Bitcoin Trading Simulation

Nicholas Loprinzo
Georgia Institute of Technology
North Ave NW
Atlanta, GA 30332
nloprinzo3@gatech.edu

Jordan Shartar
Georgia Institute of Technology
North Ave NW
Atlanta, GA 30332
jshartar6@gatech.edu

Aishwarya Chitnis
Georgia Institute of Technology
North Ave NW
Atlanta, GA 30332
aishwaryachitnis@gmail.com

Graeme Sharpe
Georgia Institute of Technology
North Ave NW
Atlanta, GA 30332
graeme.sharpe001@gmail.com

1 Problem Description

Cryptocurrency is the new popular investment trend that people want to make money off of today. Think of it as a new stock market. Our project is to create a cryptocurrency forecaster using different regression models of machine learning. Taking the forecasted prediction of our trained models, we will choose to either "Buy", "Sell", or "Hold" on our Bitcoin Investment. The result will then be compared to the real data, giving the accuracy of the bot.

1.1 Project Plan

1. Collect and clean historical bitcoin price data.
2. Split data into various training and testing groups using cross validation to reduce our chances of overfitting. Specifically use cross validation for time series.
3. Create a backtesting program that we can use to evaluate our models over historical data.
4. Identify, create, and test various algorithms to identify which performs best.
5. Create an action function which decides to either "buy", "sell", or "hold" bitcoin. Use a
6. Create an assemble learner that combines algorithms to make appropriate decisions to buy or sell.

1.2 Research Goals

1. To set up our forecasting models and "buy"/"sell" rules such that we finish the simulation with more money than would have been earned with no trading.
2. For project members to learn about time series regression, and to create process for analyzing time series/tuning hyperparameters in the future.
3. For project members to know about the different types of Neural Networks and apply them to trading cryptocurrencies.
4. For project members to become familiar with sklearn, keras, tensorflow, statsmodels and other packages in python which are useful for machine learning/ hyperparameter tuning.

1.3 Data

We scrapped mean daily bitcoin value from the internet and placed it into a normal CSV file to aid in loading it into our environment with pandas. The training set is defined from 1/1/2016 to 8/20/2017. The testing set is defined from 8/21/2017 to 10/20/2017.

2 Related Works

- What Hedge Funds Really Do by Romero and Balch [6] was used to learn different approaches to building an investment strategy and hedge funds. This was mostly used as a background of the topic to be applied to cryptocurrency.
- The trade bot for apple [2] is an example of a bot that gives recommended actions based on changes in the stock for Apple. It gives a good idea of what the bot for the project

should be achieving and what our data should look like graphically.

- "Bitcoin price forecasting with deep learning algorithms" [3] was used as the base model for the project. This work was used to get a initial result then was improved upon.
- Advanced time series analysis [7] was used as a resource for teaching the group how to do time series manipulation, how to build SARIMA, and how to make predictions using xgboost.
- Hyperparameter optimization with Keras article[4] gave a machine assisted hyperparameter configuration using a Keras tutorial and explanation. This article helped us set a grid search to find more optimal hyperparameter settings in keras models.
- The automated hyperparameter tuning article [5] was used in unison with the Keras optimization[4]. This article gave the group valuable information on how to change hyperparameters to be more useful without having to manually change them based on educated guesses.

The baseline model we wish to are using comes from resource [3]. This articles uses GRU and LSTM Neural networks to predict the nexts days bitcoin price values. We will attempt to use hyperparameter optimization and cross validation to select a better LSTM model and GRU model, and then use the best versions of these models to create the predictions used in our optimized action function. The RMSE of the baseline LSTM Neural network was 18.693 and the RMSE of the baseline GRU model was 27.386. Both of these baselines are very accurate, so the window for improvement is quite small.

3 Methodology

We used python 3.6 while taking advantage of pandas, numpy, and matplotlib for data storage and graphing. We used keras, tensorflow, and statsmodels to build regression models, we used many available optimized packages such as hyperopt, pyramid-arima, and sklearn's grid search to help optimize the hyperparameters of each of our functions. A full list of packages used includes: pandas, numpy, matplotlib, seaborn, statsmodels, pyramid-arima, keras, tensorflow, sklearn, and hyperopt.

