Machine Learning Project Report

Definition

Project Overview

Student provides a high-level overview of the project in layman's terms. Background information such as the problem domain, the project origin, and related data sets or input data is given.

The domain background for this project is the application of machine learning tools to predict prices for financial securities. For this project I took inspiration from a piece of code uploaded to machine learning platform Kaggle at https://www.kaggle.com/kimy07/eurusd-15-minute-interval-price-prediction/data and written by kimy07.

Our aim is to extend the analysis done in above code to a more granular (sub-second) dataset for EUR/USD with longer history (back to 2000). This should allow me to refine the model used by kimy07 and make better predictions.

From the literature on predicting financial security prices it appears the LSTM model (**L**ong **S**hort **T**erm **M**emory Network) can achieve better prediction capability compared to a simple Recurrent Neural Network and linear regression (http://colah.github.io/posts/2015-08-Understanding-LSTMs/) so we will use this approach. Other attempts at price prediction include bitcoin price prediction (http://trap.ncirl.ie/2496/1/seanmcnally.pdf), general evaluation of reinforcement learning algorithms on the foreign exchange market (https://www.doc.ic.ac.uk/teaching/distinguished-projects/2015/j.cumming.pdf), and stock market price prediction using LSTM specifically (http://ieeexplore.ieee.org/document/7966019/).

The dataset for this project is downloaded from http://www.histdata.com/download-free-forex-data/?/ascii/tick-data-quotes and consists of tick data - that is sub-second price quotes - for the EUR/USD currency pair since the beginning of 2000 up to the present day.

Problem Statement

The problem which needs to be solved is clearly defined. A strategy for solving the problem, including discussion of the expected solution, has been made.

The problem we will attempt to solve is price prediction based on historical tick data in the EUR/USD market.

We formulate this as a regression problem, as we try to predict future prices based on past information encoded in state variables, and those variables are continuous. An alternative formulation could be as a classification problem, but due to time constraints this was postponed.

The inputs for the problem are the millisecond date-time stamp, the bid and ask prices, as well as features derived from these such as open, close, high, low, bid-offer spread.

The strategy for solving this problem will involve the following steps: first the data will be cleaned so it is available for numerical processing by ensuring each value has a type. Next the original features will be extended to add more description of the 15 minute intervals to allow pattern finding. Then we will run a linear regression to get a baseline error for the more complex models to follow, such the LSTM model. The final prediction will be made with a LSTM model which will be given a certain lookback window (multiples of 15 minutes) for each sample. This should allow the model to improve upon the prediction of linear regression, which only looks at the previous 15 minute window.

Metrics

Metrics used to measure performance of a model or result are clearly defined. Metrics are justified based on the characteristics of the problem.

The success of any solution will be measured by looking at the error of predicted out-of-sample next prices.

Given that the problem is stated as a regression, we will use the mean absolute error and the mean squared error as evaluation metrics. I would hope to achieve a MAE of less than 0.003, which would be an improvement over the Kaggle benchmark model.

The reason for using that MSE in the optimisation algorithm is that it makes it easy to compute the gradient, and it punishes larger outliers more heavily due to squaring which is useful as larger price errors would lead to a larger financial loss. Also, it does not matter whether errors are positive or negative which is helpful as I do not need a bias towards under or over performance of the price prediction.

The reason for evaluating the model on MAE in the end is that it gives an idea how far the predicted prices diverges from the actual price in units of currency, which is of course a better value measure than currency squared. MSE is only a mathematical construct to help the optimiser to work faster and focus the optimisation on the larger errors. (https://www.quora.com/What-is-the-difference-between-squared-error-and-absolute-error)

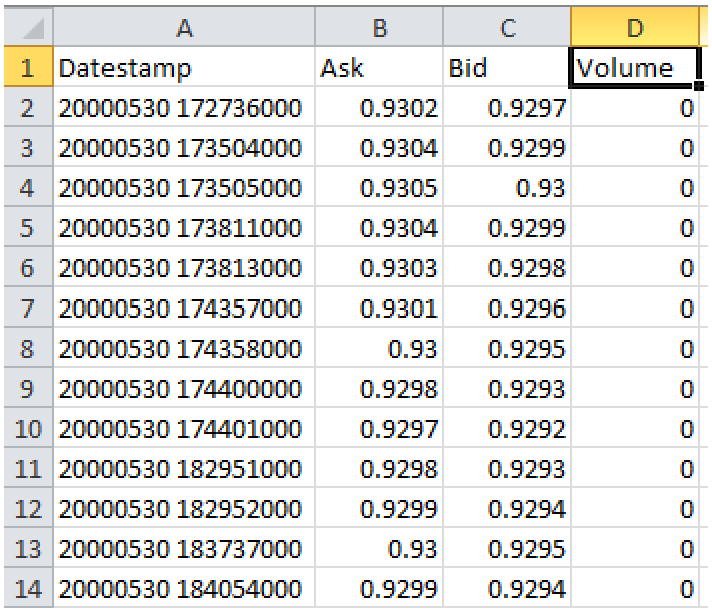
Using an R2 score would not be appropriate for optimisation here as it is not an error measure, it is an accuracy measure. A high R2 score corresponds to a low MSE. On the other hand, MSE is not normalised, so it will inflate with higher absolute deviations in the dataset, whereas R2 has a maximum value of 1 (perfect fit). Still, given that price differences are very small in our dataset we do not need to worry about this. (https://en.wikipedia.org/wiki/Coefficient_of_determination)

Analysis

Data Exploration

If a dataset is present, features and calculated statistics relevant to the problem have been reported and discussed, along with a sampling of the data. In lieu of a dataset, a thorough description of the *input space or input data has been made. Abnormalities or characteristics about the data or input that need to be addressed have been identified.*

The dataset is sourced from http://www.histdata.com/download-free-forex-data/?/ascii/tick-data-quotes. It contains for the EUR/USD currency pair tick by tick bid and ask price data for every day since 2000. Here a sample:



	A	B	C	D
1	Datestamp	Ask	Bid	Volume
2	20000530 172736000	0.9302	0.9297	0
3	20000530 173504000	0.9304	0.9299	0
4	20000530 173505000	0.9305	0.93	0
5	20000530 173811000	0.9304	0.9299	0
6	20000530 173813000	0.9303	0.9298	0
7	20000530 174357000	0.9301	0.9296	0
8	20000530 174358000	0.93	0.9295	0
9	20000530 174400000	0.9298	0.9293	0
10	20000530 174401000	0.9297	0.9292	0
11	20000530 182951000	0.9298	0.9293	0
12	20000530 182952000	0.9299	0.9294	0
13	20000530 183737000	0.93	0.9295	0
14	20000530 184054000	0.9299	0.9294	0

I will use date, bid and ask prices for this problem. Date is a millisecond datetime stamp, bid and ask are floating point numbers. Thus all three can be said to be continuous. The dataset has around 126 million rows. The outcome variable is the future price in the next period. Given I have many examples, I will split the data 99% training and 1% test, where validation is 10% of the training set.

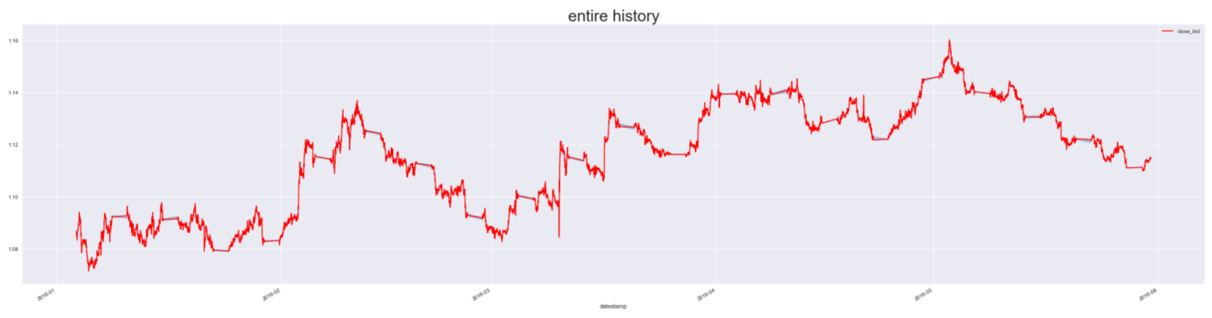
To prevent the algorithm from using future information to predict previous prices, I will attach actual future prices as target labels to each historical window I use for training the model. I will experiment with different window ranges as training sets, each being a multiple of 15 minutes to allow comparison to the Benchmark model.

Regarding class balances, there should not be any price groups in this dataset – it is unlikely that there is a "preferred price" at which the currency pair trades. To stationarize the data, i.e. to give no particular preference to absolute price levels, we could predict returns instead of absolute price values. This also leads to balanced classes, as both positive and negative return between ticks are approximately equally likely. I will not attempt this due to time constraints, and instead focus on price prediction only.

