical predictions as well as the actual prices into percentage change values and then into binary labels (up or down). Doing so would allow us to use classification metrics such as:

- **Precision**
- **Recall**
- **F1-score**
- **Support**

Analysis

2

In this section, we will take a deep dive into the problem and explore data to uncover insights about how we could make the required predictions. We will also discuss the algorithms and benchmarks used in this project.

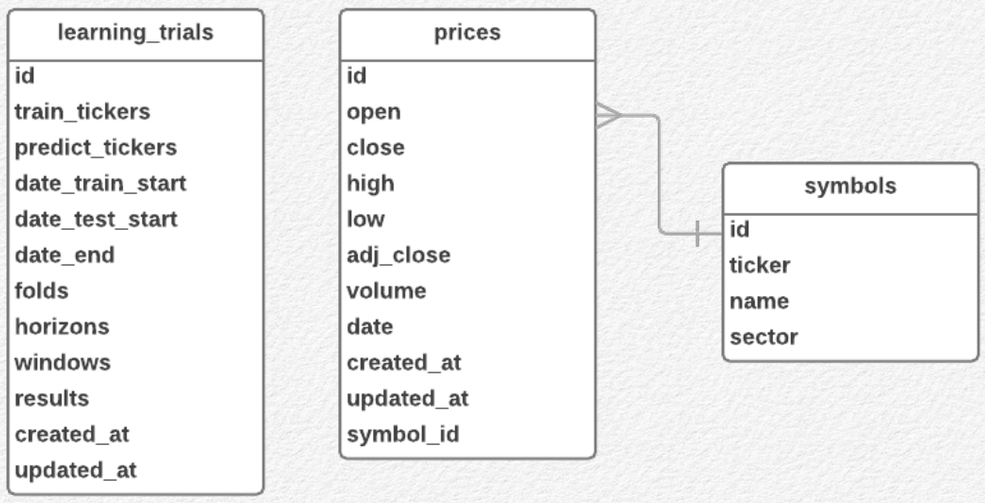
Data

Dataset

In order to download the required data for this project, a script was created to crawl a Wikipedia page containing a list of all the SP500 companies (*reference 2.1*). Once this page is crawled, we proceed to insert each symbol and it's name into the database.

Once the symbols are available, the script will proceed to download each symbols daily prices from Quandl in JSON format (*reference 2.2*). Once downloaded, the data is inserted in the database. This is done to avoid over-usage of the Quandl API.

The data is stored into multiple tables but can be extracted into a single dataframe by using SQL joins between the prices table and symbols table. Below is the structure of the database:



The **prices** and **symbols** will keep the data regarding stock prices while the **learning_trials** will house the history of all prediction trials requested via the API.

Data Exploration

Once the seed script is done (only in development mode), we can start to load the data from the database directly into a pandas dataframe (*see reference 2.3 for the database query*). Below is a sample of the data:

date	adj_close	volume	high	low
2016-06-01	719.44	3252707	726.43	718.22
2016-05-31	722.79	3537039	724.23	711.32
2016-05-27	712.24	2226407	716.6	711.1

The datatypes for each of the fields in the samples above are as follows:

- **ticker**: String
- **volume**: Big Int
- **adjusted_close**: Float(4)
- **date**: Date

Other fields such as **open**, **high**, **low** and **close** can be removed as keeping them didn't have an significant impact on the algorithm.

As we comb through the data, we notice that some stocks have missing values for some dates. This means, we will have to be careful when joining data from one stock onto the other. We will also have to use a method or more to interpolate the values of the missing fields.