3.1 First Approach

Our first approach was based on generating different models to forecast the next days mean bitcoin price, and then creating a separate action function to trade based on expected one day percent increase in bitcoin value.

3.1.1 Model 1: HoltWinters

This is a time series forecasting technique which adjusts its predictions based on each daily update in price. We trained the hyperparameters of the HoltWinters model using a grid search over the entire parameter space, and we trained the regular parameters of the model using statsmodels optimizer within our function call to create the model. The RMSE of this best model was 49.5. We performed 10 fold Cross validation on our model and found that RMSE ranged from approximately 17.3 to 308.1. This indicates that our model may have been overfitting on the price data towards the beginning of our time series, and the parameters chosen for that portion of the series were do not apply well to the entire dataset. After looking at the Bitcoin price graph, it became fairly obvious that the graph was fluctuating differently across time. Investigating more ways to make a time series stationary may be useful in this situation

3.1.2 Model 2: ARIMA

This is a another time series specific technique which creates variables to eliminate trend in residual values. We trained the ARIMA model using a combination of Cross Validation and an auto_arima package with an optimized search for each of the parameters on the log transform of the bitcoin price data. The RMSE of our ARIMA model was 101.6 and the range of RMSEs observed during cross validation was 45.4 to 203.6. The RMSe of the base model is not very good, and that in combination with the large variance in RMSEs during cross validation indicated that this model will not generalize well to test data.

3.1.3 Model 3: GRU Neural Network

Similarly, we acquired the baseline model for a GRU Neural Net from our Medium article. We tried to optimize the hyperparameters using a grid search over a small set of variables to save run time. We weren't able to identify any better hyperparameters than those set in our baseline. We were able to improve upon the baseline by running 10 fold cross validation on the data and then subtracting the mean RMSE of the cross validated models from the predictions of the original model. Baseline RMSE was 27.386, but we were able to reduce it to 22.402 after subtracting the mean RMSE. The combination of using cross validated models and their average RMSE with the original predictions performed fairly well on both training and test data and actually improved the model.

3.1.4 Model 4: LSTM Neural Network

We acquired the layout and code to run the LSTM as our baseline from a Medium article. We used a grid search to attempt optimize a small set of parameters for the Neural Network. However, we were not able to identify better hyperparameters than those used in our baseline forecasting model. We wanted to improve the RMSE by subtracting the mean RMSE of the Cross Validated models from the predictions of our original model as we had done with the GRU Neural Network. However, our RMSE increased from 18.693 to an average of 27.255performing this step, so we decided to not use it in our final predicted values. The range of the 10 RMSEs was from 3.319 to 125.773 with a mean of 29.810. The large variance in RMSEs could indicate that the original model may be overfitting on the training set. This could be investigated in future works. The symmetric mean absolute percentage error using the original predictions was 0.384.

3.1.5 Action Function: “Buy”, “Sell”, “Hold”

We created an action function which would decide whether to “buy”, “sell”, or “hold” on our current bitcoin investment based on the predicted daily growth from each individual model. We set the function to invest in bitcoin when the percent growth was greater than some alpha, to sell when percent growth was less than some gamma, and to hold when percent growth was in between these values. In this current setup there are two parameters: alpha and gamma which we solved for using a Tree-of-Parzen-Estimators (TPE) algorithm, a type of sequential model based optimization (Bayesian learning). We chose to use alpha gamma, the buy amount, and the sell amount as parameters to maximize profit

3.2 Second Approach

The goal of our second approach was to create an ensemble learner that combined both reinforcement and supervised learning. We used two separate learners, one utilizing Q-learning while the other uses bootstrap aggregating to create a random forest learner made of many individual trees. We will compare the results of the individual models versus the combine performance of the ensemble.

3.2.1 Model 1: Q-learner

We used a Dyna-Q learning algorithm for training, and its implements a greedy reward function which prioritizes high profit trades. Fundamentally, this is equivalent to accepting greater risk. The reward function is strongly related to change in profit from actions. The total state space was defined by integers ranged from 1 to 5000. They were hashed representations of the current price and market indicators.

