Machine & Deep Learning in the Financial Markets

Udacity Machine Learning Engineer Nanodegree: Capstone Project

Peter Foy

10/11/2018

Table of Contents

I. Project Overview

The search for algorithms that can predict future price movements is the holy grail of finance, and has been coined *'the truth'* by quants. While algorithmic trading is nothing new and the computerization of order flow began in the 1970's, the last decade has seen a radical shift in three areas causing what has been termed *the fourth industrial revolution*. As this Machine Learning & Big Data paper by J.P. Morgan [1] puts it:

> *In fact, over the past year, the exponential increase of the amount and types of data available to investors prompted some to completely change their business strategy and adopt a 'Big Data' investment framework. Other investors may be unsure on how to assess the relevance of Big Data and Machine Learning, how much to invest in it, and many are still paralyzed in the face of what is also called the 'Fourth Industrial Revolution'. - J.P Morgan*

In particular, massive increases in data, access to low cost computing power, and advances in machine learning have significantly changed the financial industry. This paper will explore how several machine and deep learning algorithms can be applied in the cryptocurrency market.

Problem Statement

A common method for price prediction are regression-based strategies. Such strategies use regression analysis to extrapolate a trend to derive a financial instruments direction of future price movement. The problem to be solved, then, is understanding the relationship between historical data and future price prediction. In other words, we are looking for the relationship between the dependant and independent variable. Linear regression is appropriate for this problem as it analyzes two separate variables in order to find a single relationship. To solve this problem I will first use a linear ordinary least squares (OLS) model and then a neural network regression model using Tensorflow and Keras.

II. Analysis

In order to implement a data-driven investment approach we first need to acquire and understand our dataset, then we need to preprocess our data and implement the machine

learning models, and finally we need to analyze the results against a benchmark and suggest improvements for the models. The metric I will be using to analyze each model is the coefficient of determination, or R^2, which is “a statistical measure that represents the proportion of the variance for a dependent variable that's explained by an independent variable.” [4]. The equation for R^2 is as follows:

$$R^2 = 1 - (\text{Explained Variation} / \text{Total Variation})$$

I have chosen R^2 as it is an important metric in finance that can be used to measure the percentage of an assets performance as a result of a benchmark. It is important to note that the R^2 does have limitations as it simply provides an estimate of the relationship between price movements, but this does not necessarily indicate when the machine learning model is good or bad or if the predictions are biased.

Benchmark

The benchmark I will use is an R^2 score of 0. This number was chosen because financial asset (including cryptocurrencies) returns are often said to be unpredictable. An R^2 greater than 0 indicates that there is a relationship between the dependant and independent variable and an R^2 less than 0 suggests otherwise.

Datasets & Inputs

The dataset I have chosen to use is the largest cryptocurrency in terms of market capitalization - Bitcoin, relative to the US dollar (BTC-USD). The time frame for the datasets will be from 2016-01-01 to 2018-09-18, and the dataset have been downloaded from Quandl using this Bitfinex source [5].

	Date	High	Low	Mid	Last	Bid	Ask	Volume
0	2016-01-01	436.49	426.26	433.910	433.78	433.83	433.99	12922.080199
1	2016-01-02	435.80	430.00	433.330	433.20	433.20	433.46	6658.092194
2	2016-01-03	433.79	421.73	427.905	427.98	427.76	428.05	18644.760056
3	2016-01-04	435.67	426.20	434.475	434.49	434.47	434.48	13092.850666
4	2016-01-05	434.91	427.91	432.090	431.82	431.96	432.22	11532.984890