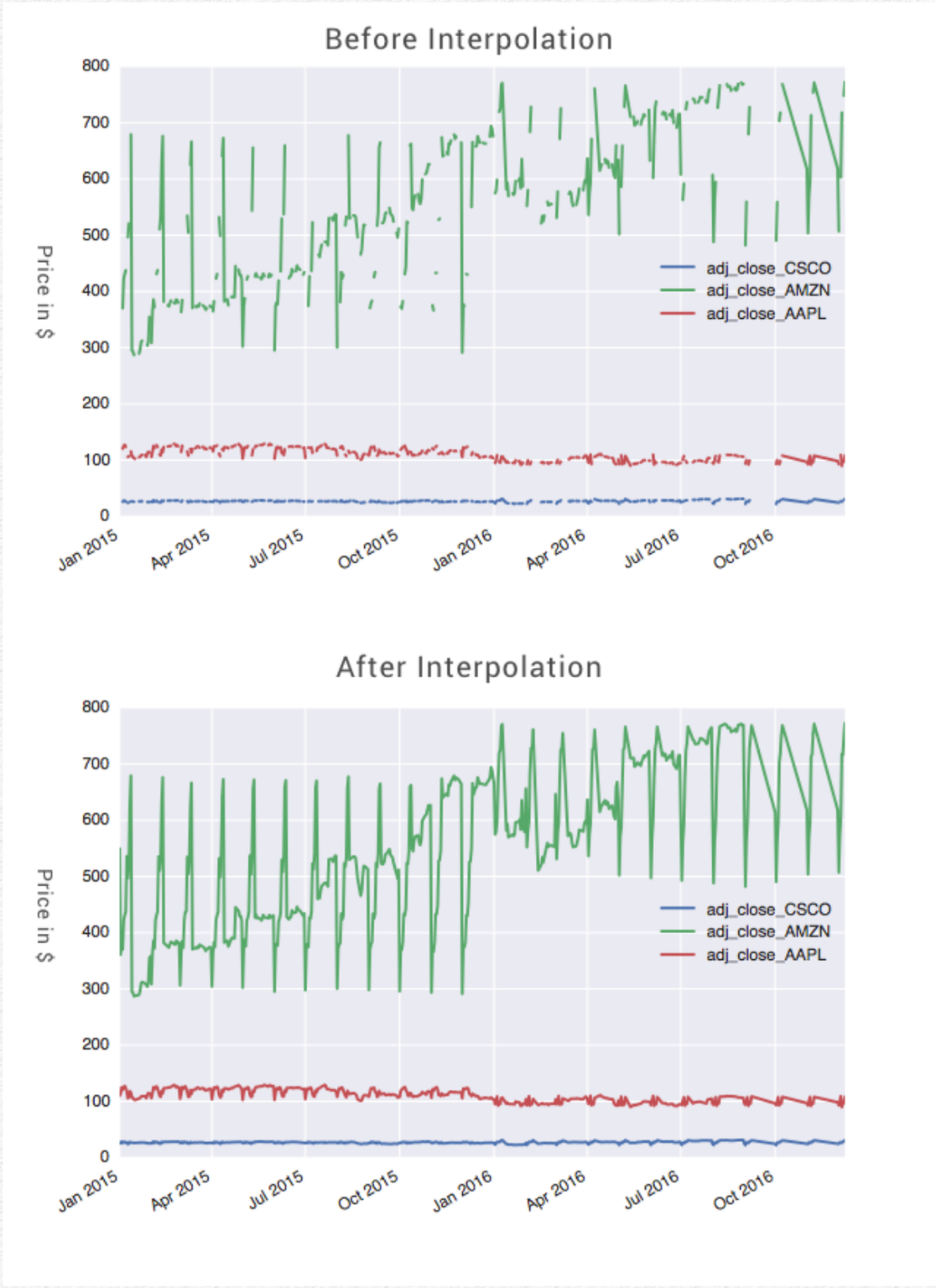
Some techniques for interpolation in time-series data include:

1. **Backfilling** / **Frontfilling** values

2. Using the rolling mean of that period (i.e. **mean interpolation**)

3. Using **linear interpolation**

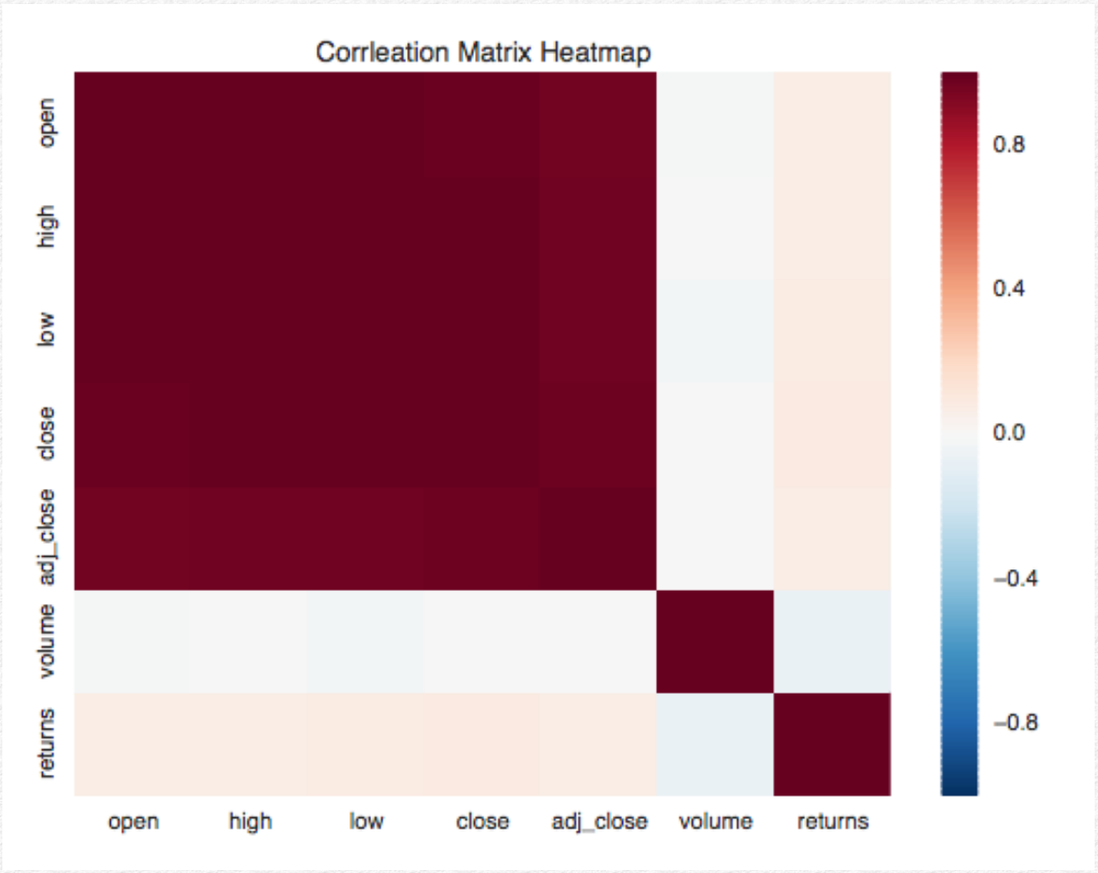
The graphs on the right show adjusted closing price of 3 stock indicators (Amazon, Cisco and Apple) before and after using linear interpolation and mean interpolation.