3.2.2 Model 2: RF Bag Learner

To create the Random Forest learner we first implemented a simple random tree classifier. Next we created a wrapper class that utilized bootstrap aggregating to train 10 separate trees from different sets of data. Together this wrapper return an average of all trees’ results to give a final classification.

3.2.3 Model 3: Ensemble Learner

The final ensemble learner trained both the Q-learner and the Random Forest on the same data. Next it queried each learner and combined the weighted votes to determine what action to make each day. If either learner’s action lost profit their voting weight would be updated.

4 Results

We shall present the results of our bitcoin simulation in the same manner in which they were introduced in the methodology. The first set of results is based on passing the fitted values from the each individual model and into the action function and letting Bayesian optimizer select values for each of the parameters. A graph of parameters tested shows how the bayesian optimizer balances exploration and exploitation to find local minimums in the objective function.

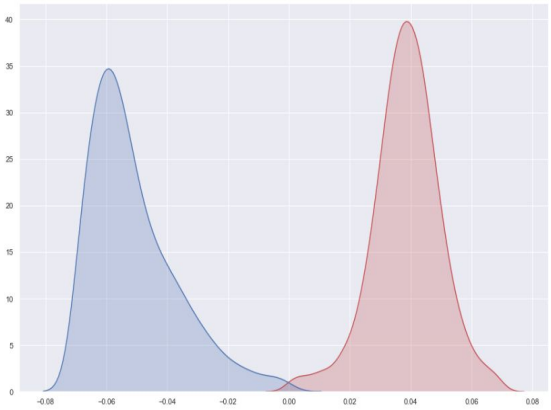

Figure 1: Distribution of Alpha (red) and Gamma (blue) parameters tested by our Action Function

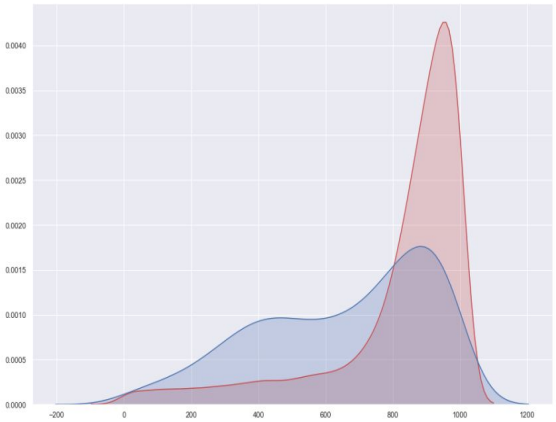

Figure 2: Distribution of Buy (Red) and Sell (Blue) parameters tested by our Action Function

4.1 Model + Action Function Results

4.1.1 Model 1: HoltWinters

This model finished with $27,709 USD Profit after 21 Buy/Sell trades running on test data from August 21st 2017 to October 20th 2017. The model made no buy trades in the last 10 days because it had run out of money in its reserve to invest.

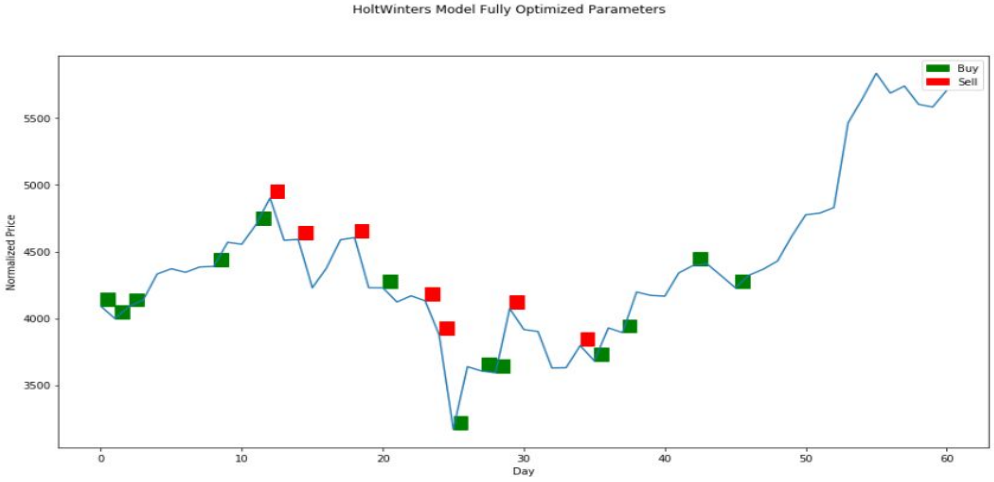

Figure 3: HoltWinters Results

4.1.2 Model 2: ARIMA

The model finished with $31,693 USD Profit after 31 Buy/Sell trades on test data from August 21st 2017 to October 20th 2017. The ARIMA model itself fits very poorly to the time series.

