Anders Fischer-Nielsen

Bitcoin Price Predictor

Assignment 2 - COMP809

Purpose

The purpose of this data mining project is to determine, given a dataset tracking 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 (BTC) on an hourly basis. By developing three data mining models over the dataset and evaluating their performance by looking at regression metrics, predictions are found and their results can be compared. The three models that have been developed and evaluated in this project are a Long Short-Term Memory recurrent neural network (LSTM), a Linear Regression (LR) and a Support Vector Regression model.

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

The goal of this project is predicting the `Weighted_Price` feature given the remaining features.

- `Timestamp` indicates 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 are been measured at the end of the timestep. The `Open` feature shows the `Close` feature of the previous timestep since this is known when the one-hour timestep measurement is begun.

Preprocessing

Given the nature of the dataset, the `High`, `Low` and `Close` features cannot be used in a practical prediction scenario, 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, that are known at the beginning of each timestep measurement.

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

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 improve the prediction accuracy on recent data. The validation dataset consist of the most recent data points and would therefore be more interesting to predict in a real-world scenario, where predicting historical data that is already known is of little value. A model that predicts future values is therefore desireable. Validaing on these recent data samples should hopefully better show the shortcomings 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 rather than before, in order to in order to improve accuracy and make sure there is no correlation between the values between the data sets.

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 strengths and limitations are described in the following.

LR

Linear Regression is, in short, an attempt to find a linear function that models some given data points as accurately as possible. The model aims to predict a y value such that the error difference between predicted value and true value is as small as possible, and does this by trying to reach the best value that minimizes the error between predicted y value and true y value. This is accomplished using a loss function, Root Mean Squared Error (or RMSE) between the predicted and actual values. Through the use of Gradient Descent on the loss function, the optimal linear function is determined across the data points.

Strength and Limitations Linear Regression can only model a linear function, which means that if future data can not be modelled linearly, the accuracy will drop. Compared to a neural network, the network would be able to model a nonlinear and more complex trend, which the regression model would not be able to. Linear regression assumes that the data are independent, which in this scenario might no be the case. The price for a given timestep probably has a correlation to the previous timestep and is therefore not independent. The Timestep measurements might differ greatly, but would probably still have a correlation. Linear regression also has poor outlier resistance, since a single outlier can affect the resulting model significantly.

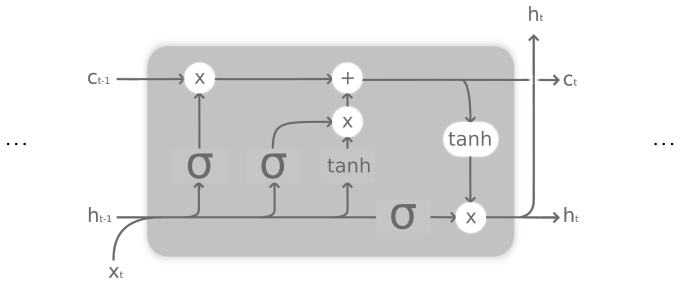
SVR

The goal of the SVR model is to find a function $f(x)$ that deviates from yn by a value no greater than ϵ for each training point x that at the same time is as flat as possible. Furthermore, the optimal function will be the function that has the maximum margin from the input data. The model achieves this by lifting the input into a space with higher dimensions with a non-linear mapping. A linear model is then constructed in the resulting space. The accuracy of the model depends on this mapping, called a kernel. The non-linearity of the kernel function allows non-linear regression, which Linear Regression would not.

Strength and Limitations The SVR model is resistant to overfitting, since the training takes place of "error points" outside of the margin of the currently found function. SVR effectively ignores the data points that are far from the decision boundary of the function. A limitation of the model is that the accuracy depends on kernel function. Picking a good kernel function is hard, since it requires a good understanding of the input. The default kernel function which currently gives the best accuracy might not be well-suited for future data, as seen in the following performance evaluation, where sudden changes in the dataset are not predicted accurately by the model.

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 inside and between the internal neural nodes of the network. Using these gates an individual node can "decide" to retain or forget information.