Exploratory Visualizations

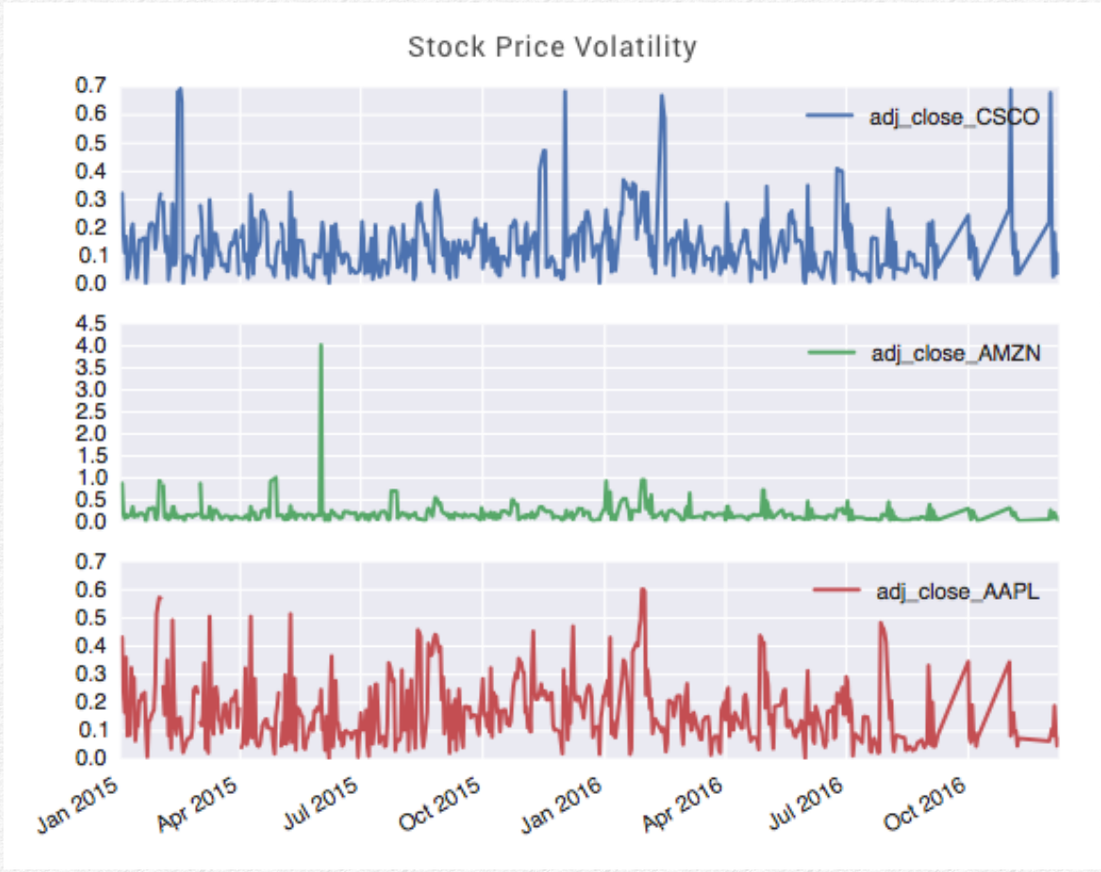
One of the most useful visualizations we can start with is a heatmap of the correlation between the main features for each ticker. This will give us insights into which features we can possibly drop without major consequence to the final predictions.

Below is an example of Cisco's stock correlation heatmap:



We observe that there is a high correlation between the features **open**, **high**, **low**, **close** and **adj_close** which is expected since these values should be close to one another. This also means we can drop all but one of them, **adj_close**, since it's the feature we are trying to predict. We will discuss this further in the next chapter (Methodology).

Another way we can explore the information at hand is to explore the volatility of a given stock's adjusted close price (**adj_close** feature). Below is a graph of the volatilities of the three stocks explored previously (AMZN, CSCO, AAPL):



Benchmark

The benchmark selected for this implementation of the stock price prediction is an average of **5%** (or less) **error between actual and predicted stock prices**.

We arrive to this value simply by choosing a statistically significant error percentage.

We select this benchmark because it allows us to easily compare performance across different prediction trials with different stocks (i.e. it's easy to compare with a % err but not with a metric such as R^2).

If we were working towards a classification of the direction of the stock. We would have a hypothesis that a precision score of 57% to 60% would be acceptable for the problem of prediction stock prices.

Methodology

3

In this section, we will discuss the detailed steps taken to build the stock prediction models.