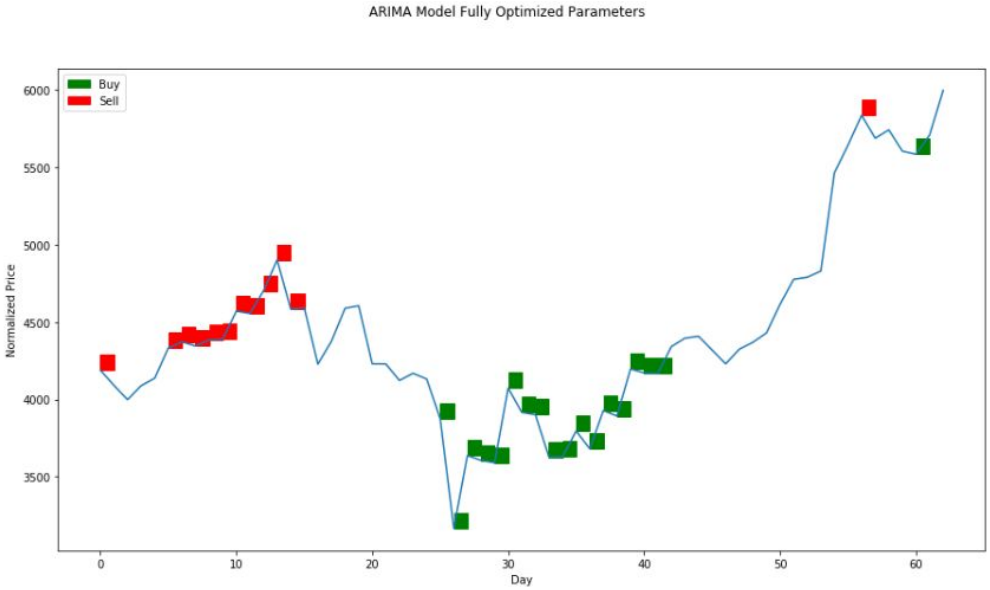

Figure 4: ARIMA results

4.1.2 Model 2: GRU Neural Network

This model finished with $28,112 USD Profit after 15 Buy/Sell trades running on test data from August 21st 2017 to October 20th 2017. Again, the model made no trades in the last 20 days because it had run out of money it could invest.

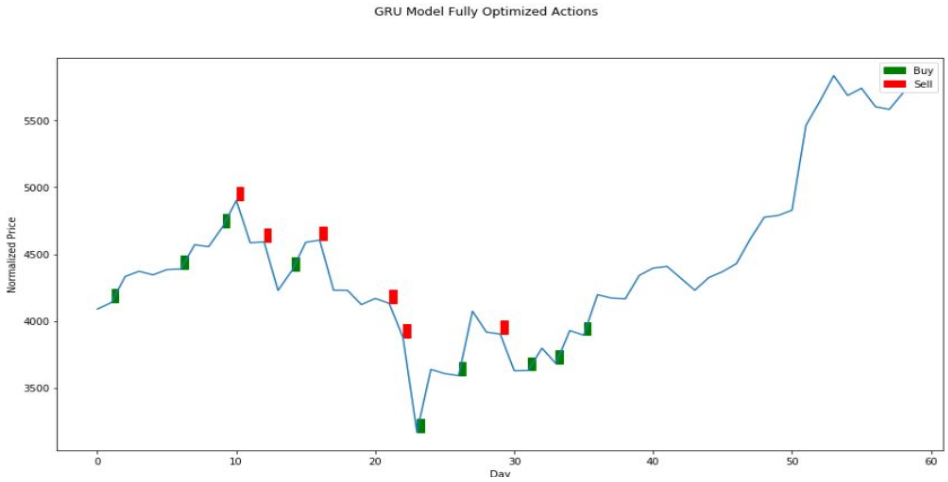

Figure 5: GRU Neural Network Results

4.1.3 Model 3: LSTM Neural Network

This model finished with $28,215 USD Profit after 13 Buy/Sell trades running on test data from August 21st 2017 to October 20th 2017.