Exploratory Visualization

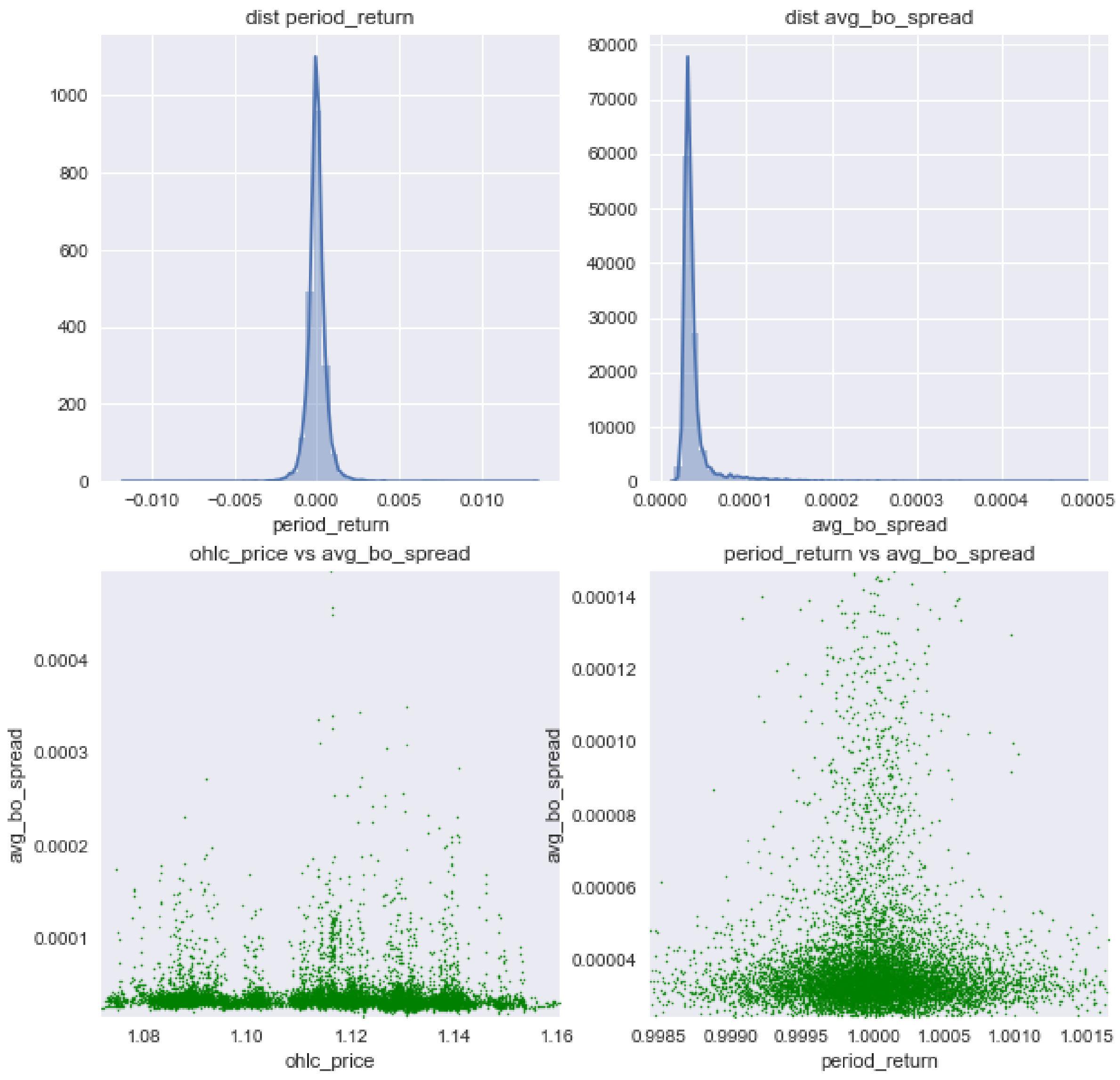
A visualization has been provided that summarizes or extracts a relevant characteristic or feature about the dataset or input data with thorough discussion. Visual cues are clearly defined.

Below the performance history of eurusd:



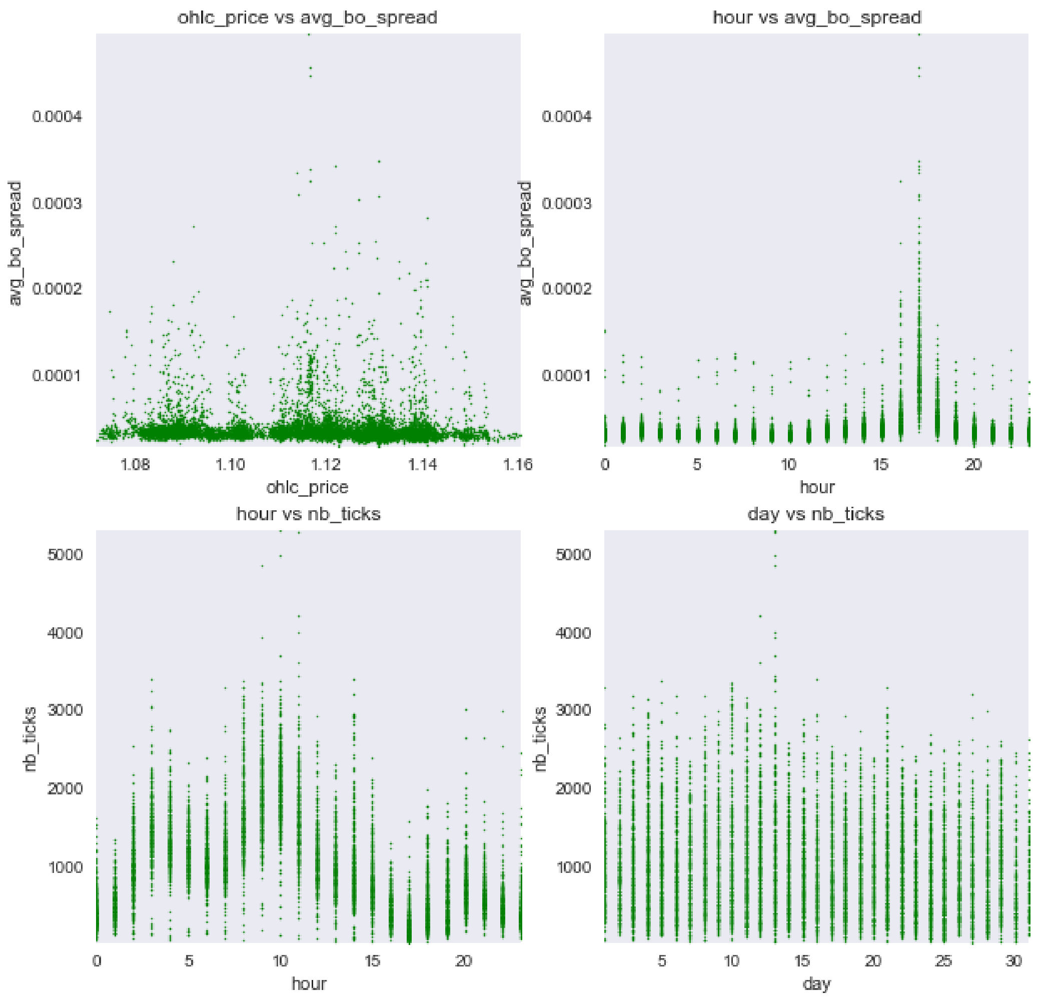
Y axis is price, x axis is datetime. There are periods where the price doesn't move (weekends), this should be accounted for when building the model.

After building features, we find the following distributions:

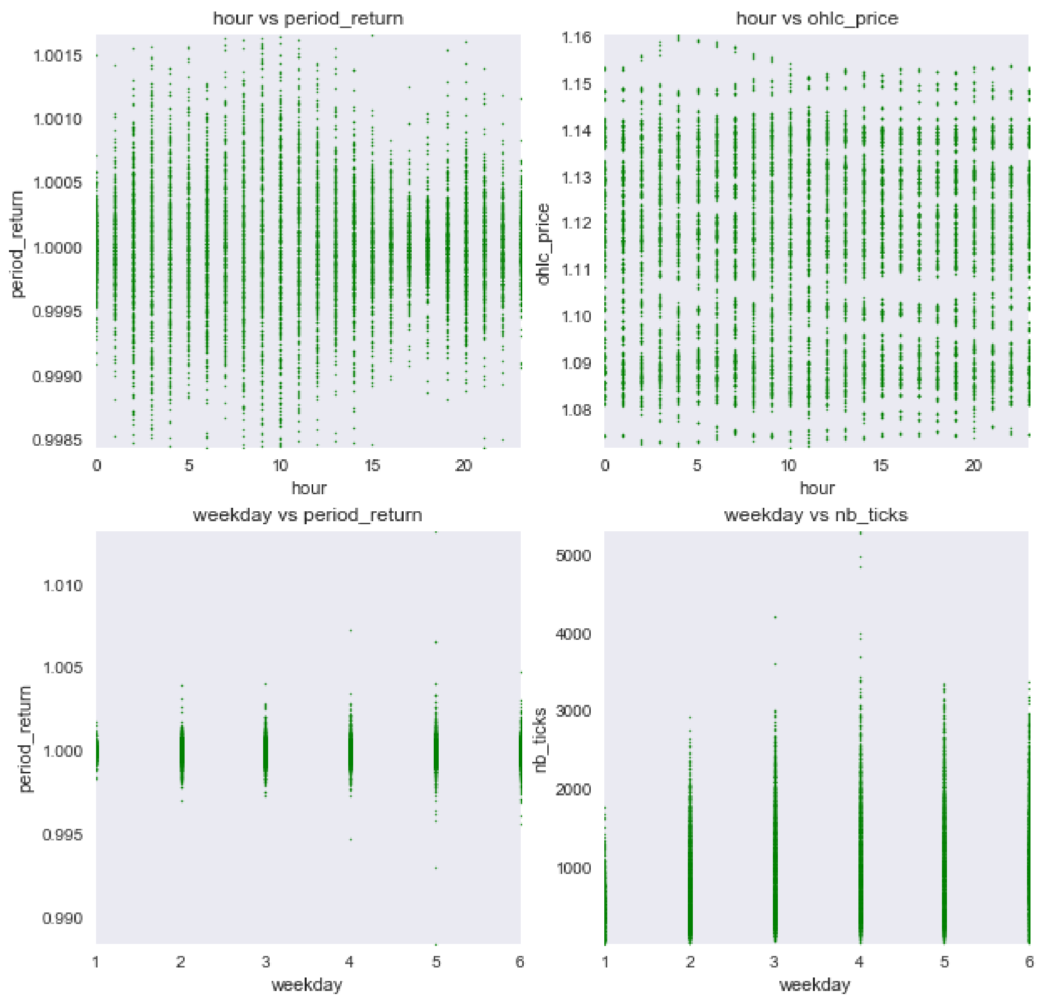


Returns seem evenly distributed, bid offer spreads are usually around 0.00005, prices are evenly distributed across the spectrum, returns around 0 seem to have the highest bid offer spreads. This makes sense as returns of 0 occur over the weekend where spreads would be wide given traders protect themselves against unforeseen market changes.