These steps include data preprocessing, implementation and refinements.

Data Preprocessing

Once each ticker's data is loaded from the database for the specified date range, it's inserted into a pandas dataframe (i.e. data table). Each dataframe (containing a single ticker's data) is then concatenated into a single dataframe.

The combined dataframe is then combed through and has its NA values filled using 2 methods (*reference 3.1*), **linear interpolation** and **mean interpolation**.

Once we've interpolated the missing data, we can start to add more features (columns) to our dataset based on some preliminary analysis. These features include:

- The **rolling mean** of each ticker's **adj_close** feature (i.e. *Adjusted Closing Price column*)
- The horizon (or lag) columns. These can be created by adding new columns with the shifted **adj_close** values for each ticker. For example, if we would like to predict 5 trading days into the future, we can create 4 new columns for each ticker, containing the prices for the next 4 days (relative to each row/day).

In the implementation provided, these columns are created for multiple horizons and rolling mean windows. Each version of the preprocessing steps is then saved into a CSV file to be loaded during the Cross Validation steps. This step is taken to avoid having large datasets loaded directly into memory.

In other words, the data processing steps will output multiple version of the data each with a different number of new features added based on the windows and horizons specified.

A summary of the steps taken during the data preprocessing stage are as follows:

1. Load data for each ticker from the database (or CSV file)
2. Combine each ticker's data into a single dataframe indexed by all the days specified in the input date range
3. Interpolate missing data for all stocks
4. Add new features for each ticker to represent the rolling mean and horizons (i.e. future prices) or lags (i.e. past prices)
5. Save the final dataframe into a CSV file
6. Split the data into a training and a testing set

Implementation

Several algorithms were selected in the first version of the implementation in order to evaluate which one performs best.

Since we are predicting the price (i.e. a number) of stocks, we have to look into regressor algorithms. We also have to use multi-output support in order to predict multiple outputs (i.e. multiple stock prices at once). We will be using the **MultiOutputRegressor** class from sklearn to achieve this.

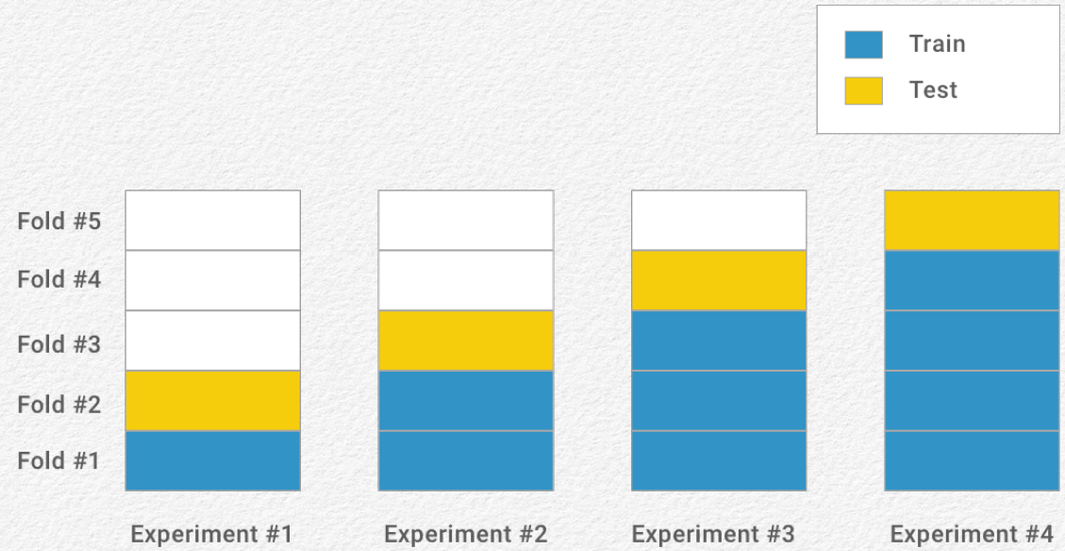
The algorithms chosen to build models are:

- **SGD Regressor**
- **Support Vector Regressor** (SVR)
- Ensemble Methods
    - **AdaBoost Regressor**
    - **Gradient Boosted Trees Regressor**

Another important factor to consider during the implementation is cross validation. In order to correctly apply cross validation with time-series data (*reference 3.2*), it's important to keep that data in order (by date, from older to most recent).

