Anders Fischer-Nielsen

Bitcoin Price Predictor

Assignment 2 - COMP809

Purpose

The purpose of this data mining project is to determine, given a dataset which tracks the movement of bitcoin over a period of time spanning close to a year a and half, whether it is possible to accurately predict the weighted price of bitcoin on an hourly basis. The prediction will be performed by developing three data mining models over the dataset and evaluating their performance by looking at regression metrics. Long Short-Term Memory recurrent neural network (LSTM), Linear Regression (LR) and Support Vector Regression models have been developed and evaluated in this project.

Data

The dataset is given in the form of a `CSV` file, containing 10776 data samples with 8 features each.

The Dataset

The dataset shows the movement of bitcoin spanning over the time period from `01/01/2017` to `25/03/2018`. A section of the raw dataset can be seen below.

Timestamp	Open	High	Low	Close	Volume_BTC	Volume_Currency	Weighted_Price
1/01/2017 10:00	961.68	963.54	960.74	962.93	77.19857878	74300.08902	962.8058379
1/01/2017 11:00	962.93	964.18	962.01	963.58	174.8056494	168584.3686	963.265489
1/01/2017 12:00	963.59	966.3	963.59	966.3	171.1138657	165313.3274	964.5229141

Predicting the `Weighted_Price` feature is the goal of this project.

- `Timestamp` indicated the time of day the remaining features were extracted and represents a one-hour timestep.
- `Open` is the opening price at the time of measurement.
- `High` is the highest price observed in the one-hour timestep.
- `Low` is the lowest price observed in the one-hour timestep.
- `Close` is the closing price observed in the one-hour timestep.
- `Volume_BTC` is the volume of bitcoins present in the bitcoin network at the time of measurement.
- `Volume_Currency` is the volume of actual currency (assumed to be USD) present at the time of measurement.

The `High`, `Low` and `Close` features are unknown at the beginning of the one-hour timestep, and have been measured at the end of the timestep. The `Open` feature shows the `Close` feature of the previous timestep, since this known when the one-hour timestep measurement is initiated.

Preprocessing

Given the nature of the dataset, the `High`, `Low` and `Close` features cannot be used in practical prediction, since these are known "after-the-fact", that is in a realistic prediction setting, knowing what the highest, lowest and closing BTC values would make it trivial to predict the `Weighted_Price`. A more interesting scenario would be to predict the `Weighted_Price` for a given timestep only knowing the `Open`, `Volume_BTC` and `Volume_Currency` features available at the beginning of each timestep measurement.

The `High`, `Low` and `Close` features have therefore been removed in preprocessing. The `Timestamp` feature has been transformed into simple timestep values from $[0...N]$ where $N = datasamples = 10776$.

The dataset is split into training, testing and validation sets. The training dataset is the first 66% of the data samples according to timestamp, with the remaining 33% of the data samples being split into 23% test data for training and 10% validation data after training. The validation data set is used for generating regression metrics to estimate the performance of the generated models.

The dataset is split into subsets according the the timestamps of the data samples in an attempt to "force" the models to get improved prediction accuracy on recent data. The validation data set represents the most recent data, and would therefore be more interesting in a real-world prediction scenario, where predicting historical data is of little value. A model that predict future values is desireable. Validating on recent data with values unknown to the models will therefore be "harder" for the models and represents a real-world prediction scenario. Validaing on these recent data samples should hopefully therefore better show the inaccuracies of the generated models.

After splitting the dataset, each set has been normalized to values between 0 and 1. Normalisation has been performed after splitting in order to make sure there is no correlation between the values in the data sets in order to improve accuracy.

A section of the processed dataset cat be seen below.

	0	1	2	3
0	0.1565588569028475	0.1559126213592233	0.1570755441908829	0.15641807434215976
1	0.15676235346212766	0.15601618122977345	0.15728318199208036	0.1565236601566244
2	0.1568697996454276	0.15635922330097088	0.15754150303608974	0.1569654961802301
3	0.1572947004612046	0.15647087378640778	0.15770663231105778	0.1570142280945984

Chosen Algorithms

The Long Short-Term Memory recurrent neural network (LSTM), Support Vector Regression (SVR) and Linear Regression (LR) models have been selected. These models and their strengsth and limitations are described in the following.

