Machine Learning Engineer Nanodegree

Capstone Project – Crypto Predictor for Literal Day Trading

John Barney

12/28/2019

I. Definition

Project Overview

In the article Day Trading: What It Is and Is It Right For You? on the website nerdwallet.com, JAMES ROYAL, PH.D defines day trading as, “the practice of buying and selling stocks in a short timeframe, typically a day. The goal is to earn a tiny profit on each trade and then compound those gains over time.” He goes on further to say, “successful day traders treat it like a full-time job, not merely hasty trading done between business meetings or at lunch.”

This project aims to develop a model that accurately predicts the future closing price of a cryptocurrency, Litecoin to be specific. Therefore, this project aims to make day trading profitable with a very part time approach. For example, models would be run at the end of each day for trades to be executed at the beginning or end of the next day.

Deep learning models have been and continue to be developed to predict future prices of stocks and cryptocurrencies. One such model I’ll be referencing throughout this report comes from GitHub user khuangaf’s repository, CryptocurrencyPrediction. The data used in khuangaf’s models contain 5 features at every 5 minutes. Khuangaf’s models go on to predict the price for the next 80 minutes.

The data used in this project contains 32 features taken at the closing of each day. The model’s in this project predict the next day closing price or the next 30 days closing prices.

Problem Statement

The purpose of this project is to create a model that more accurately predicts the future price of a cryptocurrency through a classification model that takes in 32 features as input and produces the predicted closing price in USD for either the following day or the following 30 days.

Again, a key difference between the data used to train khuangaf's model's and the data used to train the models in this project are the number of features, with khuangaf's data coming from poloniex and containing only 5 features. Please see khuangaf's Data Collection Notebook at https://github.com/khuangaf/CryptocurrencyPrediction/blob/master/DataCollection.ipynb. Whereas the data my models are trained with contains 32 features. Please see a detailed explanation of all the features in the table from the data exploration section below. The data this project uses is able to have much more features because it is using data gathered at the end of the day.

This project uses Amazon's built in model XGBoost predict the next day closing price. This project also uses Amazon's built in PCA model for dimensionality reduction in conjunction with XGBoost. To predict the next 30 days closing price, Amazon's built in model, DeepAR, is used.

Along with the metrics defined below, this project determines how successful a model, and this project is as a whole by simulating a $20,000 initial investment through a test dataset. The cryptocurrency is simulated being bought and sold when the model predicts increases and decreases in the next day closing price.

Metrics

The test loss this project uses is the Mean Absolute Error (MAE). MAE was chosen to keep the test loss in US Dollars. This makes it very easy to understand and visualize how far off a model's predictions are. It also gives a good understanding of how risky these predictions are.

MAE is defined by JJ in the article MAE and RMSE – Which Metric is Better? on the website medium.com as measuring "the average magnitude of the errors in a set of predictions, without considering their direction. It's the average over the test sample of the absolute differences between prediction and actual observation where all individual differences have equal weight." Please see the formula below:

$$\text{MAE} = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j|$$

As previously mentioned in the problem statement above, the other way a model is tested is by running a simulation of making trades over the test dataset. If the predicted next day price is greater than the predicted current day price and an investment isn't currently made, the cryptocurrency is purchased. On the other hand, if the predicted next day price is less than the predicted current day price and an investment is currently made, the cryptocurrency is then sold. The trades are done using the actual prices for each day.

Each trade is printed out and at the end of the test dataset the overall net gain or loss is also printed out. An initial investment of $20,000 was chosen, because it is large enough to gain or lose a significant amount, without being so large that it would become unfeasible for most readers to invest.