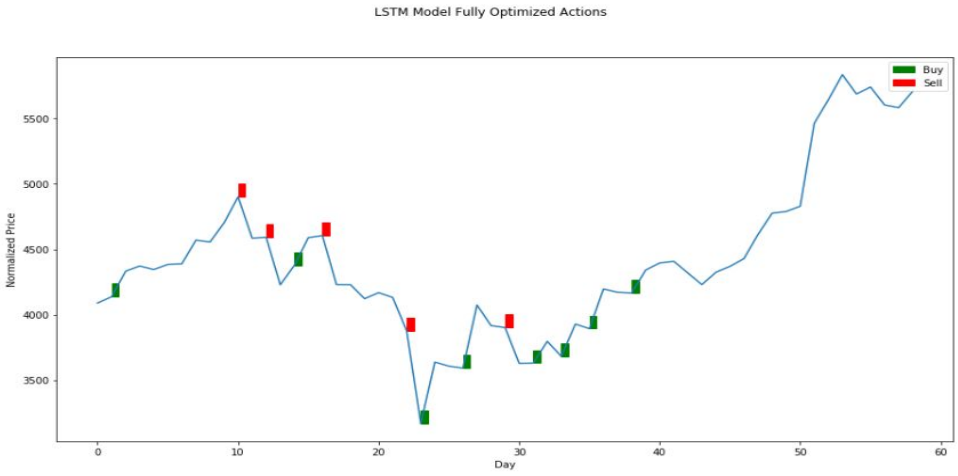

Figure 6: LSTM Neural Network

4.2 Q-learner Results

RF-learner ended with $0USD profit after 0 Buy/Sell trades when tested from 8/21/2017 to 10/20/2017.

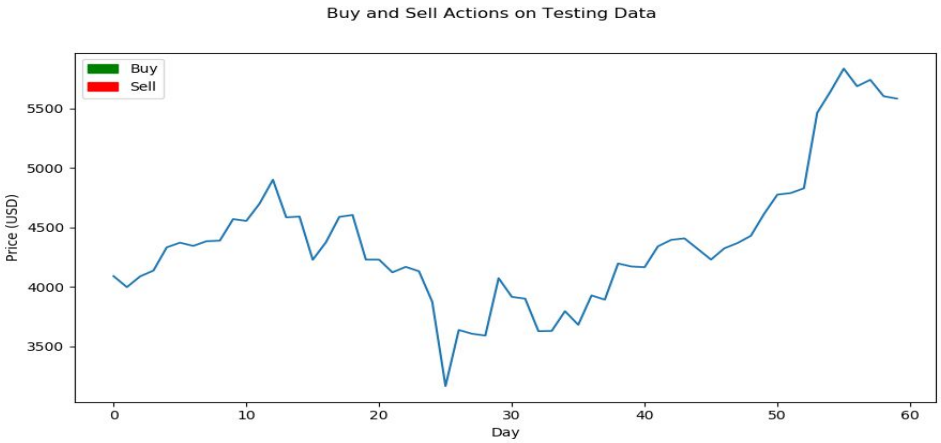

Figure 7: Q-learner Results $10,000 Initial Value

Q-learner ended with $7000 USD profit after 39 Buy/Sell trades when tested 01/01/2017 to 12/01/2017.