We then proceed with K-Fold Cross Validation. For example, in an iteration with 4 folds out of 10. We use the first 3 folds for training and the fourth to test (see illustration below), we then record the accuracy from that iteration and keep moving forward until we run out of folds. The final accuracy is mean of the accuracies of all folds.

The illustration below demonstrates how K-Fold cross validation is used in the current implementation with 5 folds. In the implementation of our model, we've chosen to use 10 folds.



The regression metrics considered for this problem are:

- R^2 **score**
- **Mean Squared Error**

$$1/n \sum_{i=1}^{n} (prediction_i - actual_i)^2$$

- **Mean Absolute Percentage Error**

$$1/n \sum_{i=1}^{n} (|prediction_i - actual_i| \ / \ actual_i) * 100$$

As we have chosen a benchmark (5% error on average) which requires us to calculate a specific accuracy for each model's prediction.

This means using the R^2 score would not help us in meaningful ways unless normalized (i.e. we need to know what's the worse and what's the best score - reference 3.3). Hence we are left with the **mean_squared_error**. This metric is great if we want to find out the dollar value for how far are our predictions from the actual prices.

A slight modification to the mean squared error formula allows us to evaluate our predictions with a percentage (*see **Mean Absolute Percentage Error** above*).

We proceed by using these metrics on the predictions made by each algorithm and then pick the best two algorithms. Results will be discussed in the next chapter.

Refinements

A few refinements have been made to the initial preprocessing steps in order to ensure better outcomes. These improvements include:

1. Adding the horizon (or lag) columns ONLY to the tickers we are trying to predict rather than all the tickers that we are training our algorithm on.
    i. Doing this reduces our feature space (i.e. size of dataset) tremendously, thus improving the speed and accuracy of our predictive models.
2. Transforming the values of each volume feature into a percentage change or scale it logarithmically because the values are too large and could effect the some algorithms. This will not effect ensemble tree methods since they are invariant to the scaling of inputs.
3. Dropping the **open**, **high**, **low** and (unadjusted) **close** columns of all tickers as these values do not effect the accuracy of our predictions at a significant level. We

therefore default to reducing the amount of features used. Another reason is that the **adj_close** column should help us predict all those other columns which we are dropping. We can also verify this by checking

Future improvements could include adding features of industry/sector aggregated stock indices. For example, if most of the stocks we are trying to predict are in the Information Technology sector, then we can add a new column containing an aggregation of all other companies in the sector.

A different approach to the problem may include trying to predict an ordinal output, this means we can predict the direction of the price (up or down) as well as how strongly we are confident about this change (low, medium, high).

Results

4

In this chapter, we will go over the results of our final model in detail. We will also discuss how our model performs with different sets of inputs to validate that we have arrived to a well generalized model.

Model Evaluation and Validation

The final algorithm selected is **Gradient Boosted Trees**. The general idea is to compute a sequence of (very) simple trees, where each successive tree is built for the prediction residuals of the preceding tree.

Evaluating Different Stocks As Inputs

We can then use the selected model to evaluate different sets of input data and check if our model is indeed arriving to an acceptable result (i.e. realistic model):

Train Tickers	Predict Tickers	Algorithm	Mean % Err
MSFT, AMZN, GOOG, AAPL	MSFT	GradientBoosted	2.514%
PYPL, ADBE, QCOM	QCOM, PYPL	GradientBoosted	1.233%
RHT, ADSK, EBAY	EBAY	GradientBoosted	3.863%

Evaluating Different Training Data Sizes

Using the results table above, we can observe that our model generalizes well for different stock tickers. Another evaluation to make is the size of the training data.