II. Analysis

Data Exploration

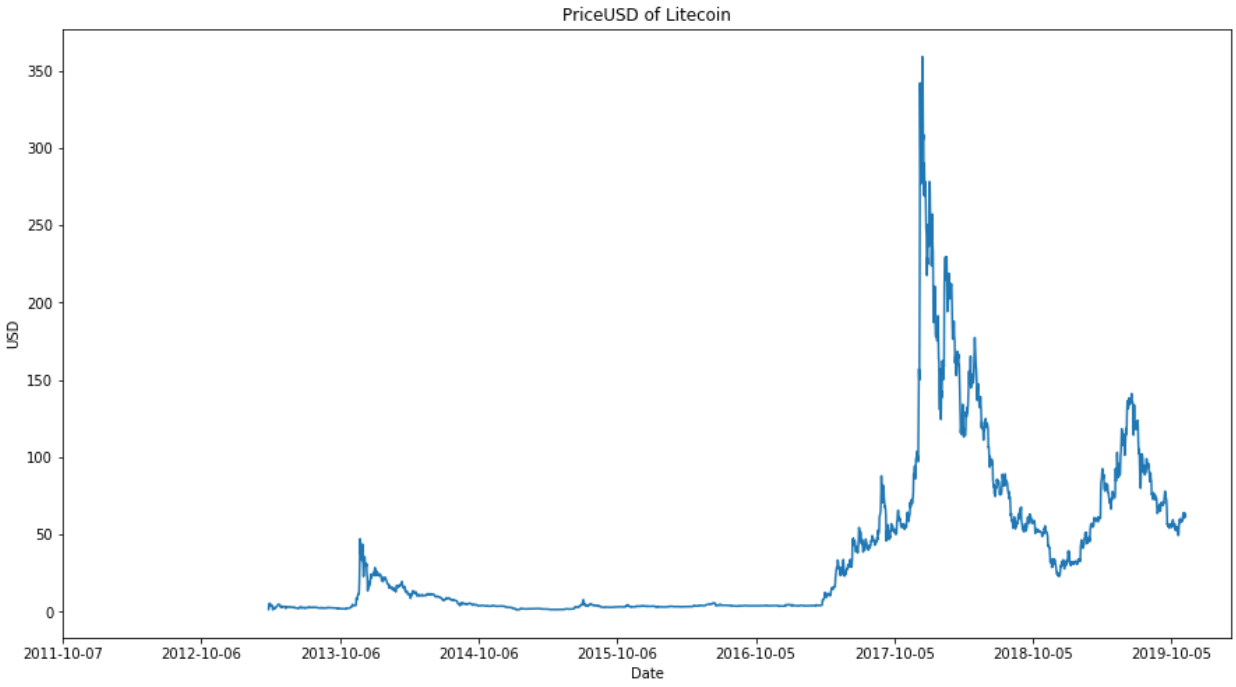
The dataset being used for this project comes from Coin Metrics and be found by following this link https://coinmetrics.io/data-downloads/, and clicking on Litecoin. This dataset contains 32 features gathered at the end of each day, beginning 10/07/2011 and ending 11/11/2019.

The feature these models predict is PriceUSD. PriceUSD has a mean of $36.14, a standard deviation of $52.66, a minimum of $1.16 and a maximum of $359.07. Unfortunately, PriceUSD is a NAN value until 4/01/2013, and because of the feature, volatility 180d, there's another 180 days (09/28/2013) until all of the features are without NAN.

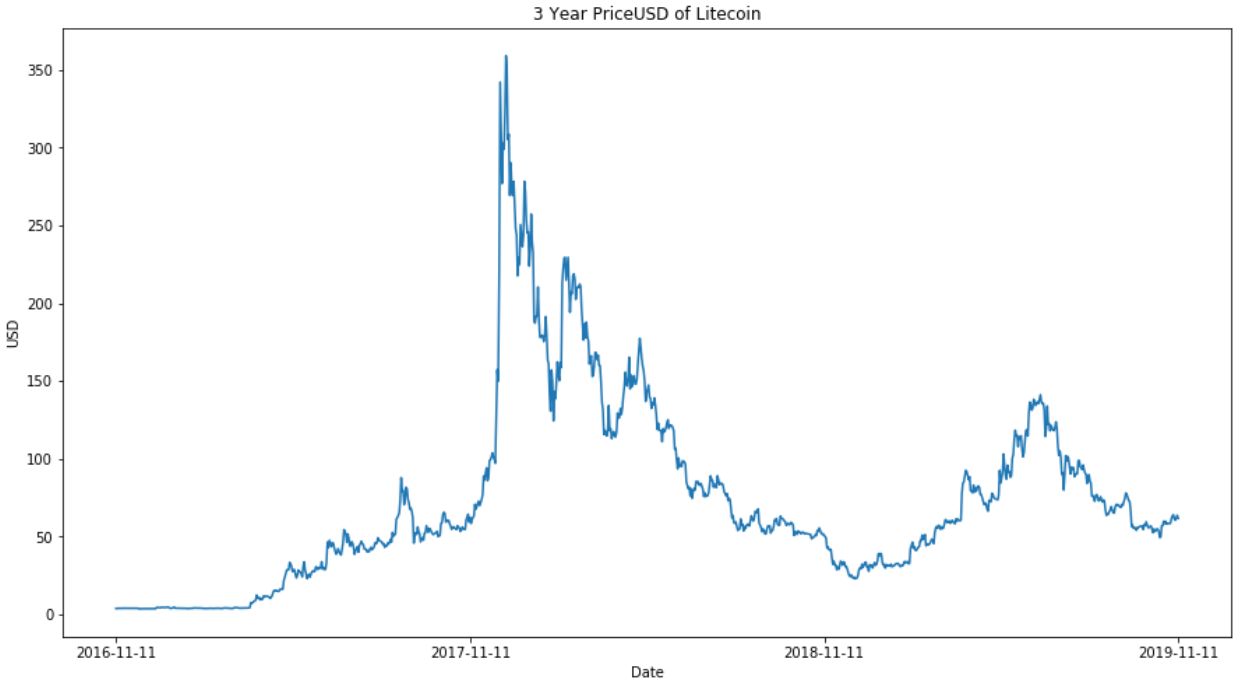
The final dataset these models use start on 11/11/2016 and end 11/11/2019. This time frame was chosen because, as you can see in the graph below, the PriceUSD feature really starts to vary a lot more towards the end of 2016, which is important when looking for a stock or cryptocurrency to practice day-trading with.