Next, we will check seasonality. We can see there is seasonality in the dataset, as the average bo spread changes with the hour, it seems highest and most spread out around 5 pm. The number of prices also varies with time, and to a smaller degree with the day of month. It seems around 5pm the least number of prices are observed, as well as around midnight.

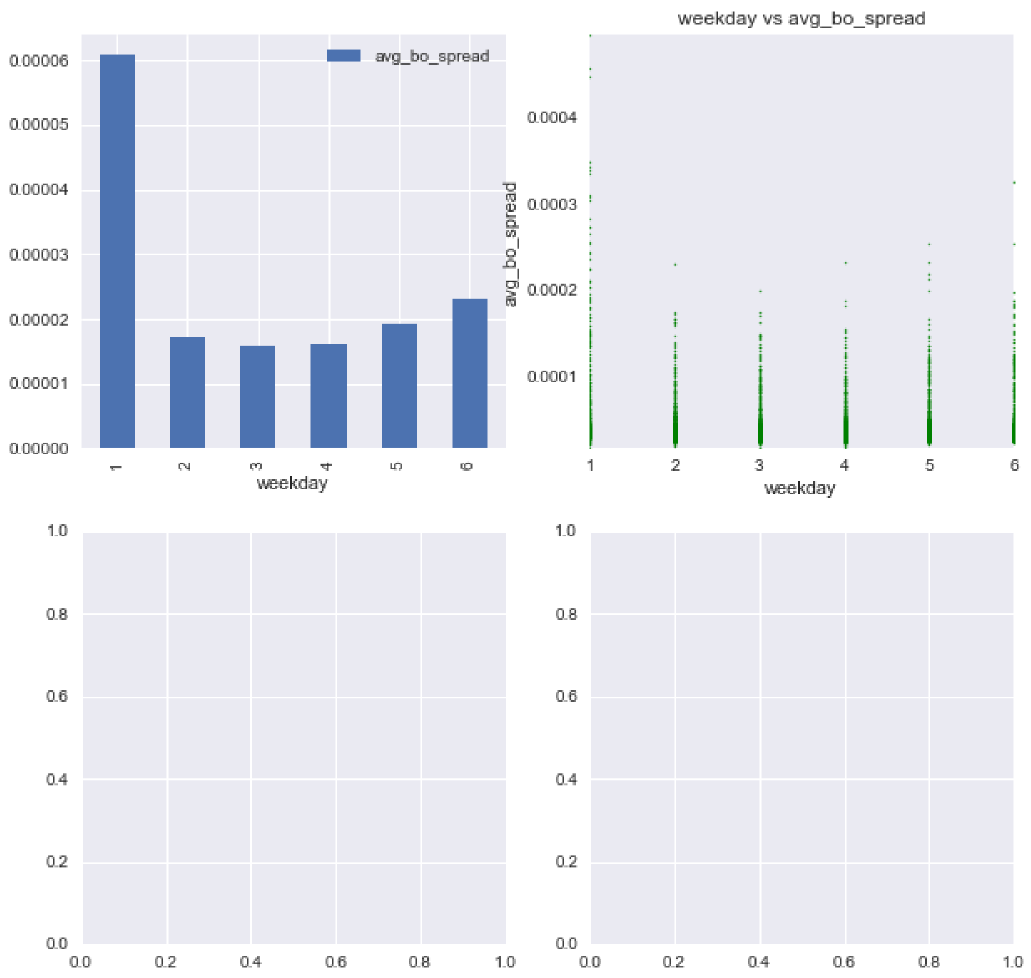


Further seasonality can be seen in the period return data and tick data on an hourly and weekday basis:



We can see here that weekday 4 (Thursday) has the largest variance in number of ticks, weekday 1 (Monday) clearly has a low absolute number of ticks. The period returns seems more narrow in the early hours and around 5pm, whereas there are no price patterns by the hour. This indicates that running return prediction might make for a more accurate model.

Finally, average bo spread is generally higher on Mondays:



As a result of above, we have incorporated seasonality (hour, day, etc) as features so that any model could use them to improve prediction accuracy.

Algorithms and Techniques

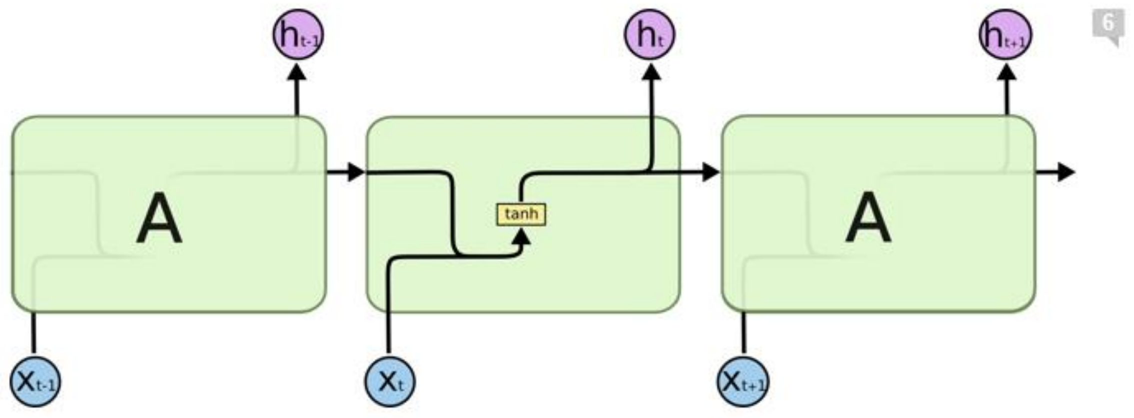
Algorithms and techniques used in the project are thoroughly discussed and properly justified based on the characteristics of the problem.

Algorithms we tried are linear regression given the continuous variables in the problem, and in order to get a first idea of how accurate a simple algorithm would be to discover any easy predictors. Linear regression will just fit a straight line through a cloud of multidimensional points and try to achieve a best fit using a linear function.

Next, PCA was used for dimensionality reduction and feature engineering as a preprocessing step for the subsequent neural network (LSTM). The idea was to remove correlated features that may have impacted linear regression. PCA reduces the dimensionality of a dataset by projecting for example 2 dimensional coordinates onto a 1 dimensional line. This line then represents a new feature, combining 2 other features. The meaning of this is often unclear, as it is a combination of features. However, you can show the loadings of each PCA components, to get an idea how much each original feature is weighted to get to a particular PCA component. PCA tries to minimize the squared projection error of x onto the line or more generally onto the hyperplane.

We also tried a random forest regressor to help extract feature importances to see if there is an easy way to predict returns or prices. It seems the best feature was the previous close price, so direct historical dependency which indicated no pattern. A random forest works by trying out many individual decision trees. The Decision Tree algo works using Information gain: the entropy reduction in the datasets due to splitting. You define some labels, say good and bad datapoints, and then you check how their entropy is reduced by splitting the dataset on various features, like size, age etc. Entropy reduction is defined as parent dataset entropy minus weighted child dataset entropy after splitting on a feature. Like this, if the split does not make the dataset less entropic, you would not choose this feature.

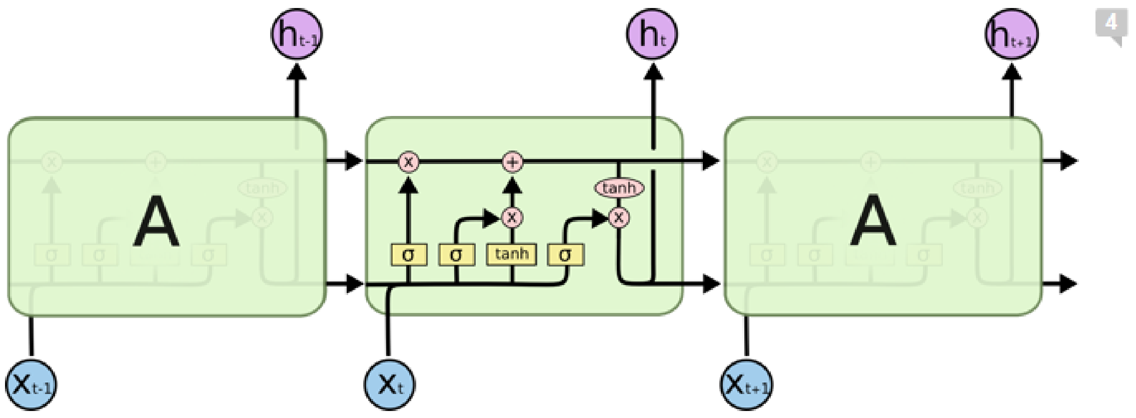
An LSTM network was used to make the final prediction as it could look at a timeseries of features and differentiate between recent and less recent values in order to find patterns. The LSTM network has the benefit that it can take previous states into account when coming to a decision, not just the immediate state. Thus it models very well the sequential evolution of prices in a timeseries, which a traditional NN could not do. This also requires the test set to be at the end of the timeseries, as else future events would be used to model past events. Thus RNNs add loops to allow each step of the network to give information to the next.



The repeating module in a standard RNN contains a single layer.

Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

However, this works best if the context required for a correct prediction is relatively close in terms of steps. For information further in the past, it does not work so well. The LSTM is a special kind of RNN: it has a cell state, which acts like a conveyor belt to let information flow through a neural network unchanged. In addition, it has a number of gates, which control how the information on this conveyor belt is changed, if necessary. These gates decide what to remove and what to add to the conveyor based on both current and previous state. The result is then output and used by the next node.



The repeating module in an LSTM contains four interacting layers.

Source for this section and all visualisations is: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Benchmark

Student clearly defines a benchmark result or threshold for comparing performances of solutions obtained.

As benchmark the Kaggle code created by kimy, a contributor on machine learning platform Kaggle, will be used: https://www.kaggle.com/kimy07/eurusd-15-minute-interval-price-prediction/notebook. The benchmark model attempted price prediction with bid price data on EUR/USD with a fairly coarse 15 minute observation window. We use the same interval using our own dataset to allow comparison of our results and get an idea if adding more features can improve the prediction accuracy.

The main model used in above scripts was a LSTM model trained on only a few features (specifically no ask prices in their dataset) to make predictions.

Methodology

Data Preprocessing

All preprocessing steps have been clearly documented. Abnormalities or characteristics about the data or input that needed to be addressed have been corrected. If no data preprocessing is necessary, it has been clearly justified.

To attempt a solution, the dataset is loaded into a sql database. Next data cleanliness is checked, to account for nulls, positive and negative outliers and to create a final, clean dataset.

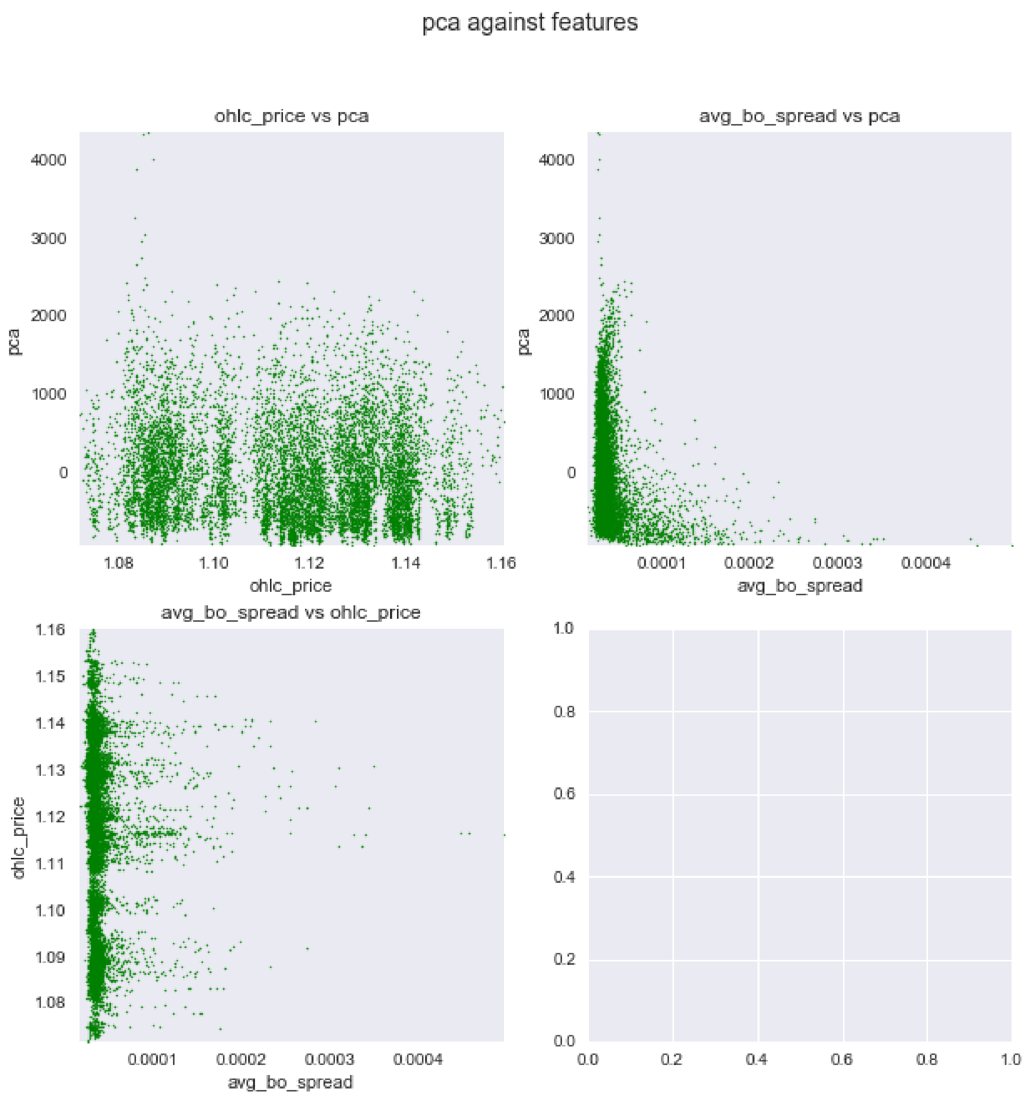
The dataset is enriched by creating additional features for my state space, such as:

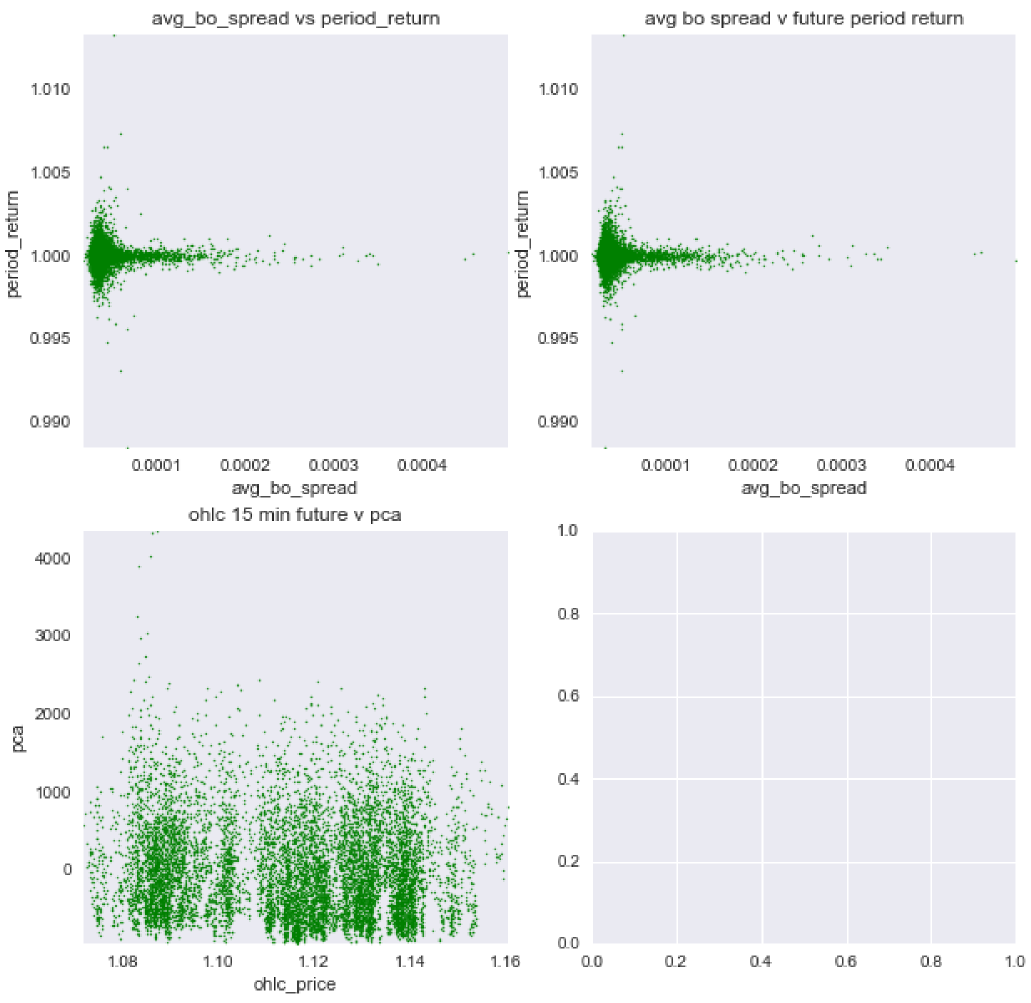
- returns
- bid-offer spreads
- period high, low, average

The tricky bit here is that although the prediction from the previous tick row is straightforward, as each feature can be used directly, using several tick rows to cover a 15 minute lookback window requires a decision how to use the features of each row. To address this, we will group the data into 15 minute intervals, with a number of aggregates to describe price behaviour during this time, such as:

- average
- period return
- open-high-low-close price (ohlc price)
- number of prices (called 'ticks') during period