Train Tickers	Predict Tickers	Training Data Size (in months)	Mean % Err
MSFT, AMZN, GOOG, AAPL	MSFT, AMZN	18	2.661%
MSFT, AMZN, GOOG, AAPL	MSFT, AMZN	30	2.351%
MSFT, AMZN, GOOG, AAPL	MSFT, AMZN	42	3.218%

The table above shows that model is not significantly impacted when change the range of the size data available for training. In fact, the model generally performs better when there's a relatively smaller amount of data. A hypothesis for why our model is behaving this way is that historic or past trends are effecting future predictions.

We also use 10 folds when applying cross validation to with our dataset to ensure that our model generalizes well.

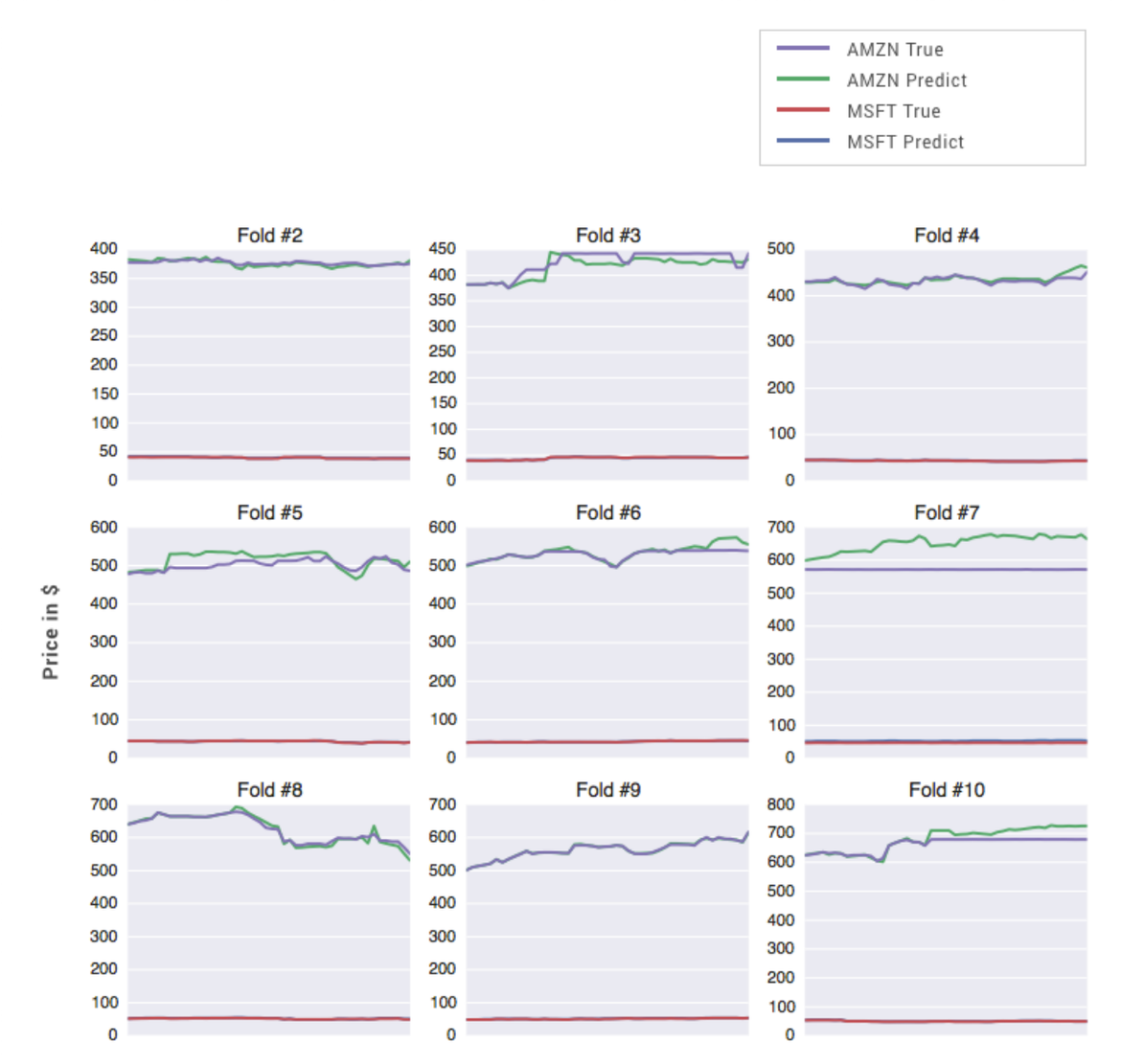
Evaluating With Lags (vs. horizons)