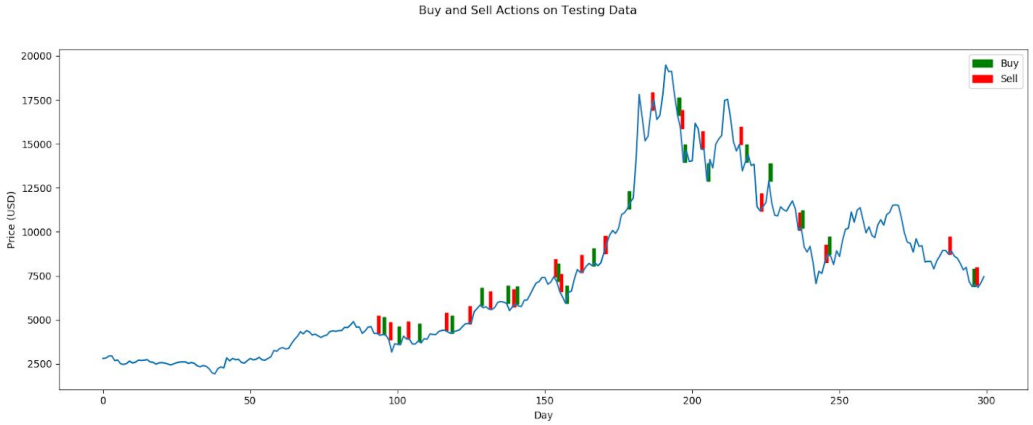

Figure 8: Q-learner Results $10,000 Initial Value

4.3 RF Learner Results

RF-learner ended with $3,128 USD profit after 27 Buy/Sell trades when tested from 8/21/2017 to 10/20/2017.

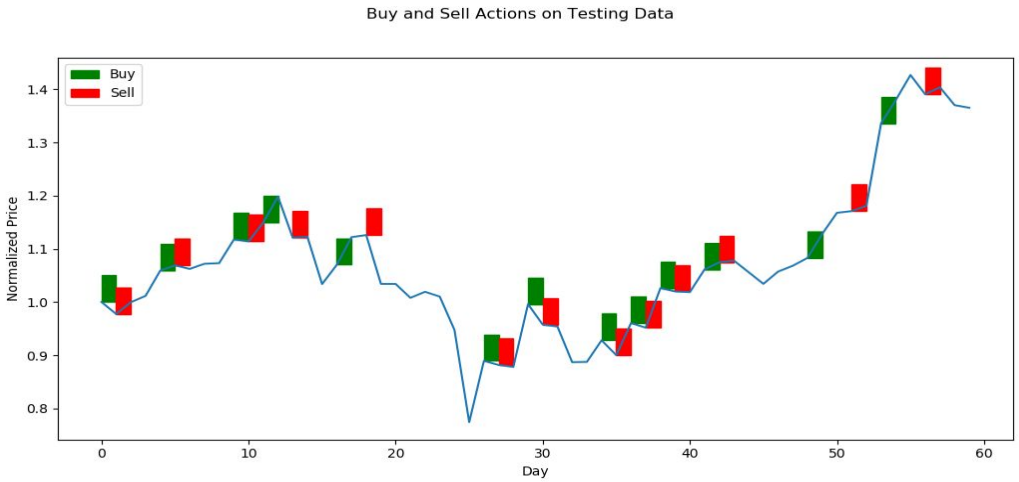

Figure 9: RF Learner Results

RF-learner ended with ~$14,000 USD profit after 55 Buy/Sell trades when tested from 01/01/2017 to 12/01/2017.

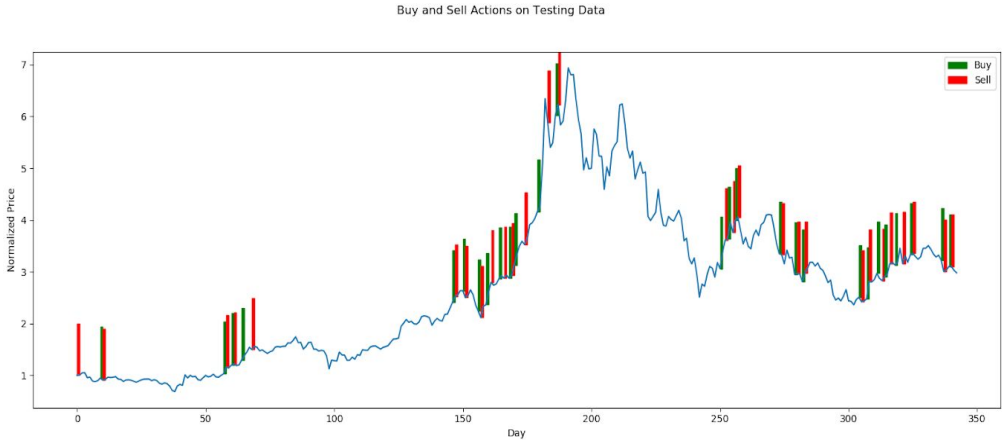

Figure 10: RF Learner Results $10,000 Initial Value

4.4 Ensemble Learner Results

Ensemble learner ended with $843 USD profit after 17 Buy/Sell trades when tested from 01/01/2017 to 12/01/2017.