(Chevalier, 2018)

The *vanishing gradient* problem seen in traditional RNNs, where gradients that are back-propagated can vanish over large inputs, can be partially solved by an LSTM since it allows the gates allow the gradient to pass through unchanged, though the inverse problem of *exploding gradients*, where gradients are tending to infinity, can still occur.

Strength and Limitations One limitation of the LSTM model is that it is a black box, meaning that the model can be hard to reason about after training, unlike a Descision Tree. It is therefore hard to e.g. see which features influence the prediction, and a trial-and-error aprroach where features are included and removed randomly has to be performed in the worst case. Training the LSTM model is furthermore computationally expensive, requiring a lot of either CPU of GPU power for training when trained using a high number of epochs or with very large datasets. As mentioned previously, the exploding gradient problem is still present for LSTMs, which can result in an unusable model if this is not caught during training.

Performance of the Models

The LSTM performs slightly better than LR with SVR in last 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.

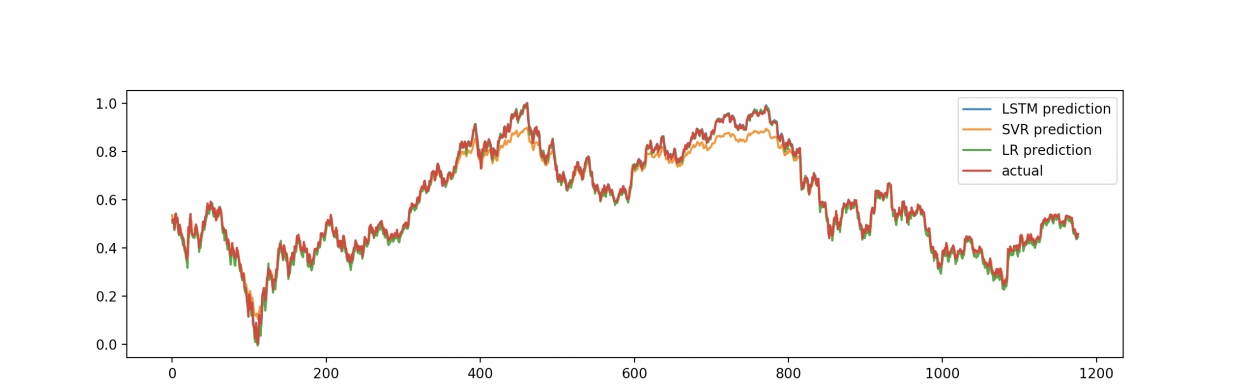
	MAE	MSE	RMSE	R^2	EV
LSTM	0.010710268	0.000207735	0.014413024	0.995107292	0.995156721
LR	0.011783905	0.000216703	0.014720851	0.994896068	0.996982989
SVR	0.022348931	0.000960744	0.030995877	0.977371961	0.983135368

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. Experiments showed that the outlier resistance of the SVR model might prevent it from learning from the price spikes in the input data, since these could be seen as outliers. This is evident in the zoom above.

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, even though the models are close.

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 unsurprisingly 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 30. 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. Dropout has been shown to prevent overfitting to a certain extent (McNally, Roche, & Caton, 2018), but a large dropout impacted the accuracy of the model negatively. Ultimately, a dropout of 10% was deemed optimal.

The size of the hidden layer was determined by using the formula $N_h = \frac{N_s}{(\alpha * (N_i + N_o))}$ where N_i is the size of the input dimension (number of features), N_o is the number of the output neurons (1), N_s is the number of data points and finally $\alpha = 2$ as an arbitrary scaling factor.

Different kernels were used for the SVR model, with the Radial Basis Function (RBF) kernel function giving the best performance. Refitting the SVR model with a different kernel function is low-cost, and therefore allowed fitting the model with all available kernel functions in order to determine the optimal function. The C parameter was also tuned in experiments, with small gains at the risk of overfitting.

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 than the other models developed when looking at most performance metrics. An Amazon EC2 instance was used temporarily for LSTM training in order to determine whether a higher number of epochs would yield better performance. Raising the number of epochs was deemed unnecessary due to the low model performance gain.