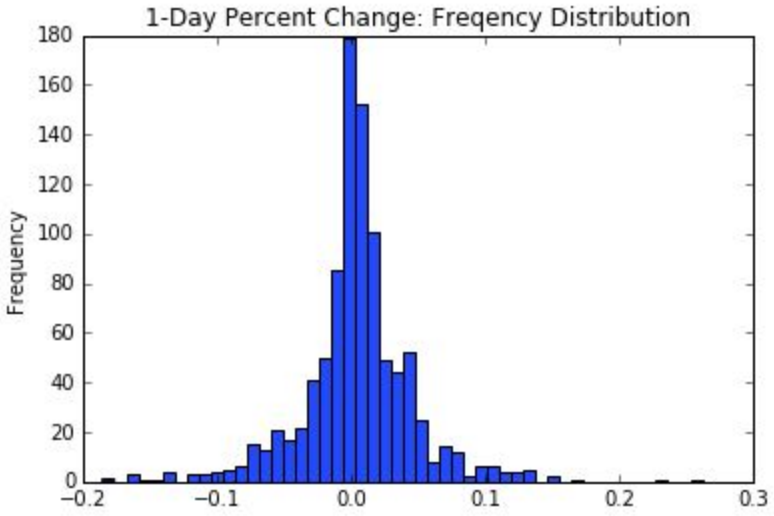
	Date	High	Low	Mid	Last	Bid	Ask	Volume
959	2018-09-04	7415.4	7246.2	7360.65	7360.700000	7360.6	7360.7	21047.188306
960	2018-09-05	7404.0	6900.0	6906.25	6906.000000	6906.2	6906.3	46598.570547
961	2018-09-06	6932.9	6302.2	6456.50	6454.200000	6456.4	6456.6	61607.045343
962	2018-09-07	6549.5	6322.8	6412.50	6412.594448	6412.4	6412.6	21379.071539
963	2018-09-08	6475.5	6119.5	6175.85	6175.900000	6175.8	6175.9	21694.860747

After importing the data we see this dataset has 8 columns and 964 entries. The dataset is daily time series data with the following features: Date, High, Low, Mid, Last, Bid, Ask, Volume. I will start by reviewing the daily close price, which is the 'Last' column in the dataset. This feature has a minimum value of $359.16, a maximum of $19,210, a mean of $3869.46, and a median of $1775. Also this dataset does have 3 NaN values, which will need to be dropped in the data preprocessing phase.

Exploratory Data Analysis

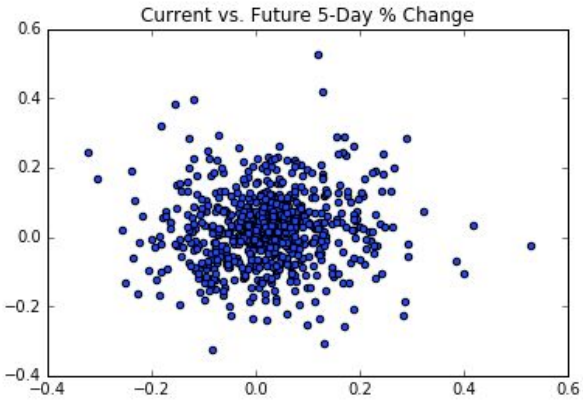
Using machine learning for finance can be accomplished in many ways such as predicting the raw prices of our stocks, but as described in this Machine Learning for Finance DataCamp course, typically we will predict percent changes [6]. This makes it easier to create a general-purpose model for stock price prediction.

To get the percent change over a particular time period I have used the *'pct_change'* function on a pandas dataframe. The following graph displays a histogram of daily percent changes over time, which we see is right-skewed and has a nearly normal distribution.



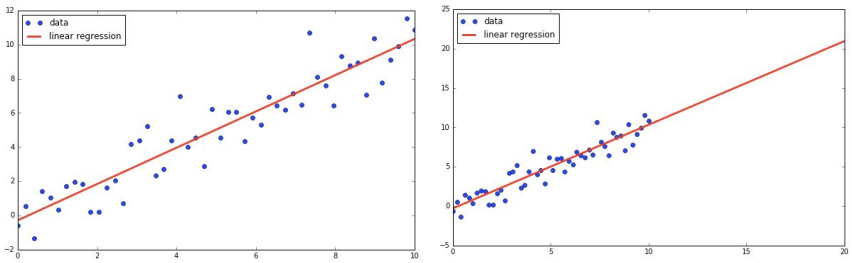
The next step is to shift our pandas dataframe by -5 in order to create a new column with the closing price 5 days in the future, this column is called *'5_day-future_close'.* The 5-day percent changes of the daily price for the current day, and 5 days in the future are then created. After creating these columns, the correlation between our percent price changes (present and future) is calculated to see if previous price changes can predict future price changes. We can see correlation is near 0 at 0.051, which indicates that the two variables are not linearly correlated.

	5d_close_pct	5d_close_future_pct
5d_close_pct	1.000000	0.051273
5d_close_future_pct	0.051273	1.000000



Algorithms/Techniques

In this section we will review the algorithms that will be applied to three types of trading strategies: in particular, linear regression, logistic regression, and deep learning based strategies. Linear regression is used to extrapolate a trend from the underlying asset. Linear regression and ordinary least squares (OLS) are decades-old statistical techniques that can be used to extrapolate a trend in the underlying asset and predict the direction of future price movement. A simple example of linear regression trend extrapolation can be seen below:



Finally, we will look a neural network regression algorithm to predict the future price of an asset. Loosely modeled after neurons in a biological brain, as the image below demonstrates, neural networks are connected by nodes that transmit a signal from one another through what are referred to as *hidden layers*. The final layer in the neural network is our target prediction variable, which in our case will be a future price prediction. To solve this, this section will make use of the deep learning libraries Tensorflow with Keras running on top of it.