Exploratory Visualization

Below is a line graph representing the feature PriceUSD of Litecoin through the entire dataset downloaded from Coin Metrics. As you can see, there is no value, or NAN, for about the first year and a half. At the end of 2013, the PriceUSD starts to move quite a bit, but then it flattens back out towards the end of 2014 and it stays flat until the beginning of 2017. After that, the PriceUSD of Litecoin really starts to move a whole lot.



Below is a line graph representing the PriceUSD of Litecoin through the 3-year dataset the models in this project all use. As you can see, it starts out a little slow, but then around the beginning of 2017, it becomes very volatile. The highly volatile nature of Litecoin is the reason why it was chosen for this project. Accurately predicting the future movement of such a volatile cryptocurrency, could be extremely profitable.



Below, is a table downloaded from Coin Metrics, containing brief descriptions of each feature found in the dataset used in this project. As you can see, a lot of the features here are very similar to one another.

For example, NVTAdj is defined as the ratio of the network value (or market capitalization, current supply) divided by the adjusted transfer value, also referred to as NVT. This is very similar to NVTAdj90 that just moves the ratio of the network value to the 90-day average, and is still divided by the adjusted transfer value.

These similarities amongst the features, lead me to believe that dimensionality reduction through either PCA or deductive reasoning would be very effective. Please see the results below.

Short name	Metric name	Definition
AdrActCnt	Addresses, active, count	The sum count of unique addresses that were active in the network (either as a recipient or originator of a ledger change) that day. All parties in a ledger change action (recipients and originators) are counted. Individual addresses are not double-counted if previously active.
BlkCnt	Block, count	The sum count of blocks created that day that were included in the main (base) chain.
BlkSizeMeanByte	Block, size, mean, bytes	The mean size (in bytes) of all blocks created that day.
CapMVRVCur	Capitalization, MVRV,	The ratio of the sum USD value of the current supply to the sum "realized" USD value of the current supply.

	current supply	
CapMrktCurUSD	Capitalization, market, current supply, USD	The sum USD value of the current supply. Also referred to as network value or market capitalization.
CapRealUSD	Capitalization, realized, USD	The sum USD value based on the USD closing price on the day that a native unit last moved (i.e., last transacted) for all native units.
DiffMean	Difficulty, mean	The mean difficulty of finding a hash that meets the protocol-designated requirement (i.e., the difficulty of finding a new block) that day. The requirement is unique to each applicable cryptocurrency protocol. Difficulty is adjusted periodically by the protocol as a function of how much hashing power is being deployed by miners.
FeeMeanUSD	Fees, transaction, mean, USD	The USD value of the mean fee per transaction that day.
FeeMedUSD	Fees, transaction, median, USD	The USD value of the median fee per transaction that day.
FeeTotUSD	Fees, total, USD	The sum USD value of all fees paid to miners that day. Fees do not include new issuance.
IssContNtv	Issuance, continuous, native units	The sum of new native units issued that day. Only those native units that are issued by a protocol-mandated continuous emission schedule are included.
IssContPctAnn	Issuance, continuous, percent, annualized	The percentage of new native units (continuous) issued on that day, extrapolated to one year (i.e., multiplied by 365), and divided by the current supply on that day. Also referred to as the annual inflation rate.
IssTotUSD	Issuance, total, USD	The sum USD value of all new native units issued that day.
NVTAdj	NVT, adjusted	The ratio of the network value (or market capitalization, current supply) divided by the adjusted transfer value. Also referred to as NVT.
NVTAdj90	NVT, adjusted, 90d MA	The ratio of the network value (or market capitalization, current supply) to the 90-day moving average of the adjusted transfer value. Also referred to as NVT.
PriceBTC	Price, BTC	The fixed closing price of the asset as of 00:00 UTC the following day (i.e., midnight UTC of the current day) denominated in BTC.
PriceUSD	Price, USD	The fixed closing price of the asset as of 00:00 UTC the following day (i.e., midnight UTC of the current day) denominated in USD. This price is generated by Coin Metrics' fixing/reference rate service.
SplyCur	Supply, current	The sum of all native units ever created and visible on the ledger (i.e., issued) as of that day. For account-based protocols, only accounts with positive balances are counted.
TxCnt	Transactions, count	The sum count of transactions that day. Transactions represent a bundle of intended actions to alter the ledger initiated by a user (human or machine). Transactions are counted whether they execute or not and whether they result in the transfer of native units or not (a transaction can result in no, one, or many transfers). Changes to the ledger mandated by the protocol (and not by a user) or post-launch new issuance issued by a founder or controlling entity are not included here.
TxTfr	Transactions, transfers, count	The sum count of transfers that day. Transfers represent movements of native units from one ledger entity to another distinct ledger entity. Only transfers that are the result of a transaction and that have a positive (non-zero) value are counted.
TxTfrValAdjNtv	Transactions, transfers,	The sum of native units transferred that day removing noise and certain artifacts.