Since we are currently using horizon features (prices from future days) during preprocessing, we can validate that our model will perform well by using lags (prices from past days) to ensure that we have reached an acceptable model. The table below shows the result:

Train Tickers	Predict Tickers	Training Data Size (in months)	Mean % Err
MSFT, AMZN, GOOG, AAPL	MSFT, AMZN	18	2.861%
MSFT, AMZN, GOOG, AAPL	MSFT, AMZN	30	2.455%
MSFT, AMZN, GOOG, AAPL	MSFT, AMZN	42	3.732%

K-Fold Cross Validation Results

We an also plot the predictions against the actual values to visually evaluate how closely our model performs well with data it has not seen before. The graph below outlines the predictions of Fold #6 in our Cross Validation process.



The plots above compare the actual values of the stock prices for Microsoft (MSFT) and Amazon (AMZN) to the values predicted by our model for each fold during cross validation.

Justification

Using the results report in this chapter, we notice that our model has outperformed the benchmark set at the beginning of the project (5% mean error) in the case where most of the needed data is available.

Since we have tested our model with different types of inputs and it maintained its performance, we deduce that the model is generalized well and accept the results.

It is important however to note that these results are not sufficient to solve the problem of predicting whether or not to purchase a certain stock. In order to achieve that, we would need to add buy/sell indicators which could be achieved by adding other models (perhaps classifiers) to support our final algorithmic trading model.

Conclusion

5

In this chapter, we will discuss final thoughts about the project from start to beginning.

Reflection

To conclude this project, we can summarize the steps taken as follows:

1. Read about stocks and understand how they work
2. Research data vendors and gather the required training data for the problem
3. Fill the gaps in the available dataset
4. Transform, remove and add features to the dataset as needed. This step is based primarily on the work done in the Analysis section.
5. Research algorithms that can be used to solve the problem
6. Run tests with different algorithms and pick the one that generalizes best
7. Refine data further and rerun tests until satisfied with results
8. Stress test your model by running test with differing inputs
9. Use cross validation to ensure consistent results

Each of the steps above required its own mentality and approach and can give you insight into the problem from a different angle.

For example, ensuring that your data is processed correctly gives you insight into mechanics of the problem. In our project, having time-series data is one of those milestones you must pay attention to.

Another interesting aspect of this project is the number of ways you can improve your results. Since stocks are traded for a variety of industries, there is a lot of opportunity for building models that take into account the sectors of the stocks we want to predict and decided automatically which other stocks it needs to train with.

As a final thought, I believe building this project can give insight into how a lot of the financial firms work as well as the behaviour of people when it comes to making investments in stocks.

Improvement

Many improvements could be made to the model that we have created in this project. Some of these improvements include:

- Adding a buy/sell indicator
- Taking into account similar stocks to the ones we want to predict
- Taking into account fundamental data for long term predictions
- Adding a portfolio management feature where you specify the stocks in your portfolio and the trading capital available, then the algorithm manages the ratios invested in each of companies