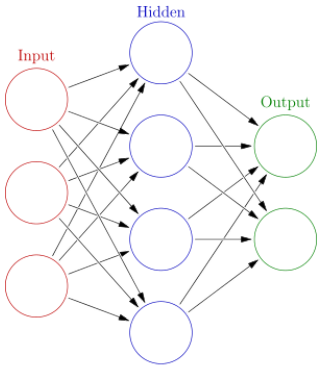


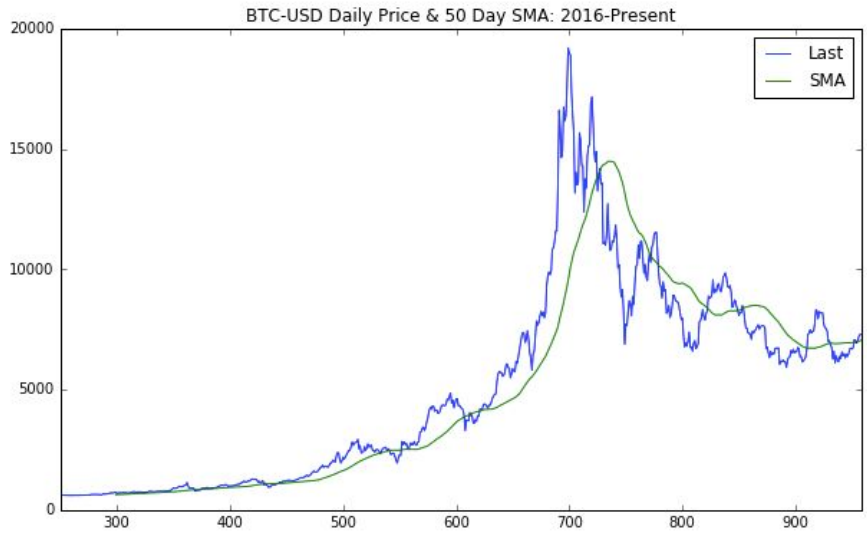
Image source [2]

III. Methodology

Data Preprocessing

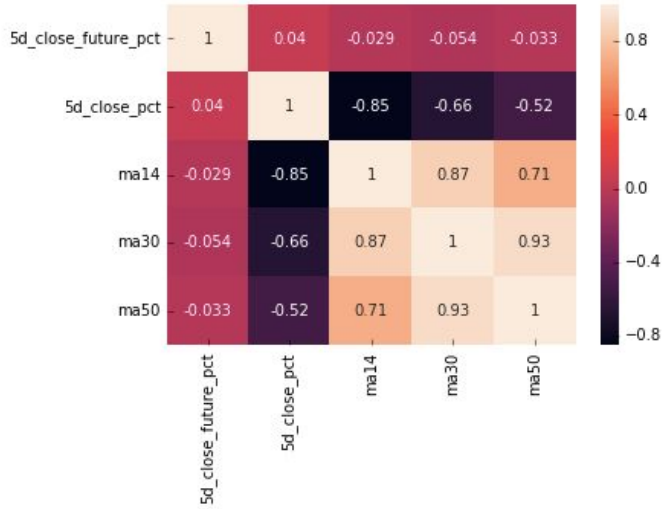
Now that we have explored and are familiar with our data, it is time to prepare it for machine learning. The first step is to create **features** and **targets**. Our features are the inputs we use to predict future price changes with, in this case we are using historical data points as features, and our target is the 5-day future price change.

We want to incorporate historical data as our features, for example the price changes in the last 50 days. Instead of including each daily price percent changes, we can concentrate historical data into a single point using simple moving averages. A simple moving average (SMA) is the average of a value in the last *n* days. The features I have created are SMA's for the time periods of 14, 30, and 50 days. The following graph displays the 50-day simple moving average with the daily closing price for BTC-USD from 2016-01-01:



The SMA indicator is calculated using the Ta-Lib python package. We can now use these indicators for prediction by creating a DataFrame that includes both a list of our features and the target 5-day future percent change. This is done so that we can analyze if there are

correlations between the features and targets before implementing our machine learning algorithms. The following plot uses the *seaborn* library to visualize a heatmap of the correlations.