Figure 10: Ensemble Learner Results

Ensemble learner ended with $10,281 USD profit after 76 Buy/Sell trades when tested from 01/01/2017 to 12/01/2017.

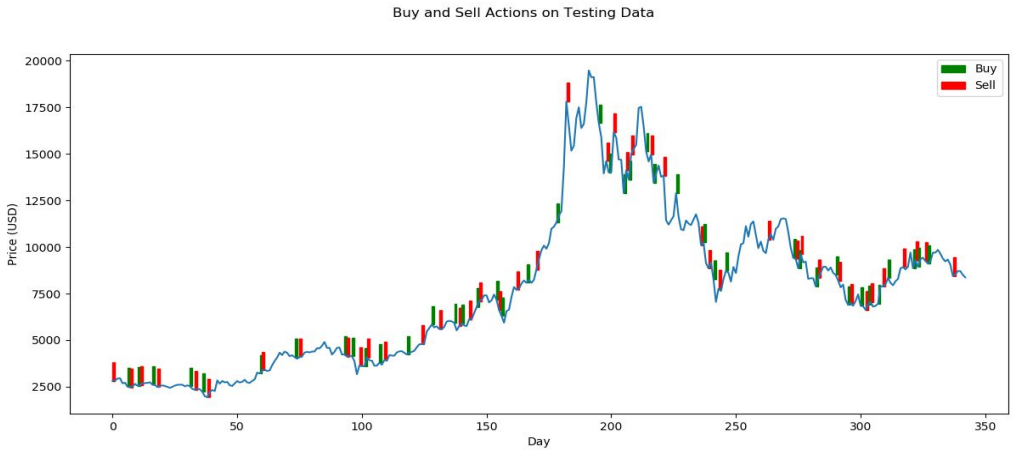

Figure 11: Ensemble Learner Results

4.5 Analysis

4.5.1 HoltWinters

The HoltWinters model performed well on the testing data and earns a profit comparable to that of the GRU and LSTM models. The HoltWinters model may indicate a case where it pays off more to have a simple model that earns almost as much as the complex models. Since it was so close to the GRU and LSTM models, we would recommend training using the HoltWinters model over the neural networks.

4.5.2 ARIMA

The ARIMA model made money because the action optimizer found a working solution, not because the model predicted well. We do not recommend using the ARIMA model as a backing for the action function because we suspect that once the pattern in this window ends, the model will lose a lot of money. The model works well in the long run because of a general upwards trend in bitcoin value, but it doesn't act on the next day's info well at all. This model is an example of a scenario where the model which makes the most money in testing isn't always the best. Understanding a model's error is important to
creating good solutions.

4.5.3 LSTM Neural Network

The LSTM model earns the most out of the models with reasonable errors. It predicts well, and from looking at the graph, you can tell that a lot of its decisions make sense. It sell right before a loss, and buys right before a gain most of the time.

4.5.4 GRU Neural Network

The GRU model also performs well on test data and from looking at the the graph of actions seems to show that logical predictions, given that we're only predicting 1 day into the future. It sells right before a loss and buys right before a gain most of the time as well.

4.5.5 Q-learner

The most difficult part of training the Q-learner involved tuning the reward function to get the proper balance between holding assets and making trades. Both have pros and cons and ultimately a more risky frequent trading strategy was favored while still holding when appropriate.

Unfortunately(Figure 7), our Q-learner did not learn the entire state space perfectly. Because it had not seen these states before it chose neither to buy nor sell but rather hold its assets. We ran the Q-learner again on a larger trading window to test performance.