LSTM

The Long short-term memory (LSTM) is an artificial recurrent neural network, (RNN) model. The model has “memory” due to having input, output and forget gates between the internal neural nodes. Using these gates, an individual node can “decide” to retain or forget information. The “vanishing gradient” problem seen in traditional RNNs, where gradients that are back-propagated can “vanish”, can be partially solved by an LSTM since it allows the gates allow the gradient to pass through unchanged, though “exploding gradients” tending to infinity can still occur.

Algorithm

Strength and Limitations

SVR

Algorithm

Strength and Limitations

LR

Algorithm

Strength and Limitations

Performance of the Models

The LSTM performs slightly better than LR with SVR in third place. The performance of the LR model is very close to the LSTM when looking at the evaluation metrics.

The raw metrics after predicting the most recent 10% of the validation data set can be seen below.

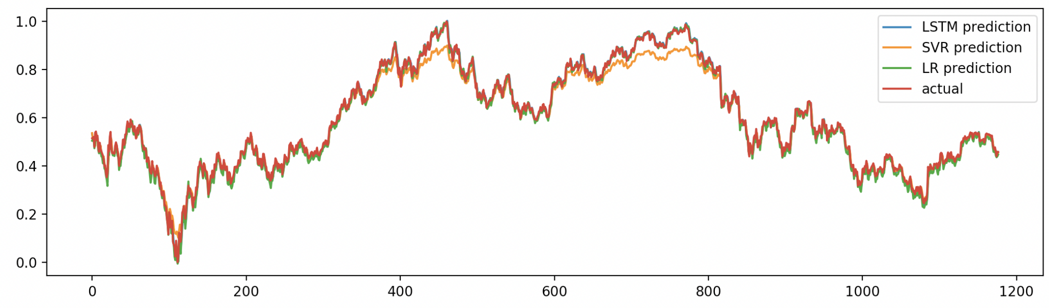
	MAE	MSE	RMSE	R^2	EV
LSTM	0.010710268	0.000207735	0.014413024	0.995107292	0.995156721
LR	0.011783905	0.000216703	0.014720851	0.994896068	0.996982989
SVR	0.022348931	0.000960744	0.030995877	0.977371961	0.983135368

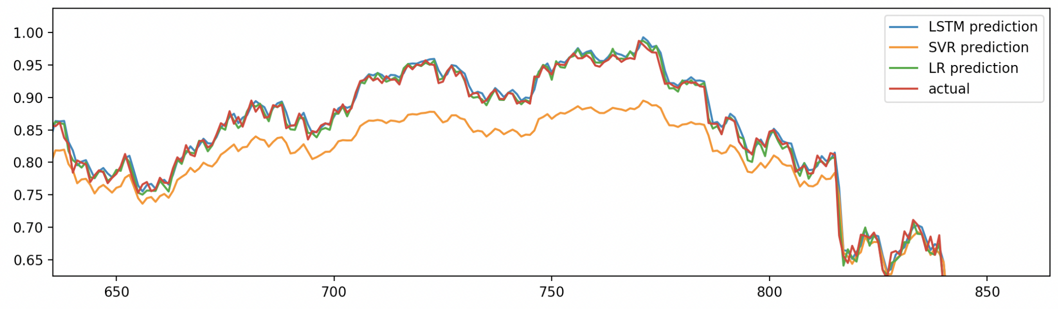
I will only look into the differences between the LSTM and LR models here since they are quite close and will require further examination. The SVR model does clearly not perform as well as the other models and the results will therefore be examined further.

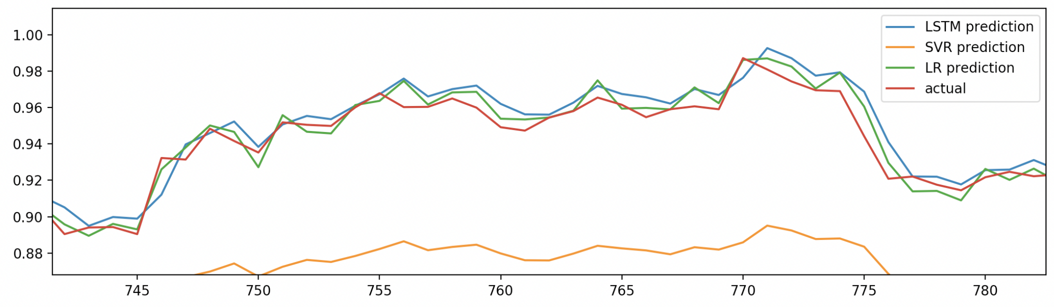
- The mean absolute error is 0.01 higher for the LR model compared to the LSTM model with the mean squared error being 0.00001 higher for the LR model than the LSTM model, indicating a slightly higher accuracy for the LSTM model.
- The root mean squared error is 0.0003 higher for the LR compared to the LSTM, indicating that the LSTM model predicts the outliers of the data set slightly better.
- The R^2 score is very close, with the LR having a 0.000217 lower score than the LSTM, indicating that the LSTM model has a slightly better *goodness of fit*.
- The explained variance score for the LR mofel is 0.00182 *higher* than the score for the LSTM, indicating that the LR model accounts slightly better for the dispersion of the data set.