Generally, we consider a correlation of 0.2 or greater to signify that there are linear correlations between the variables. From this heatmap, however, we see that none of the variables fit this criteria in relation to our 5-day future price change target.

Algorithms Implementation

Now that we have prepared our data for machine learning, we will start implementing a linear model with our data. To accomplish this we need to split our data in train and test sets - in this case 80% of the data is used for training and 20% for testing. The purpose of this is to fit our model to the training data and then test on the most recent data to understand how our algorithm will perform on unseen data. A key difference in modeling financial time-series data is that we can't use sklearn's *'train_test_split'* function because it randomly shuffles the data.

Linear Models

For linear models we need to add a constant to our features, which adds a column of 1's for a y intercept term. To do this we will use *statsmodels 'add_contstant'* function, and then we will the data into train and test subsets. Now that we have these two subsets we can fit a linear

model by using the Ordinary Least Squares (OLS) function from *statsmodels*. After fitting the data, the OLS Regression Results are printed and displayed below:

OLS Regression Results

Dep. Variable:	5d_close_future_pct	R-squared:	0.013
Model:	OLS	Adj. R-squared:	0.007
Method:	Least Squares	F-statistic:	2.200
Date:	Thu, 27 Sep 2018	Prob (F-statistic):	0.0674
Time:	12:03:31	Log-Likelihood:	594.95
No. Observations:	688	AIC:	-1180.
Df Residuals:	683	BIC:	-1157.
Df Model:	4		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-0.1074	0.092	-1.165	0.244	-0.288	0.074
5d_close_pct	0.0876	0.075	1.162	0.246	-0.060	0.235
ma14	0.3638	0.145	2.507	0.012	0.079	0.649
ma30	-0.3905	0.139	-2.811	0.005	-0.663	-0.118
ma50	0.1604	0.076	2.122	0.034	0.012	0.309

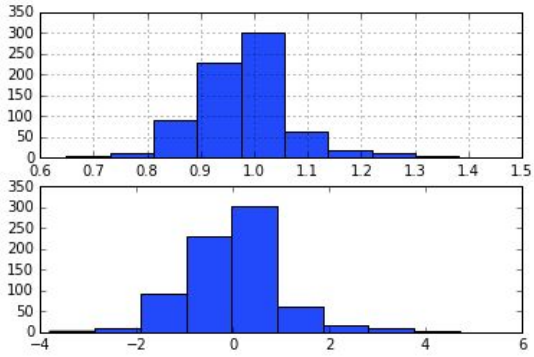
Omnibus:	41.007	Durbin-Watson:	0.402
Prob(Omnibus):	0.000	Jarque-Bera (JB):	100.163
Skew:	0.303	Prob(JB):	1.78e-22
Kurtosis:	4.769	Cond. No.	103.

As this displays many results, the one we will focus on for now is R^2. An R^2 of 1 indicates a perfect fit, and the lower the R-squared value the worse our fit. From the results we see an R-squared of 0.013, indicating the dependant variable does not explain the independent variable. This is to be expected however, since linear models are one of the simplest machine learning models available.

Neural Network

The use of neural networks has rapidly grown due to the exponential increase in GPU computational power over time, the size of data available, and improvements in software. Neural networks are similar to the previous models we have used in that they use features and targets to produce predictions, however neural networks have been shown to outperform other models because they capture variable interactions, have non-linearity, and are highly customizable.

Neural network models typically work better with standardized data, and one way to do this is by scaling our features. To do this we use sklearn's *'scale'* feature, which sets the mean to 0 and standard deviation to 1. A histogram of the scaled data are plotted below:



Each layer our our neural network uses input, a weight, a bias and an activation function to add non-linearity. We will use a common activation function called 'ReLu' or Rectified Linear Units, which is 0 for negative numbers and linear for positive numbers, to add non-linearity.

Predictions are made by passing data through our neural network in the forward direction, and the final node yields the prediction - this is know as forward propagation. Once we have our predictions we use the loss function to compare our predictions and targets. For regression, we often use the mean-squared-error loss function. We then use our error from the loss function and pass this backwards through the network, which updates weights and biases in order to improve our predictions - this is known as back propagation. Back propagation is how neural networks "learn", and is part of the reason why we needed to standardize our data earlier.