	value, adjusted, native units	
TxTfrValAdjUSD	Transactions, transfers, value, adjusted, USD	The USD value of the sum of native units transferred that day removing noise and certain artifacts.
TxTfrValMeanNtv	Transactions, transfers, value, mean, native units	The mean count of native units transferred per transaction (i.e., the mean "size" of a transaction) that day.
TxTfrValMeanUSD	Transactions, transfers, value, mean, USD	The sum USD value of native units transferred divided by the count of transfers (i.e., the mean "size" in USD of a transfer) that day.
TxTfrValMedNtv	Transactions, transfers, value, median, native units	The median count of native units transferred per transfer (i.e., the median "size" of a transfer) that day.
TxTfrValMedUSD	Transactions, transfers, value, median, USD	The median USD value transferred per transfer (i.e., the median "size" in USD of a transfer) that day.
TxTfrValNtv	Transactions, transfers, value, native units	The sum of native units transferred (i.e., the aggregate "size" of all transfers) that day.
TxTfrValUSD	Transactions, transfers, value, USD	The sum USD value of all native units transferred (i.e., the aggregate size in USD of all transfers) that day.
VtyDayRet180d	Volatility, daily returns, 180d	The 180D volatility, measured as the deviation of log returns
VtyDayRet30d	Volatility, daily returns, 30d	The 30D volatility, measured as the deviation of log returns
VtyDayRet60d	Volatility, daily returns, 60d	The 60D volatility, measured as the deviation of log returns
Short name	Metric name	Definition

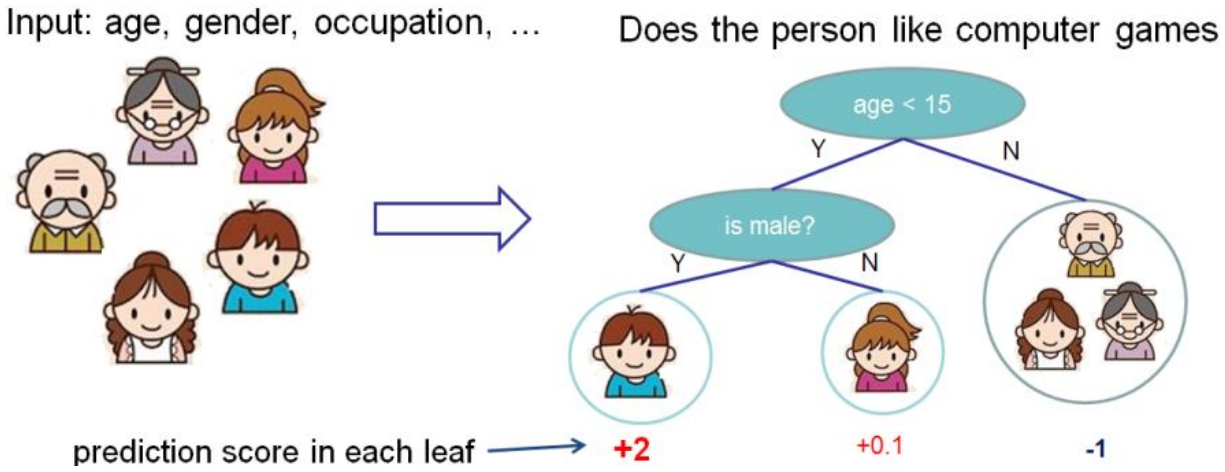
Algorithms and Techniques

The models used in this project are all built-in models from Amazon Web Services (AWS). These built-in models used are XGBoost, DeepAR, and PCA.

XGBoost is a supervised learning algorithm that performs well for problems involving regression, classification and ranking. XGBoost stands for Xtreme Gradient Boosting. Gradient boosting differs from models using stochastic

gradient descent (SGD) where SGD uses a fixed function and optimizes the parameters. Gradient boosting uses an ensemble, or collection of functions, then optimizes both the function and the parameters. Each function is created sequentially so that they learn from the mistakes of previous functions. This makes each function learn the most from the biggest mistakes, or observations with the largest errors. By learning from the mistakes of the previous functions, less iterations are needed to converge on a solution than would be if the functions were created independently like in gradient bagging. This does however allow for overfitting to occur when stopping criteria isn't carefully chosen.

XGBoost structures its ensemble into gradient boosting trees, turning the model as a whole into a classification and regression tree (CART). In the example below, the challenge becomes how to split the leaves. You could add a branch like is done with the [is male?] node, or you could create a new tree to accommodate [is male?] and occupation with [is nurse?], and you could do this for every possible split then choose the tree which has the lowest loss. This becomes very inefficient, especially when dealing with data that has many features. XGBoost bypasses this inefficiency by looking at the distribution of features across all data points in a leaf before searching for possible splits.



This project uses XGBoost for regression by training to predict the next day closing price of Litecoin. Only train and test datasets are used for all models trained, unless using AWS's hyperparameter tuning feature, in which case, a validation dataset is also used. The test sets are all for 110 days, and predictions are made for each day in the test dataset. A test loss is then calculated along with a simulation of buying and selling throughout the test set. Comparison's of all the models resulting test loss and investment simulation are all compared below in the Results section.