The Winner

The winner - 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 with a lack of computing power, the LR model would be the better model due to the lower computing requirements of fitting the model. The training and experimentation on the LSTM takes time, especially on slower CPUs or without available GPUs. The LR model is not far behind the LSTM model in regards to performance, with the SVR model trailing behind.

Adding the three error metrics, $MAE + MSE + 2 * RMSE$ gives a weighted model that punishes poor outlier resistance, since the RMSE metric is sensitive to outliers (Willmott & Matsuura, 2006). Due to the volatile nature of Bitcoin, developing a model that is able to predict sudden changes (outliers) is desirable. The average score of the R^2 and explained variance score is also worth looking at. For the LSTM and EV models, one models scores higher on R^2 while lower on the EV score and vice versa. Averaging the two scores shows an overall performance metric for the precision.

	$MAE + MSE + 2 * RMSE$	$(R^2 + EV)/2$
LSTM	0.039744051	0.995132006
LR	0.04144231	0.995939528
SVR	0.085301429	0.980253664

The weighted score shows that the LSTM outperforms the other models when looking at errors, but is slightly outperformed by the LR model when looking at the averaged R^2 and EV score.

The developed LSTM model seems to be slightly more outlier resistant than the LR model. Considering the performance of the LR model, and the cost of training the LSTM model, using the LR model for prototyping and gaining an understanding of the dataset is preferable. Training an LSTM model on powerful computing hardware in order to gain some precision compared to the LR model is optimal, but might not be feasible on older machines.

Previous Work

Utilizing recurrent neural networks for predicting the price of Bitcoin or stocks using historic Bitcoin and stock market information has been explored previously as described by (McNally et al., 2018), (Ebru Seyma Karakoyun, 2018), (Schultze-Kraft, 2018), (Karagiannakos, 2019), (Yalçın, 2018) and (B, 2018). The work described in this report adds Dropout to the trained model, improving the performance of the model slightly comparatively.

LSTMs have proven to be effective for predicting time-series data, though given less ressources simpler models predict with a comparable accuracy using less ressources for training.

The code for training the three models, the dataset and this report have been uploaded on GitHub in the reposity found at https://github.com/andersfischernielsen/Bitcoin-Price-Predictor.

References

B, B. (2018, November). Bitcoin price prediction with a rnn (lstm) and sentiment analysis. Retrieved from https://medium.com/@brentbiseda/bitcoin-price-prediction-with-a-rnn-lstm-and-sentiment-analysis-6bc323fa38f9

Chevalier, G. (2018, May). Illustration of an lstm cell. *File:The LSTM cell.png*. Wikmedia Commons. Retrieved from https://commons.wikimedia.org/w/index.php?curid=71836793

Ebru Seyma Karakoyun, A. O. C. (2018). Comparison of arima time series model and lstm deep learning algorithm for bitcoin price forecasting. In J. Vopava (Ed.), *Proceedings of mac 2018 in prague* (pp. 171–180). Academic Conferences Association, z.s. MAC-EITAI 2018.

Karagiannakos, S. (2019). Predict bitcoin price with lstm. Retrieved from https://sergioskar.github.io/Bitcon_prediction_LSTM/

McNally, S., Roche, J., & Caton, S. (2018). Predicting the price of bitcoin using machine learning. In *2018 26th euromicro international conference on parallel, distributed and network-based processing (pdp)* (pp. 339–343). doi:10.1109/PDP2018.2018.00060

Schultze-Kraft, R. (2018, April). Don't be fooled—deceptive cryptocurrency price predictions using deep learning. Retrieved from https://hackernoon.com/dont-be-fooled-deceptive-cryptocurrency-price-predictions-using-deep-learning-bf27e4837151

Willmott, C. J., & Matsuura, K. (2006). On the use of dimensioned measures of error to evaluate the performance of spatial interpolators. *International Journal of Geographical Information Science*, *20*(1), 89–102. doi:10.1080/13658810500286976

Yalçın, O. G. (2018, October). Predict tomorrow's bitcoin (btc) price with recurrent neural networks. Retrieved from https://towardsdatascience.com/using-recurrent-neural-networks-to-predict-bitcoin-btc-prices-c4ff70f9f3e4