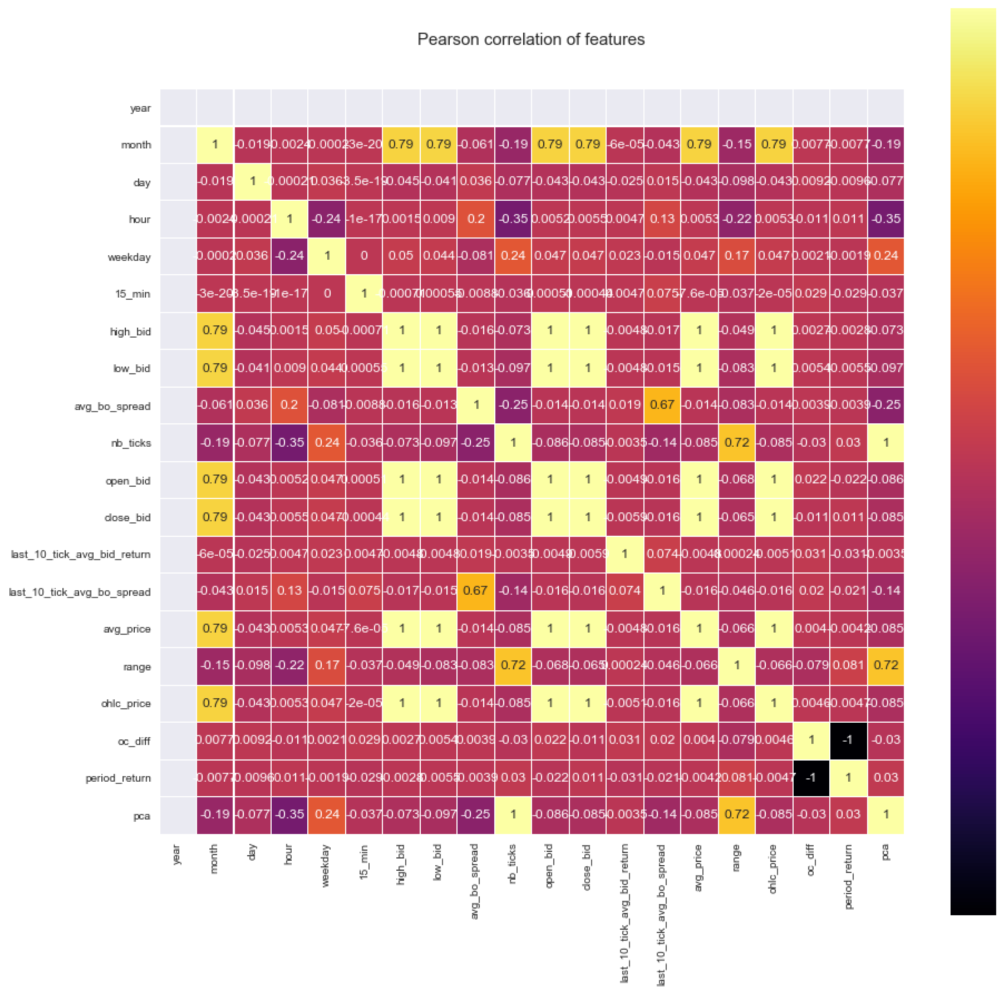
In addition, due to feature correlation (ohlc price vs close price), PCA is used to extract the first dimension as a feature:

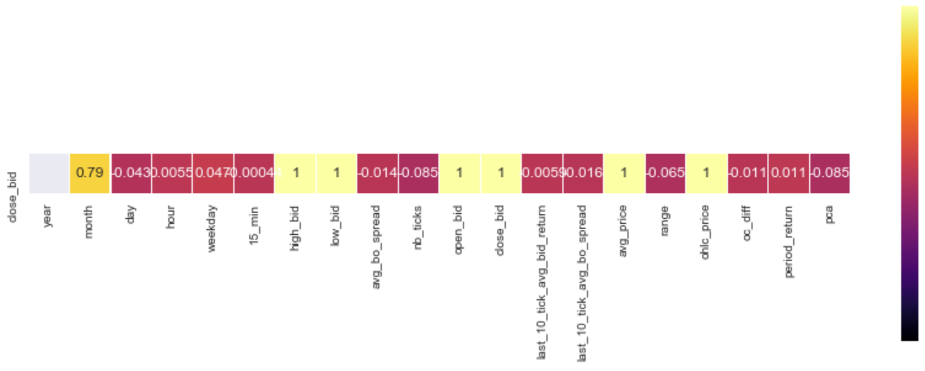

pca against features
ohlc_price vs pca
avg_bo_spread vs pca
avg_bo_spread vs ohlc_price
pca
ohlc_price
avg_bo_spread
4000
3000
2000
1000
0
1.08
1.10
1.12
1.14
1.16
0.0001
0.0002
0.0003
0.0004
1.15
1.13
1.11
1.09
0.0
0.2
0.4
0.6
0.8
1.0



The results suggest that PCA does not add extra structure to the dataset. We also checked whether next period returns are correlated to any features but there seems no particular pattern.

Next we tried explicitly correlating features (year is greyed out as only two values are available):





It can be seen that the best correlation to close is close itself, or else one of the other price composites for a 15 minute window. This is a trivial result, as the close price enters into those aggregates. Other observations are that the time and the number of ticks are correlated, which corroborates our initial findings that time provides structure to the data. Further, prices seem to have correlation with months, but given the short timeframe of data we use this will not be useful. We would have to look at several years to ensure month-price correlations are a consistent occurrence.

<u>Random Forest:</u>

A random forest regression suggests that the best feature to divide the dataset is the close price itself, by far:

```
Feature ranking:

close_bid                     0.926392
ohlc_price                    0.024249
avg_price                     0.022592
high_bid                      0.022400
low_bid                       0.003085
open_bid                      0.000675
last_10_tick_avg_bo_spread    0.000088
last_10_tick_avg_bid_return   0.000085
range                         0.000064
avg_bo_spread                 0.000060
pca                           0.000056
nb_ticks                      0.000053
oc_diff                       0.000041
period_return                 0.000039
hour                          0.000037
day                           0.000037
weekday                       0.000021
15_min                        0.000017
month                         0.000009
```

<u>Linear Regression:</u>

Running linear regression to predict close price, this is the result:

```
oc_diff                        -0.204211
nb_ticks                       -0.096250
low_bid                        -0.017280
last_10_tick_avg_bo_spread     -0.005564
avg_bo_spread                  -0.003196
avg_price                      -0.003091
last_10_tick_avg_bid_return    -0.002060
weekday                        -0.000060
year                            0.000000
month                           0.000062
15_min                          0.000098
day                             0.000105
hour                            0.000380
high_bid                        0.012807
range                           0.031856
pca                             0.096251
period_return                   0.185722
```

```
ohlc_price                      0.199248
open_bid                        0.300989
close_bid                       0.505757
```

The main factors are the open close difference, the close bid, the open bid and the period return. PCA and the number of ticks get about 10% each. The other factors are marginal.

Finally, the choice of test set was a challenge. It had to be at the end of the timeseries, as else I would be using future information to predict past prices. My test set corresponds to the last 5 days of May 2016, some of which are a weekend so no price movement takes place. The test set contains about 1 percent of rows, but as the kaggle dataset and mine have slightly different lengths, I constrained the test set to contain exactly 149 rows, or 1% of the kaggle dataset to allow better comparison.

Implementation

The process for which metrics, algorithms, and techniques were implemented with the given datasets or input data has been thoroughly documented. Complications that occurred during the coding process are discussed.

The main coding difficulty was how to keep track of a long jupyter notebook. For that we added a table of contents plugin. To plot many different correlations, we wrote functions that would plot into a matplotlib subplot. Most algorithms were already available in various python libraries and have good documentation, so visualisation was the most difficult part. In addition, we had to save all intermediate steps, for example model weights, configuration, so that they could be reloaded in case I needed to check them again. We wrote a number of helper functions to do this, and organised the output file into a folder structure with names indicating the model version they belong to.

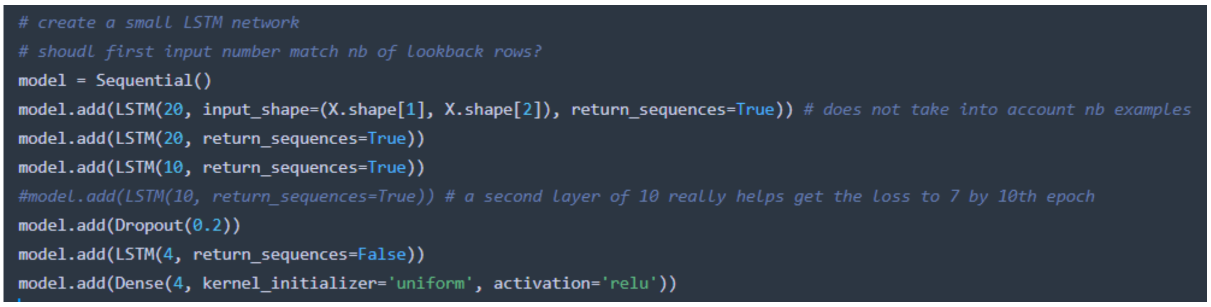
To code up the initial models, we used the documentation from scikit learn and Keras to get us started. We also took inspiration from the code on Kaggle mentioned above to get familiar with the model syntax. We then proceeded to document what some of the less obvious lines of code do, and came up with some ideas how to refine the model.

In addition, we thought to try some simpler steps to check if a model such as LSTM is even necessary and thus added Linear regression, Ridge Regression (which did not help, so results are not included), and add more visualisations and features given our dataset contains ask prices as well.

A complexity was found when trying to code custom error metrics for the LSTM model – it seemed Keras often has issues with this, for example when using custom tensorflow functions for error metrics, and debugging is not straight forward as it is hard to visualise the state of all variables in a keras model. We thus decided it is better to use the built-in error metrics and not spend time debugging the Keras sourcecode.