The hyperparameters used for all XGBoost models in this project are max_depth=5, eta=0.2, gamma=4, min_child_weight=6, subsample=0.8, objective='reg"squarederror', and early_stopping_rounds=10. Max_depth of 5 means that each tree in the ensemble is limited to being no more than 5 levels deep. Limiting the depth keeps the model from becoming too complex and overfitting the training data. Eta shrinks the feature weights to make boosting process more conservative. Gamma refers to the minimum loss reduction required to make a split on a leaf node, of course the larger the gamma is the more conservative the algorithm will be. Min_child_weight refers to the minimum sum of instance weight need in a child. If a tree partition results in a leaf node with the sum of the instance weight less than the min_child_weight then the leaf node will not be split any further. The larger the min_child_weight the more conservative the algorithm will be. Subsample refers to the ratio of the training instances that XGBoost randomly samples prior to growing trees. At 0.8, XGBoost randomly samples 80% of the training data. This is to prevent overfitting. The objective of reg:squarederror means that we are training for

regression with a squared error loss. Early_stopping_rounds stop the training of the model if it doesn't improve after this fixed number of training rounds, to prevent overfitting.

In this project AWS's PCA is trained to reduce dimensionality while still maintaining 95% of the variance in the dataset. AWS defines PCA as "an unsupervised machine learning algorithm that attempts to reduce the dimensionality (number of features) within a dataset while still retaining as much information as possible. This is done by finding a new set of features called *components*, which are composites of the original features that are uncorrelated with one another." This project then trains XGBoost on the components obtained from PCA to predict the next day closing price. Again, a test loss as well as an investment simulation is then calculated through the 110-day test dataset.

AWS describes DeepAR as "DeepAR forecasting algorithm is a supervised learning algorithm for forecasting scalar (one-dimensional) time series using recurrent neural networks (RNN)." RNN is a deep learning neural network where connections between nodes from a directed graph along a sequence allowing them to exhibit temporal dynamic behavior for a time sequence. DeepAR uses a type of RNN called long-short-term memory (LSTM). LSTM's hidden layers are composed of cells with input, output and forget gates. The input gate regulates what information flows into the cell, the output gate regulates the information flowing out of the cell, and the forget gate regulates what information flows out of the cell completely. Of the many different types of forecasts that RNN's produce, DeepAR models produce probabilistic forecasts, which output a probabilistic range along with each prediction. For example, in this project, the DeepAR models output 30-day predictions, the prediction for each day consists of a median value as well as a percentile range of 80.

In this project DeepAR is trained to predict the next 30 days (prediction length) closing prices, while seeing the previous 30 (context length). It is trained and tested on 10 quarters of data, from quarter 2, 2017 through quarter 3, 2019. The first 60 days of each quarter are the train dataset and the last 30 days are used as the test dataset. When the training is done the last quarter is used for obtaining the test loss and investment simulation.

In addition to the context and prediction lengths mention above the hyperparameters used for the DeepAR models in this project are as follows:

- 50 epochs
- a time frequency (time_freq) of daily (D)
- the number of cells (num_cells) to be used in each hidden layer of 50
- the number of hidden layers (num_layers) of 2
- the size of the mini-batches (mini_batch_size) used during training of 128
- the learning rate (learning_rate) used in training of 0.001
- the number of epochs with no progress made before stopping training of 10 (early_stopping_patience)

Benchmark

The test loss and investment simulation will be used to compare models with one another. With the test dataset having a mean of $68.56, and, as already proclaimed in the capstone proposal, the goal of having a test loss of less than $2.00; predictions being made by these models aim to be less than an average of 3% off the actual values (2.917% to be exact).

The investment simulation, as described above, will start with a $20,000 initial investment and simulate buying and selling throughout the test dataset, each time the respective model predicts movement up or down. The test dataset has a -35% change between the start and end of the dataset. If we invested $20,000 at the beginning and made no other moves, we would lose $7,060. These models aim to provide at least a 25% return on investment

through the test dataset. This would be an at least $5000 gain over 110 days, just less than doubling the investment through an entire year and providing a 60% increase from the initial strategy (or lack thereof).

III. Methodology

Data Preprocessing

As described in the Data Exploration section above, the first about year and a half of the data consisted of NAN's for the PriceUSD feature. NAN's were found throughout other features in the dataset during that time and continued until 180 days after the PriceUSD. Also, except for a little movement at the end of 2013, the PriceUSD feature stayed very flat until the beginning of 2017. To account for these NAN's and lack of volatility in the early going of the dataset, it was condensed to the most recent 3 years for all models.

All models, with the exception of the DeepAR models, were split by 10% of the dataset to make 110-day test datasets. The models trained with hyperparameter tuning were split again to make a 110-day validation dataset. The PCA algorithm needed all features in the dataset to be scaled between 0 and 1. Once trained the PCA algorithm provided 11 components, or features, which accounted for over 95% of the explained variance in the dataset. An XGBoost model was trained with a shuffled train dataset, but it was not found to improve performance, so the remainder of the models were trained without shuffling the train dataset. An XGBoost model as well as a DeepAR model used a dataset with a handful of features dropped through deductive reasoning.

