# Equity Price Prediction with LSTM

Anthony Chi, Data Scientist

**September 6th, 2018**

### Project Overview

Equity price prediction has always been a difficult task. That’s why long-short or pair strategies are prevalent in the industry. Using traditional machine learning, ie. classification or regression, the results are quite mediocre. Reasons are often:

1. Users unable to engineer meaningful features that correlate with desired targets;
2. Traditional machine learning algorithms do not incorporate all available information represented in the data and its structure.
3. No one model can generalize for individual stocks, or even at the industry level.

The goal for this project is to explore the predictive power with traditional and non-traditional means. I want to know how the latest and greatest machine learning algorithms and feature engineering procedures can potentially achieve superior results.

### Project Goals and Some Directions

The goal of the exercise was to develop a RNN construct, specifically LSTM, to beat benchmark model performances. I also wanted to explore the potential to optimize LSTM with hyperparameters. Here were the steps that I took:

* Explore data – there were a few data sources available for use. I needed to select a source.
* Clean data –
  + Outliers: It was important to investigate the outliers. If outliers were not due to mistakes, it was important to include outliers as the model needed to be able to fit and predict extreme behaviors. This may be more relevant in this domain than other domains.
  + Missing data: Explored the reasons for missing data. Database was likely discarded if no reasonable explanation was uncovered for the missing data.
* Feature treatments – by looking at the features’ distribution as well as their correlation with the targets, did we need to implement special treatments? Here were the questions we needed to answer:
  + Was PCA necessary, for dimensionality reduction and improved model performance?
  + Did we need transformations to make the distribution more normal, and thereby improving model performance?
  + Were there correlation between features and was it difficult to select features to use in modeling algorithms?
* Prepare data – how should the data be structured, as they were fed into modeling algorithms? LSTM needs special attention.
* Model selection – there were two benchmark models: Lasso Regression, and XGBoost Classification, as they were historically higher performance modeling algorithms.
* My goal was to develop a LSTM construct that would beat the benchmarks, in both classification and regression. I would also try to optimize the LSTM with hyperparameters.

### Measure of Success – Metrics

For the classification exercise, I used accuracy as my metric. For the regression exercise, mean square error was used. MSE was compared to the target mean (as a percentage), to get an idea of the magnitude of the error.

### Data and Some Difficulties

The goal here was to examine the available data, and achieve the following objectives:

Compare Data Sources: many data sources are becoming unavailable, turned into paid services, or only available on proprietary platforms. Many APIs were also inactivated, and webpages were inactivated or unable to be scrapped. I was only able to access three data sources: Quandl API, NASDAQ, and finance.yahoo.com.

Some concerns after comparing three data sources:

* There were apparent differences in Adjusted Closing Price and Adjusted Volume between data sources. I wasn’t sure which one to trust.
* There were also occasional missing data from Quandl API.
* The NASDAQ maximum historical data only spanned 10 years, making the max length of dataset ~2500 rows.

With these difficulties in mind, I went with data from finance.yahoo.com. Main reasons are no missing data and large datasets.

Using the available features, including open, close, high, low, volume and adj. close, I generated additional features:

* High-Low Range, as a percentage of previous day’s close: this described how volatile the day’s movements were
* Open-Close Range, as percentage of same day’s open: this described the general movement of the day’s activities
* Moving average (adjusted close & volume) for a specific time period.

The new and old features needed to answer the following questions or explore ideas:

* What am I trying to predict: specific price, change in price, simple direction, or general trend/direction? This will determine if this is a classification (bi or multi) or a regression exercise.
* Were there any relationship between the independent and dependent features?
* What did the feature distributions look like? Did I need to make any transformation?
* Were there any missing or outliers that I needed to look into?
* Did I needed to use scalers, or other feature engineering techniques such as PCA?
* Was there a model that can be applied to all equities? Or did I need to train for each equity?

### Data Exploration

### **Feature Engineering**

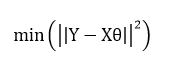
### **PCA**

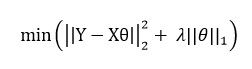
### Model Optimizations

**Benchmarks**

**Lasso**

Lasso is short for least absolute shrinkage and selector operator. Lasso regression was similar to an ordinary least square regression, where the algorithm was trying to minimize the squared error term. Where the Lasso was different from OLS was the λ term. When λ increased, many of the correlated coefficients would be reduced to zeros. This property was known as feature selection, where Lasso regression only keep features that are uncorrelated and relevant. However, due to feature selection, we may loose information contained in the dataset. Due to Lasso’s simplicity, efficiency, and ability to perform feature selection, Lasso could also be a good benchmark for future regression exercises.