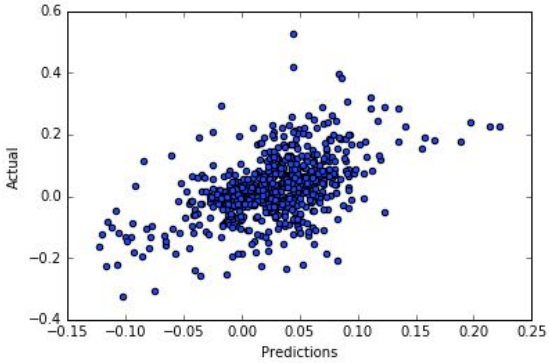
To implement our neural network we use the Keras library with TensorFlow backend. Keras is a high level API that allows us to build neural networks with minimal code, but still allows for customization. In order to test several different models, I created a *model_func* function to test different numbers of epochs and layers. I then printed out the loss function of the model with the best R2 score. In the model I used the ReLu activation function and the last layer is 1 node, and has a 'linear' activation function.

After creating the model, we then compile the model with an *'adam'* optimizer, and a loss function - in this case *'mse'* , or mean squared error. We then fit the model with our features

and targets, and specify the number of epochs - or training cycles. After training our model we plot the loss vs. epochs as we want to see that the loss has flattened out.

After this I have calculated the R^2 metric and plotted the prediction vs. actual values, which are displayed below:

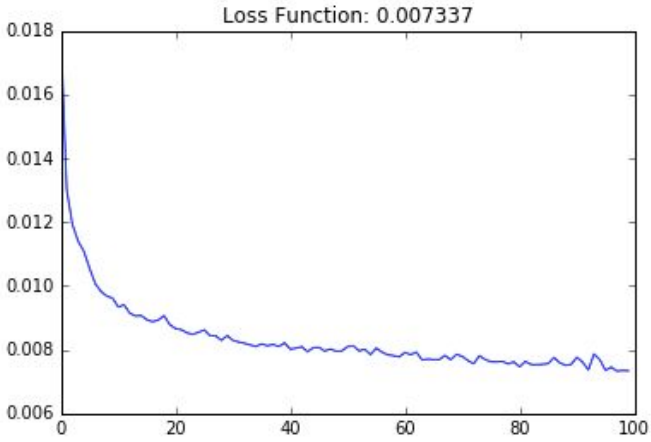
```
0.2665895316547897
-0.15807545123265498
```



IV. Results

Model Results

In order to validate the neural network model's robustness, I tested 25, 50, 75, and 100 epochs, as well as [10, 20, 1] and [15, 20, 1] for the number of layers. From these tests I see that 100 epochs with [15, 20, 1] layers has the best loss function at 0.00792. We also see the mean and the variance for this model are 0.0249 and 0.0019 respectively.



From these tests, we also see that the mean, variance, and loss functions do not change very much for each model. As we can see from these models, the R^2 measure is low for all of the models. Since all of these models are relatively simple, however, these results are to be expected. The first step towards improving the models would be adding more features. Other features that could be added include volume, fundamental data, and more engineered technical features using Ta-Lib. Another possible way to improve our neural network model is with custom loss functions.

The final model I have chosen for this project is the neural network. Although it does not have an R^2 higher than the benchmark of 0, neural networks have been shown to outperform all other machine learning models and the parameters can be changed to improved performance.

Model Refinement

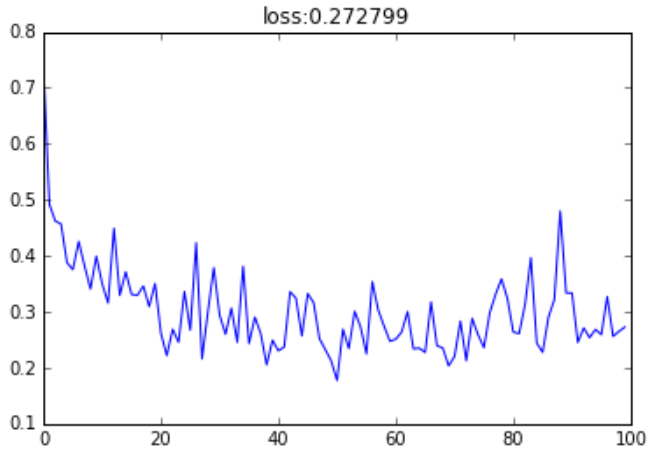
In order to try and refine the model I will make use a of a custom loss function. Since direction is an important aspect of price change prediction, one way to guide our neural network is to apply a penalty to incorrect direction prediction. We can still use the mean squared error (MSE) penalty, but if the direction is wrong we'll multiply the loss by a penalty factor.