The DeepAR models required data to be in the form of a json object with a start date, a list of targets, and optionally a list of dynamic features. The data had to be further split up into a series of train and test datasets. All in all, the data was split into a series of 10 train and test datasets. Each train set was 60 days and the test set spanned the entire 90 days of each dataset series. These were split up by quarters, with the first quarter being quarter 2, 2017 and the last quarter being quarter 3, 2019.

Implementation

The data used by the XGBoost models were split using sklearn's algorithm train_test_split, in the model_selection library. The validation sets used when performing hyperparameter tuning were obtained in the same way. They were trained with the objective reg:squarederror for 200 rounds, with the exception of one model trained with 220 rounds. When done training a transformer was defined and then used to transform the test dataset to obtain the predictions. The test loss was obtained by comparing the predictions to the test labels and taking the mean absolute error (MAE), as well as the return on investment through the investment simulation.

The PCA algorithm trained with the entire 3-year dataset. Before it could train, the dataset was scaled down to be between 0 and 1. The model was trained for 1 component less than the number of features in the dataset. Once trained the model attributes of the mean, the makeup of the principal components, and the singular values of the components were obtained. Then the explained variance was calculated with the singular values and number of components. The number of components was increased until, the explained variance was calculated to be over 95%. A PCA predictor was then deployed and used to create a dataset consisting of the number of components found to contain over 95% of the variance. This dataset was then split into a train and test dataset and trained using XGBoost in the same way that it was described above. Again, a transformer was defined and used to

transform the test dataset into predictions. These predictions were again used along with the test labels to obtain the test loss and perform an investment simulation.

Hyperparameter tuning was performed using the XGBoost algorithm. The datasets were split in the same way as mentioned above, only this time it was split for a second time to obtain a 110-day validation set. The tuner was trained using the training dataset to lower the validation loss as much as possible. The best training job, the one with the lowest validation loss, was then used to define a transformer and transform the test dataset to obtain the predictions. The predictions and test labels were then compared to find the test loss as well as perform the investment simulation.

20 jobs training jobs were used with a maximum of 3 jobs trained at the same time for all tuning jobs. Each of the hyperparameter tuning jobs scanned over the following hyperparameters' ranges:

- max_depth: 3, 12 integers
- eta: 0.05, 0.5 continuous values
- min_child_weight: 2, 8 integers
- subsample: 0.5, 0.9 continuous values
- gamma: 0, 10 continuous values

The hyperparameter tuner with the 110-day validation dataset had a test loss of $4.61 with the following hyperparameters:

- max_depth: 10
- eta: 0.2285
- min_child_weight: 5
- subsample: .826
- gamma: 2.72

The hyperparameter tuner with the 30-day validation dataset had a test loss of $5.25 with the following hyperparameters:

- max_depth: 10
- eta: 0.062
- min_child_weight: 8
- subsample: 0.58
- gamma: 0.3199

For the DeepAR models the dataset, once reduced to 3 years, was split into 10 quarters. The quarters were split into 60-day by 30-day train and test datasets. Each of these were then converted into a json object while being split into the start (date of the beginning of the dataset), the target (list of closing day prices), and dynamic features (the rest of the features in the dataset). These train and test datasets were then used to train the DeepAR estimator. Once trained the estimator was deployed and used to obtain predictions on the test datasets. These predictions were decoded to form quantiles of 0.1, 0.5 (the median), and 0.9 for each of the test datasets. These quantile predictions were then displayed graphicly along with their respective test labels. Then the predictor was used to make predictions on a test dataset it had not seen yet, the beginning of the fourth quarter, 2019. Quantiles were predicted and displayed along with the respective test labels. A test loss and investment simulation were obtained when comparing the median quantile predictions to the test labels.

Refinement

The initial XGBoost model was refined by increasing the number of rounds trained from 200 to 220. This didn't lower the test loss, but it did however increase the return on investment through the investment simulation. After that the XGBoost model was refined by simply not shuffling the train dataset. The subsequent XGBoost model that didn't have a shuffled training set improved in both the test loss and the investment simulation. Again, the number of rounds was increased to 220. This model performed the best with both the test loss and investment simulation. The feature PriceBTC was then dropped, due to how noisy it appeared to be. This didn't improve either the test loss or the investment simulation.

AWS's Hyperparameter Tuning was also performed on the XGBoost models. This however didn't improve any of the models. Different sized test and validation sets were attempted here, but again with no improvement.

Another thing done to attempt to refine these models was to reduce dimensionality. One way the dimensionality was reduced was by training a PCA algorithm on the dataset and using that to obtain components that explained the variance of 95% of the dataset. Those components were then trained and tested by an XGBoost model. The other way reduced dimensionality was obtained was by deductively reasoning through taking out the features whose definitions were too similar. Neither of these were able to improve the performance of the XGBoost models. The test loss did however lower when done with the DeepAR model, though it was still much larger than all of the XGBoost models.