Given the speed of fitting the LR model compared to the LSTM model the above results are quite impressive. Training the LSTM with 706 hidden units over 32 epochs takes 6 minutes on the test machine, while the LR model fits in seconds. The LR model looks to be a good choice if quick, relatively accurate results are desired. The improvement gained by training a LSTM might not outweigh the time it takes to train, especially given a bigger data set and/or training over a higher number of epochs.

Plots of the predictions for the three models at two zoom levels can be seen below.







The plots show that the SVR model does not perform as well as the LR and LSTM model. At first glance the LR and LSTM model looks to have close to identical performance, which the previously detailed performance metrics help show is not the case.

Experimentation

Dataset Permutation

The initial training was done with all original features present. This gave a very precise model, since the model merely had to learn how to "predict" the `Weighted_Price` from the `Open`, `High`, `Low` and `Close` features.

The normalization was initally performed across the entire dataset before splitting into subsets. This made the problem "too easy" for the models, since all data values would match the other datasets. The decision to split before normalization made sure that there is no correlation between the normalized values between subsets.

Hyperparameter tweaking

The initial experimentation of this project was using different hyperparameters for the LSTM. The main hyperparameters that have been tweaked for the LSTM have been

- `inputs` - the size (units) of the hidden layer
- `epochs` - the anumber if training epochs
- `batch_size` - the size of the amount of timesteps to take in per epoch

The size of the hidden layer along with the number of epochs unexpectedly turned out to be the most important hyperparameters. The initial experiments were performed with a small hidden layer, due to training time. The computing power required to train a large network made it infeasible to experiment with the other hyperparameters when the network was large. The number of epochs when training the network had some significance, but had a smaller impact when the number of epochs exceeded 17. Batch size seemed to have little to no effect on the results and was furthermore difficult to change due to the requirements of the LSTM library that require input dimensions of the data to match the batch size.

Experiments with adding and removing dropout in the network were also performed. The initial model had no dropout which made it prone to overfitting. A large dropout impacted the accuracy of the model negatively. Ultimately, a dropout of 10% was deemed optimal.

Planning

No explicit experimentation plan was detailed before the beginning of the project due to starting the project early. The tweaking of hyperparameters quickly turned out to produce a relatively accurate model, and training over higher levels of epochs could then be run in parallel with developing LR and SVR models.

Experiments were carried out until the LSTM model was deemed to be "good enough" - that is better in most metrics than the other models developed.

The Winner

The winner - that is the model with the best performance - has been determined as the LSTM model. As noted in the earlier performance overview, the LSTM has a slightly better performance than the LR model.

	MAE	MSE	RMSE	R^2	EV
LSTM	0.010710268	0.000207735	0.014413024	0.995107292	0.995156721
LR	0.011783905	0.000216703	0.014720851	0.994896068	0.996982989
SVR	0.022348931	0.000960744	0.030995877	0.977371961	0.983135368

Given a scenario where computing power is not abundant, the LR model would be deemed the better model. The training and experimentation on the LSTM takes time, especially on older machines.

The LR model is not far behind the LSTM model in regards to performance, but the SVR model is trailing behind.

TODO: An analytical (this can include statistical methods) comparison of the performance of the algorithms, together with an explanation of the superior performance of the winner. You may use the Experimenter module in Weka for this purpose. Your analysis should also include suitable visualizations (model diagrams, PRC curves, whatever is appropriate) that compare the performances of your winner and runner-up algorithms. Your winner should then be compared to any significant (data mining) work previously undertaken on the data set you selected (if any). In your experimental study you will have defined a number of different performance measures and these measures should (a) be used on their own and (b) combined into a single measure. To combine several measures into one use a linear weighted model, with weights to be supplied by yourself, backed up by suitable justification.