 OLS error term

 Lasso error term

**Random Forest**

RandomForestRegressor: Random Forest was an ensemble method, largely based on the decision tree algorithm. Ensemble methods are highly regarded as models with higher accuracy. I expected Random Forest to perform better than Decision Tree.

**XGBoost**

XGBoost stood for extreme gradient boosting. It was an algorithm based on Gradient Boosting, which combines weak learners into a single strong learner in an iterative fashion. In Gradient Boosting, Gradient descent was use in each iteration to minimize the error function (an additive process). In the case for the Random Forest algorithm, it would generate independent trees with different results, which were aggregated to form the recommendation for the ensemble algorithm. Contrasting Random Forest to Gradient Boosting, instead of having independent trees, trees were added to the previous tree to compliment what the previous tree failed to realize. XGBoost was an algorithm developed to include regularization, which placed emphasis to control for over-fitting. In addition to building a more generalized model, XGBoost also had system optimization and algorithm improvement, which could run more efficiently. As an algorithm that won many kaggle competitions, I expected XGBoost algorithm to perform well.

**GridSearchCV Optimization**

In addition to the default hyper parameters, I also tried to optimize each of the benchmark algorithms to see if I can improve their performance.

**LSTM**

it is extremely difficult to optimize LSTM as hyperparameters to optimize seems endless, including number of neurons in each layer, number of layers, activation function, dropout, epochs, learning rate, batch size, optimizers, etc. Therefore, I was only determined to select the best performing model out of a fixed set of hyper parameters.

**Missing Data**

<insert chart for missing feature>

**New Features created**

**Target**

<insert target distribution and transformation chart>

**Features Transformed**

**PCA**

### Benchmark

1. I used two benchmark models, namely XGBoost Classifier and Lasso Regression. The ultimate goal was to use LSTM to beat these benchmark models. Supposedly, LSTM should be able to incorporate long and short term memories to generate additional insights, and further improve the predictive power of resulting models.
2. During the LSTM optimization exercise, I attempted to explore how the hyperparameters such as window, epoch, batch size, and LSTM constructs affected the model performances. This surely helped me tune the model for better performances.
3. Also note this was a regression, as well as, a classification exercise.
   * 1. As the exercise became computationally expensive, I moved computational and storage needs onto AWS.

**III. Methodology**

**Data Preprocessing**

scalers

**Target**

**Outliers**

**Missing Data**

**Delete Feature**

**Feature Types Transformed**

**One-hot Features**

**New Features created**

**Features Transformed**

**PCA**

### Implementation

### Refinement

### **Lasso Regression** **(benchmark)**

### **XGBoost (benchmark)**

### **LSTM**

## IV. Results

### Model Evaluation and Validation

### Justification

## V. Conclusion

### Free-Form Visualization

### Reflection

# Findings and questions

# Questions:

# Predicting next day price may not be the best thing to do. We may want to predict price in 5 days, or even much more ahead? What’s a reasonable window to predict price before randomness takes over?

# Should I predict volume?

# Should I include other features to predict price?

# Does dropout improve performance?

# Does multiple layers improve performance?

# Should we actually try to predict price? Or should we predict a range? For example, -1~1% is neutral, 1~5% is good, 5%+ is excellent, etc.

# Findings:

# Best results is with window 15 (about 3 weeks), epoch 2000, and batch-size 50-250.

# There may be upward bias, meaning all that we have seen is upward trend (most of the time).

# When selecting hyperparameters, training predict time, testing predict time, training error eval time, and testing error eval time are negligible.

# The larger the window, the longer the training time. The higher the epoch, and the smaller the batch size, the longer the training time.

# Best training error achieved is with window 15, batch size 10, epoch 500: 2.5616e-06

# Best testing error achieved is with window 10, batch size 500, epoch 2000: 7.58083e-05

### Citation and Sources

**Relevant Files and Folders**