A DeepAR model was also trained with only the PriceUSD feature, completely dropping all of the other features. The test loss and investment simulation didn't improve.

IV. Results

Model Evaluation and Validation

The best performing model was the XGBoost model that was trained on the unshuffled dataset and trained for 220 rounds. It had a test loss of $2.21 and a return on investment through the test dataset of $4,308.57.

This model was tested for robustness by forming predictions on a year of data from the period of high volatility in 2014, as previously discussed in the Data Exploration section. The test loss for this period of time was $5.23 and the total return on investment through the simulation was $6,974.70, though this could have been much higher, as at one point in the simulation it rose to $89,047.04. This model was also tested for robustness by predicting the most current year (2019) for bitcoin. The test loss for this was $7,136.50 and the return on investment was $7,456.43

I believe the results from this model can be trusted, though the model itself should be retrained on the most current data often. Also, this model should only be used to predict the future price of Litecoin.

Justification

Unfortunately, the best performing model did fall just short of both the test loss and investment simulation benchmark. With a test loss of $2.21, it just missed the goal of under $2.00. Though it didn't hit the benchmark, the test loss was still only 3.2% off of the mean price for the test dataset.

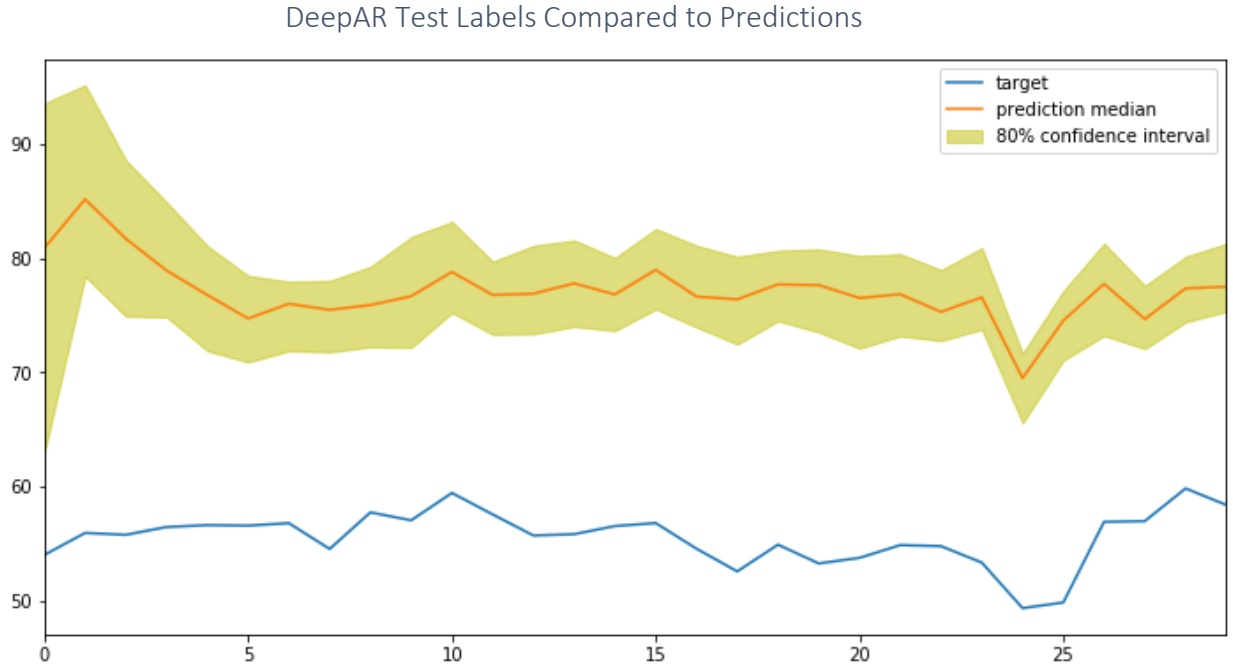
The investment simulation netting a $4,308.57 return on investment also fell just shy of the $5,000 goal. With a 35% decrease from the beginning through to the end of the test dataset, a $4,308.57 (up 21%) net gain still isn't a bad result, especially when considering the 35% decrease on a $20,000 investment would equate to a $7,000 loss.