4.5.2 RF Learner

The results of the RF Learner can vary widely due to the randomness included in both bagging and random decision trees. But, when trained well the trading strategy can outperform the Q-learner. We suspect this is strongly related to the the bootstrap aggregating used to train the multiple agents.

4.5.3 Ensemble Learner

Together the performance of the Q-learner and RF learner were combined to create a single trading strategy. Ultimately, this strategy was largely defined by the consistent action trading actions of the Q-learner that identified risky actions and the RF learner that prefered more smaller gain actions. However, based on performance it is possible that our ensemble's learners are not complementary. Our second approach proved less successful but in hindsight may be due to our failure to build a strong ensemble of complimentary learners.

4.6 Further Research

There are methods that need further research in order to complete and should be attempted to further increase the accuracy of the models. Feature engineering options could to be implemented, to create predictors from the price data given. This violates many assumptions made within linear regression, yet it's often easier to train than the other options we've explored, and it a reasonable model is created, it may be good enough. XGBoost could also be investigated. Within the Q-Learner, an autoencoder neural network should be implemented to replace the Q-table. It should simplify the state space and give better results for the Q-learner.

Another method that will increase accuracy of the models is to create an ensemble learner of all the models. It will combine the current models provided (in Methodology) to better predict future prices of cryptocurrencies (Bitcoin in this case). In addition, an unsupervised autoencoding network would have also been included in the ensemble. Then using the current method to decide trading, the ensemble learner should give "buy", "sell", or "hold" recommendations. This method is difficult to implement and time did not permit its creation.

5 Conclusion

Overall, each model performed well on the testing data. However, regression based models from our first approach that traded based on future price predict seem to have perform better. We measured performanced based on profit gained with the LSTM Neural Network earning the most. The ARIMA model is a special case which happened to work in this environment, but high cross validation error, high error in general indicate that this model is not a good one to use as the backbone to our action function.

It is interesting to compare our two separate approaches and the results. Our second approach attempted to create a trading agent that could perform well in any market using an ensemble of learners. However, it may be noted that we may have failed to build the ensemble with complimentary learners. We were disappointed with the slim profit earned by the ensemble approach compared to the neural networks.

The neural networks created in our first approach were very successful. They managed to earn considerable profit while minimizing risk. Using powerful price predictors and regression they could identify the best buying and selling points. For being relatively simple, the HoltWinters model resulted in actions that lead to a similar profit as the neural networks. We recommend that those unfamiliar with tuning hyperparameters for Neural Networks use the simpler HoltWinters for this reason.

5.1 Future Work

Together, all of our models represent a well understanding of our problem and the many possible solutions. Individually, they can be used to make informed trading decisions but can be very risky. Ideally, all of our work with these various different models could be seamlessly integrated to create a highly informed trader that minimizes risk by utilizing various models. If we continued this project, this would be our next step.

References

[1](n.d.). Retrieved from https://www.binance.com/login.html?callback=/trade.html?symbol=BTC_USDT

[2]AAPL Stock Price and Chart. (n.d.). Retrieved from https://www.tradingview.com/symbols/NASDAQ-AAPL/

[3]Bobriakov, I. (2018, March 06). Bitcoin price forecasting with deep learning algorithms. Retrieved from https://medium.com/activewizards-machine-learning-company/bitcoin-price-forecasting-with-deep-learning-algorithms-eb578a2387a3

[4]M. (2018, May 15). Hyperparameter Optimization with Keras – Towards Data Science. Retrieved from https://towardsdatascience.com/hyperparameter-optimization-with-keras-b82e6364ca53

[5]Koehrsen, W. (2018, July 03). Automated Machine Learning Hyperparameter Tuning in Python. Retrieved from https://towardsdatascience.com/automated-machine-learning-hyperparameter-tuning-in-python-dfda59b72f8a

[6] Romero, P. J., & Balch, T. (2014). *What hedge funds really do an introduction to portfolio management*. New York, New York (222 East 46th Street, New York, NY 10017): Business Expert Press.

[7]Sergeev, D. (2018, April 10). Open Machine Learning Course. Topic 9. Part 1. Time series analysis in Python. Retrieved from https://medium.com/open-machine-learning-course/open-machine-learning-course-topic-9-time-series-analysis-in-python-a270cb05e0b3