In order to implement custom loss functions we will use the TensorFlow backend again. We then create the loss function as a Python function, which we're calling *mean_squared_error.* We pass in *y_true*, and *y_pred* as parameters for the actual and predicted values. We then take

the difference of actual and predicted values, and square the result with *tf.square*. Following this we take the average of that value with *tf.reduce_mean* across the -1 axis, which is the last axis. We then return that value as the loss.

The next step is to enable to use of our loss value by importing keras.losses. We set this function as part of keras.losses, which makes it available for use in our model. We then fit the model with our MSE loss function. We can now add a penalty for incorrect direction predictions. We first check if the sign of the prediction and the actual value differ. The correct direction is represented by a positive value, and a negative value means the model has chosen the wrong direction.

We now create our custom loss function called *sign_penalty* and use a penalty value of 10. We then create a conditional statement and return the average of our squared errors, and add our loss function to keras.losses. The results of the first custom loss function are displayed below:



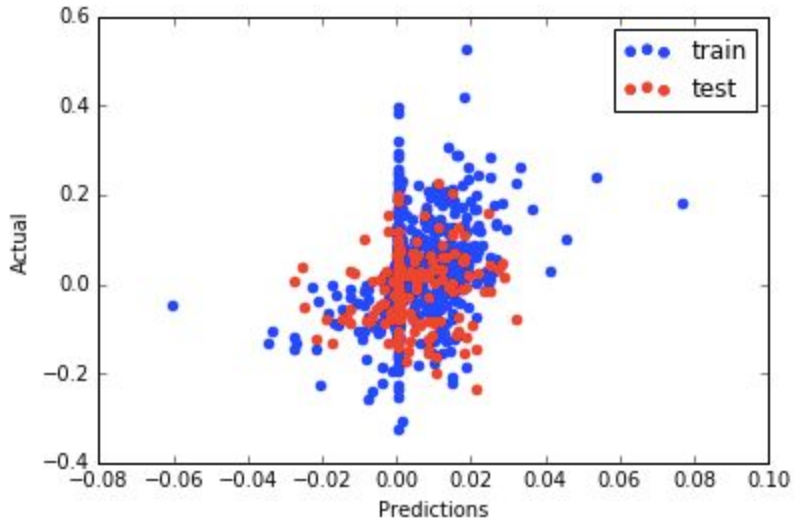
Conclusion

To summarize this project: we started with the problem of single asset price prediction based on historical data. Several features of the data were visualized to understand it better and then the data was preprocessed for machine learning. We then then implemented two different

machine learning models: OLS linear regression and a neural network model with Keras and TensorFlow. The R^2 metric for each model was recorded, which provides a starting point for improving the models in the future. The goal of this paper was not necessarily to find a model that will outperform the markets, but rather to research how regression-based machine learning models can be applied to the financial markets. The possible ways to improve the models include adding more features, and creating a custom loss function for the neural network.

For the final model, we can see that the loss function actually got worse by introducing a custom loss function. The following is the R^2 score and a scatter plot of the custom loss neural network model:

```
0.011221982594932567
-0.015075338026476981
```



As we can see the did not get better with the custom loss function better and the R^2 metric is roughly the same. Further improvements to our model could be changing the type of neural network algorithm used. In this case I used a regression algorithm, although techniques that I researched and would consider using are Deep Reinforcement Learning. This type of neural networks uses a state-action-reward function and have become increasingly popular recently. As I continue to study machine learning applied to Finance this will certainly be an area of focus.

V. References

1. JP Morgan, “Big Data and AI Strategies Machine Learning and Alternative Data Approach to Investing”, [Online]: Available: http://valuesimplex.com/articles/JPM.pdf

2. Artificial Neural Networks. [Online]. Available: https://en.wikipedia.org/wiki/Artificial_neural_network

4. What is R-Squared? [Online]. Available: https://www.investopedia.com/terms/r/r-squared.asp

5. Quandl, Bitfinex. [Online]. Available: https://www.quandl.com/data/BITFINEX-Bitfinex

6. Machine Learning for Finance with Python. [Online]. Available: https://www.datacamp.com/courses/machine-learning-for-finance-in-python

8. Scikit-learn: Logistic Regression. [Online] Available: http://scikit-learn.org/stable/modules/linear_model.html#logistic-regression