V. Conclusion

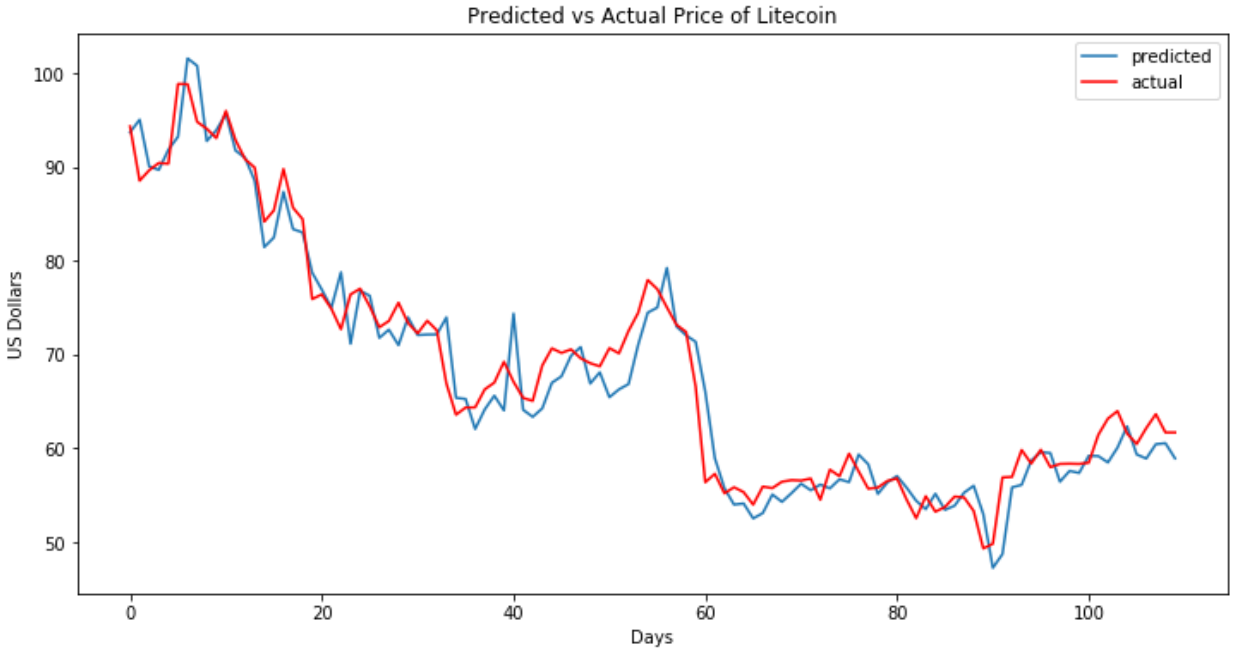
Free-Form Visualization

Model	Test Loss	Return on Investment
XGBoost Unshuffled 220 Rounds	2.21	$4308.57
XGBoost Shuffled	2.31	$825.85
XGBoost Shuffled 220 Rounds	2.42	$1970.34
XGBoost Unshuffled	2.22	$3552.97
XGBoost w/o PriceBTC	2.41	$2433.16
XGBoost Dimensionality Reduced	4.44	-$4080.84
Hyperparameter Tuner 110-day Test & Validation Datasets	4.61	-$1204.61
Hyperparameter Tuner with 30-day Validation Dataset	5.24	$1897.86
Hyperparameter Tuner with Shuffled 110-Day Validation Dataset	3.01	-$1158.56
PCA 95% Variance	7.46	-$5193.43
DeepAR	21.58	$7113.64
DeepAR Dimensionality Reduced	20.65	$-1643.39
DeepAR No Dynamic Features	29.21	$4917.45

The above table shows the test losses along with the return on investment from the investment simulation performed at the end of training each model. An interesting thing to note here is how well the DeepAR model did with the investment simulation, far outperforming the best XGBoost model and even outdoing the $5000 benchmark. It is important to consider that the test dataset was only 30 days for the DeepAR models compared to the 110-day test set for the XGBoost models. The test dataset for DeepAr also saw 7.5% growth from the beginning through to the end of the dataset. As you can see in the graph below, even though the predictions are a far way off from the predicted values, it still mirrors the movement of the price surprisingly well.



Below is a graph of the best performing model in this project, the XGBoost model trained with unshuffled data for 220 rounds. As you can see, the predictions are very close to the test labels. The predictions also do a good job of mirroring the movement of the test labels.



Reflection

Though a lot of the same steps were taken to train and test each of the models, some key differences and problems did arise. For example, the data used in the DeepAR models needed to be preprocessed much differently than that of the XGBoost models. With the XGBoost models taking in simple csv files and the DeepAr models requiring json objects. The output for these models also had to be handled much differently. Again, DeepAr outputted json objects which had to be decoded. Once decoded, they weren't the simple 1-dimensional predictions obtained from the XGBoost models, they instead consisted of 3 dimensions, quartiles of your choosing. This ended up enabling much better graphical representations, enabling the ability to show the range of the predictions. And, even though DeepAR's test losses were way off from the test losses obtained by the XGBoost models, two of them were however more profitable through the investment simulation.

This project trained many different models in search of a model that accurately predicted the future price of Litecoin well enough to make once daily trades a profitable and non-time consumptive investment option. Again, though it didn't quite reach the benchmark I set out to, I think it did perform well enough to be considered a legitimate investment.

Improvement

When investing it is more important to know whether the price is going up or down than it is to know the exact value. That is how the DeepAr model did so well in the investment simulation, even though it had such a bad test loss. It would be interesting to train an XGBoost model to predict growth or decline through binary classification. The same dataset could be used, all that would need to be done is change the labels from each day to either a 0 or a 1 depending on whether the price went up or down for that subsequent day. I think this could provide a more profitable solution.