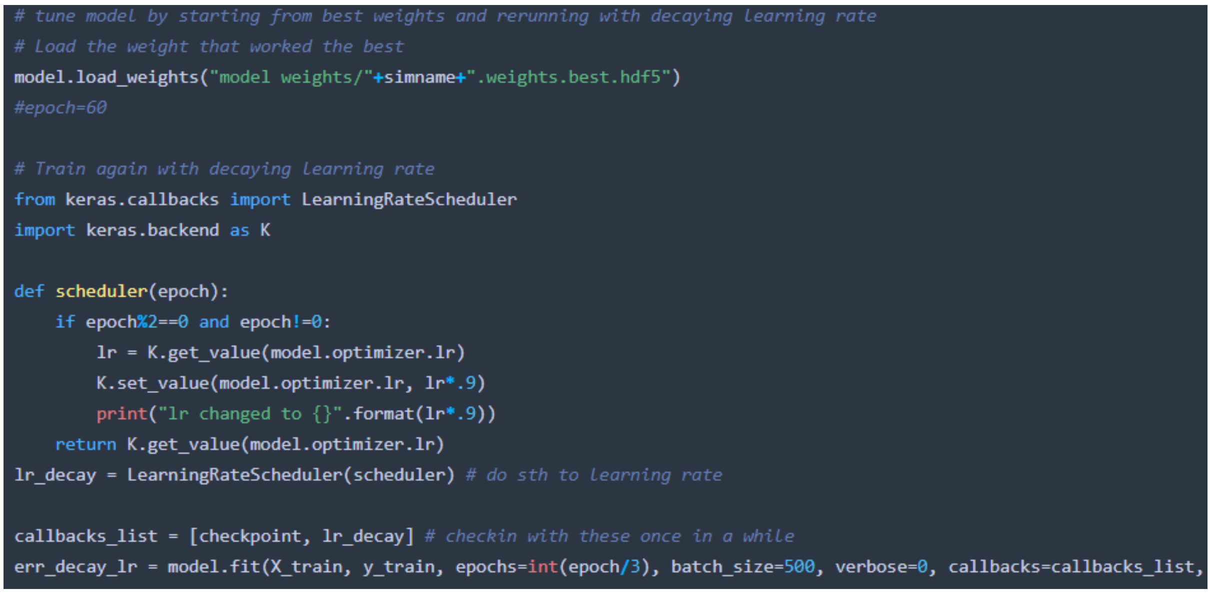
Further, the batch size seemed to have a large effect on model convergence, and the literature was not very clear on what would be an appropriate batch size, so we had to search by trial and error.

The benchmark LSTM model uses the following configuration:



```
# create a small LSTM network
# shoudl first input number match nb of lookback rows?
model = Sequential()
model.add(LSTM(20, input_shape=(X.shape[1], X.shape[2]), return_sequences=True)) # does not take into account nb examples
model.add(LSTM(20, return_sequences=True))
model.add(LSTM(10, return_sequences=True))
#model.add(LSTM(10, return_sequences=True)) # a second layer of 10 really helps get the loss to 7 by 10th epoch
model.add(Dropout(0.2))
model.add(LSTM(4, return_sequences=False))
model.add(Dense(4, kernel_initializer='uniform', activation='relu'))
```

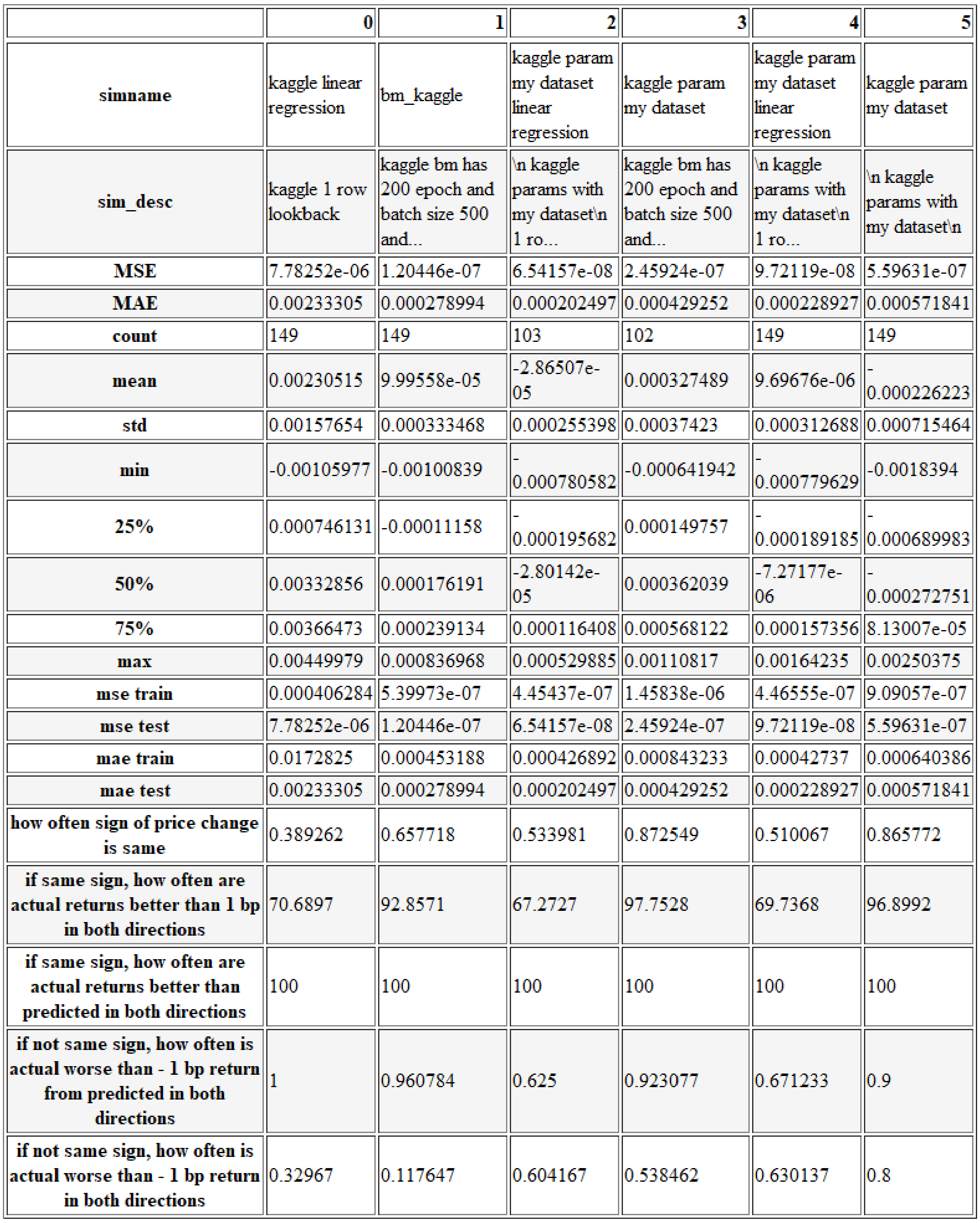
It is then rerun with a decaying learning rate to see if improvements can be made by more self-reliance of the model, instead of data reliance (there seemed to be no obvious benefit in most simulations so results are not shown):



```
# tune model by starting from best weights and rerunning with decaying learning rate
# Load the weight that worked the best
model.load_weights("model weights/"+simname+".weights.best.hdf5")
#epoch=60

# Train again with decaying learning rate
from keras.callbacks import LearningRateScheduler
import keras.backend as K

def scheduler(epoch):
    if epoch%2==0 and epoch!=0:
        lr = K.get_value(model.optimizer.lr)
        K.set_value(model.optimizer.lr, lr*.9)
        print("lr changed to {}".format(lr*.9))
    return K.get_value(model.optimizer.lr)
lr_decay = LearningRateScheduler(scheduler) # do sth to learning rate

callbacks_list = [checkpoint, lr_decay] # checkin with these once in a while
err_decay_lr = model.fit(X_train, y_train, epochs=int(epoch/3), batch_size=500, verbose=0, callbacks=callbacks_list,
```

The benchmark model did not have much visualisation of the input features, so it was not clear if it was missing anything. This was added in the preprocessing step. In addition, more error metrics (such as directional error and error beyond a certain amount of financial loss) had to be created to give a better view on how good the model was.

Initial results were this:



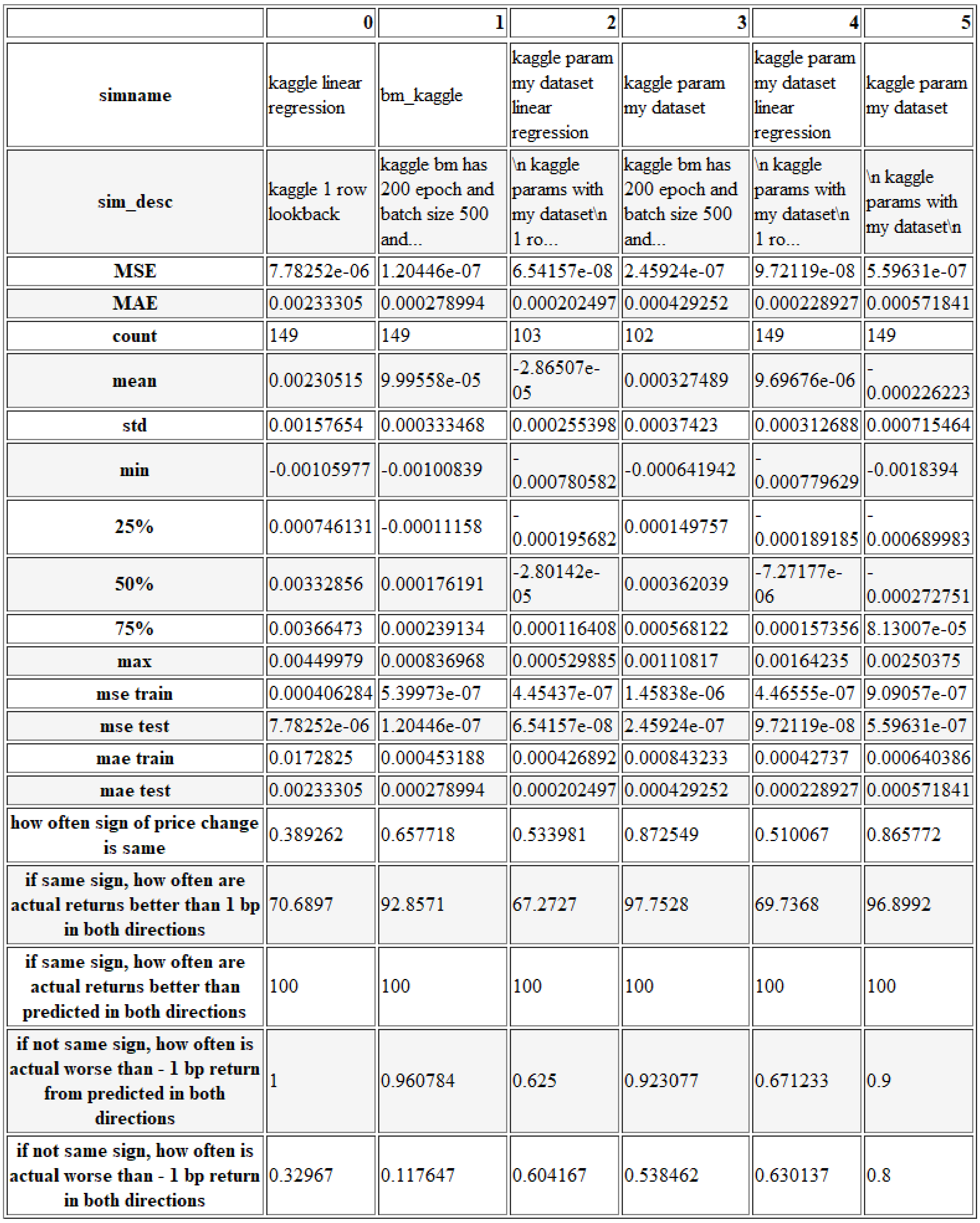
	0	1	2	3	4	5
simname	kaggle linear regression	bm_kaggle	kaggle param my dataset linear regression	kaggle param my dataset	kaggle param my dataset linear regression	kaggle param my dataset
sim_desc	kaggle 1 row lookback	kaggle bm has 200 epoch and batch size 500 and...	\n kaggle params with my dataset\n 1 ro...	kaggle bm has 200 epoch and batch size 500 and...	\n kaggle params with my dataset\n 1 ro...	\n kaggle params with my dataset\n
MSE	7.78252e-06	1.20446e-07	6.54157e-08	2.45924e-07	9.72119e-08	5.59631e-07
MAE	0.00233305	0.000278994	0.000202497	0.000429252	0.000228927	0.000571841
count	149	149	103	102	149	149
mean	0.00230515	9.99558e-05	-2.86507e-05	0.000327489	9.69676e-06	-0.000226223
std	0.00157654	0.000333468	0.000255398	0.00037423	0.000312688	0.000715464
min	-0.00105977	-0.00100839	-0.000780582	-0.000641942	-0.000779629	-0.0018394
25%	0.000746131	-0.00011158	-0.000195682	0.000149757	-0.000189185	-0.000689983
50%	0.00332856	0.000176191	-2.80142e-05	0.000362039	-7.27177e-06	-0.000272751
75%	0.00366473	0.000239134	0.000116408	0.000568122	0.000157356	8.13007e-05
max	0.00449979	0.000836968	0.000529885	0.00110817	0.00164235	0.00250375
mse train	0.000406284	5.39973e-07	4.45437e-07	1.45838e-06	4.46555e-07	9.09057e-07
mse test	7.78252e-06	1.20446e-07	6.54157e-08	2.45924e-07	9.72119e-08	5.59631e-07
mae train	0.0172825	0.000453188	0.000426892	0.000843233	0.00042737	0.000640386
mae test	0.00233305	0.000278994	0.000202497	0.000429252	0.000228927	0.000571841
how often sign of price change is same	0.389262	0.657718	0.533981	0.872549	0.510067	0.865772
if same sign, how often are actual returns better than 1 bp in both directions	70.6897	92.8571	67.2727	97.7528	69.7368	96.8992
if same sign, how often are actual returns better than predicted in both directions	100	100	100	100	100	100
if not same sign, how often is actual worse than - 1 bp return from predicted in both directions	1	0.960784	0.625	0.923077	0.671233	0.9
if not same sign, how often is actual worse than - 1 bp return in both directions	0.32967	0.117647	0.604167	0.538462	0.630137	0.8

It can be seen that MAE did not improve, however the directional error (how often sign of price change is the same between predicted and actual) did, and gave some source of optimism. More detail on this below.

Refinement

The process of improving upon the algorithms and techniques used is clearly documented. Both the initial and final solutions are reported, along with intermediate solutions, if necessary.

Here a comparison of the errors of the different simulations. We chose MAE and MSE as errors, given we are trying to hit the exact price values. In addition, we calculate directional error metrics, to get an idea how often my prediction has the correct direction, and how often a misprediction results in a loss of more than 1 basis point (0.01 percent) of return to get an idea how often costly losses are caused by misprediction.



	0	1	2	3	4	5
simname	kaggle linear regression	bm_kaggle	kaggle param my dataset linear regression	kaggle param my dataset	kaggle param my dataset linear regression	kaggle param my dataset
sim_desc	kaggle 1 row lookback	kaggle bm has 200 epoch and batch size 500 and...	\n kaggle params with my dataset\n 1 ro...	kaggle bm has 200 epoch and batch size 500 and...	\n kaggle params with my dataset\n 1 ro...	\n kaggle params with my dataset\n
MSE	7.78252e-06	1.20446e-07	6.54157e-08	2.45924e-07	9.72119e-08	5.59631e-07
MAE	0.00233305	0.000278994	0.000202497	0.000429252	0.000228927	0.000571841
count	149	149	103	102	149	149
mean	0.00230515	9.99558e-05	-2.86507e-05	0.000327489	9.69676e-06	-0.000226223
std	0.00157654	0.000333468	0.000255398	0.00037423	0.000312688	0.000715464
min	-0.00105977	-0.00100839	-0.000780582	-0.000641942	-0.000779629	-0.0018394
25%	0.000746131	-0.00011158	-0.000195682	0.000149757	-0.000189185	-0.000689983
50%	0.00332856	0.000176191	-2.80142e-05	0.000362039	-7.27177e-06	-0.000272751
75%	0.00366473	0.000239134	0.000116408	0.000568122	0.000157356	8.13007e-05
max	0.00449979	0.000836968	0.000529885	0.00110817	0.00164235	0.00250375
mse train	0.000406284	5.39973e-07	4.45437e-07	1.45838e-06	4.46555e-07	9.09057e-07
mse test	7.78252e-06	1.20446e-07	6.54157e-08	2.45924e-07	9.72119e-08	5.59631e-07
mae train	0.0172825	0.000453188	0.000426892	0.000843233	0.00042737	0.000640386
mae test	0.00233305	0.000278994	0.000202497	0.000429252	0.000228927	0.000571841
how often sign of price change is same	0.389262	0.657718	0.533981	0.872549	0.510067	0.865772
if same sign, how often are actual returns better than 1 bp in both directions	70.6897	92.8571	67.2727	97.7528	69.7368	96.8992
if same sign, how often are actual returns better than predicted in both directions	100	100	100	100	100	100
if not same sign, how often is actual worse than - 1 bp return from predicted in both directions	1	0.960784	0.625	0.923077	0.671233	0.9
if not same sign, how often is actual worse than - 1 bp return in both directions	0.32967	0.117647	0.604167	0.538462	0.630137	0.8

<u>Kaggle Benchmark Model:</u>

The kaggle benchmark LSTM model has a MAE of 0.00028 in close price prediction on the test set, and a directional accuracy of price changes of around 66 percent. If the direction matches, actual returns are always better than the returns implied by the predicted price. If direction does not match, all returns are always worse than predicted returns. So generally it seems the benchmark is biased to predict too small returns. The Kaggle linear regression benchmark shows MAE of 0.0023 and directional accuracy of around 40%, so considerably worse than the LSTM benchmark.

Rerunning the LSTM benchmark model with my dataset, and my additional features, grouped into 15 minute intervals, my MAE worsens to 0.00057, but directional accuracy improves to 86 percent. For linear regression, my MAE improves to 0.00021, and directional accuracy improves to 51 percent.

Rerunning the LSTM benchmark model with my dataset, I constrain the test set to be exactly 149 rows, just like the benchmark, to allow comparison as each observation would account for more than 1 percent of the accuracy.

It is possible given that my dataset has more features, it would probably require more training to converge and this might explain higher errors.

Parameter tuning LSTM model:

Next we tune the model parameters for the LSTM model, by changing batch size, number of epochs, lookback period, and the way data flows through the LSTM model. Here the results:

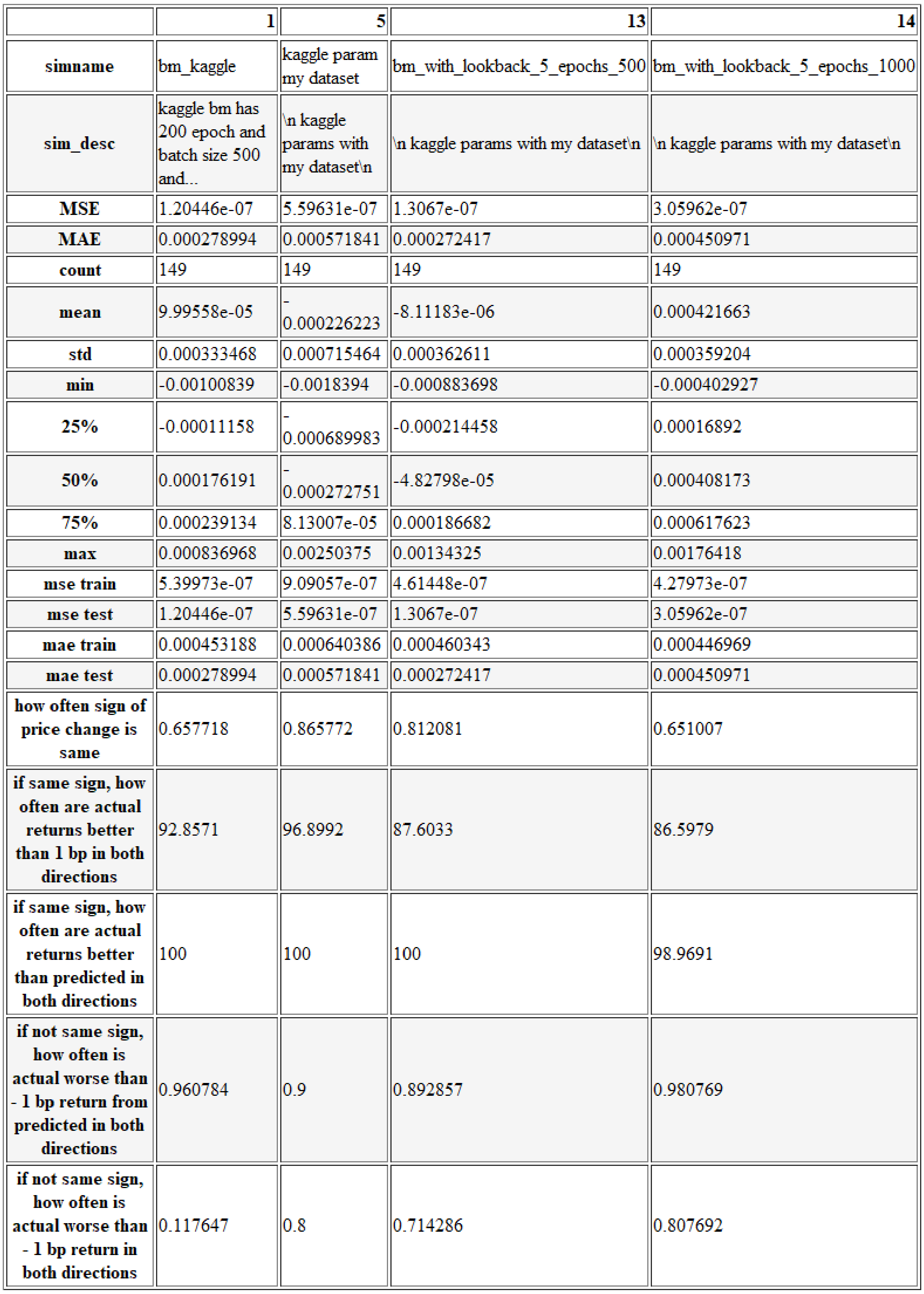
	1	5	6	7	8	9	10
simname	bm_kaggle	kaggle param my dataset	bm_with_lookback_10	bm_with_lookback_2	bm_with_lookback_30	bm_with_lookback_60	bm_with_lookback_5
sim_desc	kaggle bm has 200 epoch and batch size 500 and...	\n kaggle params with my dataset\n	\n kaggle params with my dataset\n	\n kaggle params with my dataset\n	\n kaggle params with my dataset\n	\n kaggle params with my dataset\n	\n kaggle params with my dataset\n
MSE	1.20446e-07	5.59631e-07	2.63716e-07	5.11102e-07	3.56223e-07	3.40443e-07	1.9192e-07
MAE	0.000278994	0.000571841	0.000391006	0.000586614	0.000425628	0.000417161	0.000348615
count	149	149	149	149	149	149	149
mean	9.99558e-05	-0.000226223	0.000291342	0.0004673	0.000211435	4.8595e-05	-0.000185057
std	0.000333468	0.000715464	0.000424316	0.000542872	0.00056002	0.000583409	0.000398422
min	-0.00100839	-0.0018394	-0.000664353	-0.00124609	-0.00115061	-0.00137413	-0.00126648
25%	-0.00011158	-0.000689983	2.43187e-05	0.000217319	-0.0001086	-0.000298381	-0.000445724
50%	0.000176191	-0.000272751	0.0002352	0.000464559	0.000171542	-4.79221e-05	-0.000210404
75%	0.000239134	8.13007e-05	0.000537634	0.000727296	0.000488997	0.000317574	1.64509e-05
max	0.000836968	0.00250375	0.00186908	0.00251567	0.00285125	0.00272167	0.0014925
mse train	5.39973e-07	9.09057e-07	6.15854e-07	7.92264e-07	1.37217e-06	9.96409e-07	7.81467e-07
mse test	1.20446e-07	5.59631e-07	2.63716e-07	5.11102e-07	3.56223e-07	3.40443e-07	1.9192e-07
mae train	0.000453188	0.000640386	0.000521012	0.000598451	0.000818601	0.000662895	0.000605548
mae test	0.000278994	0.000571841	0.000391006	0.000586614	0.000425628	0.000417161	0.000348615
how often sign of price change is same	0.657718	0.865772	0.838926	0.731544	0.919463	0.993289	0.744966
if same sign, how often are actual returns better than 1 bp in both directions	92.8571	96.8992	90.4	91.7431	97.0803	100	88.2883
if same sign, how often are actual returns better than predicted in both directions	100	100	100	100	100	100	100
if not same sign, how often is actual worse than - 1 bp return from predicted in both directions	0.960784	0.9	1	1	1	1	0.973684
if not same sign, how often is actual worse than - 1 bp return in both directions	0.117647	0.8	0.541667	0.65	0.75	1	0.736842

Reducing the lookback period from 20 intervals of 15 minutes to 10 yields and improvement in MAE to 0.00039, but decreases directional accuracy to 0.84. Running other lookback values shows that a longer lookback value gives better directional error whereas a shorter lookback gives better MAE.

Changing batch size from 500 makes MAE worse in both directions, but increasing epochs helps reduce MAE:

	1	5	11	12	13
simname	bm_kaggle	kaggle param my dataset	bm_with_lookback_5_batch_100	bm_with_lookback_5_batch_1000	bm_with_lookback_5_epochs_500
sim_desc	kaggle bm has 200 epoch and batch size 500 and...	\n kaggle params with my dataset\n	\n kaggle params with my dataset\n	\n kaggle params with my dataset\n	\n kaggle params with my dataset\n
MSE	1.20446e-07	5.59631e-07	2.50884e-07	4.70728e-07	1.3067e-07
MAE	0.000278994	0.000571841	0.000401295	0.000571145	0.000272417
count	149	149	149	149	149
mean	9.99558e-05	-0.000226223	0.000349894	-0.000443419	-8.11183e-06
std	0.000333468	0.000715464	0.000359619	0.000525319	0.000362611
min	-0.00100839	-0.0018394	-0.000430465	-0.00178719	-0.000883698
25%	-0.00011158	-0.000689983	0.000124812	-0.000771403	-0.000214458
50%	0.000176191	-0.000272751	0.000325561	-0.000485659	-4.82798e-05
75%	0.000239134	8.13007e-05	0.000590563	-0.000166774	0.000186682
max	0.000836968	0.00250375	0.00160778	0.00173998	0.00134325
mse train	5.39973e-07	9.09057e-07	4.57856e-07	1.27704e-06	4.61448e-07
mse test	1.20446e-07	5.59631e-07	2.50884e-07	4.70728e-07	1.3067e-07
mae train	0.000453188	0.000640386	0.000457225	0.000710906	0.000460343
mae test	0.000278994	0.000571841	0.000401295	0.000571145	0.000272417
how often sign of price change is same	0.657718	0.865772	0.66443	0.604027	0.812081
if same sign, how often are actual returns better than 1 bp in both directions	92.8571	96.8992	86.8687	88.8889	87.6033
if same sign, how often are actual returns better than predicted in both directions	100	100	100	100	100
if not same sign, how often is actual worse than - 1 bp return from predicted in both directions	0.960784	0.9	1	1	0.892857
if not same sign, how often is actual worse than - 1 bp return in both directions	0.117647	0.8	0.8	0.779661	0.714286

Increasing epochs too much increases MAE again, an indication that the model does not converge:



	1	5	13	14
simname	bm_kaggle	kaggle param my dataset	bm_with_lookback_5_epochs_500	bm_with_lookback_5_epochs_1000
sim_desc	kaggle bm has 200 epoch and batch size 500 and...	\n kaggle params with my dataset\n	\n kaggle params with my dataset\n	\n kaggle params with my dataset\n
MSE	1.20446e-07	5.59631e-07	1.3067e-07	3.05962e-07
MAE	0.000278994	0.000571841	0.000272417	0.000450971
count	149	149	149	149
mean	9.99558e-05	-0.000226223	-8.11183e-06	0.000421663
std	0.000333468	0.000715464	0.000362611	0.000359204
min	-0.00100839	-0.0018394	-0.000883698	-0.000402927
25%	-0.00011158	-0.000689983	-0.000214458	0.00016892
50%	0.000176191	-0.000272751	-4.82798e-05	0.000408173
75%	0.000239134	8.13007e-05	0.000186682	0.000617623
max	0.000836968	0.00250375	0.00134325	0.00176418
mse train	5.39973e-07	9.09057e-07	4.61448e-07	4.27973e-07
mse test	1.20446e-07	5.59631e-07	1.3067e-07	3.05962e-07
mae train	0.000453188	0.000640386	0.000460343	0.000446969
mae test	0.000278994	0.000571841	0.000272417	0.000450971
how often sign of price change is same	0.657718	0.865772	0.812081	0.651007
if same sign, how often are actual returns better than 1 bp in both directions	92.8571	96.8992	87.6033	86.5979
if same sign, how often are actual returns better than predicted in both directions	100	100	100	98.9691
if not same sign, how often is actual worse than - 1 bp return from predicted in both directions	0.960784	0.9	0.892857	0.980769
if not same sign, how often is actual worse than - 1 bp return in both directions	0.117647	0.8	0.714286	0.807692

After review, it emerged that a faster way to compute all these variants would have been to use the GridSearchCV class, which we will incorporate in a future version of this project to test more variants.

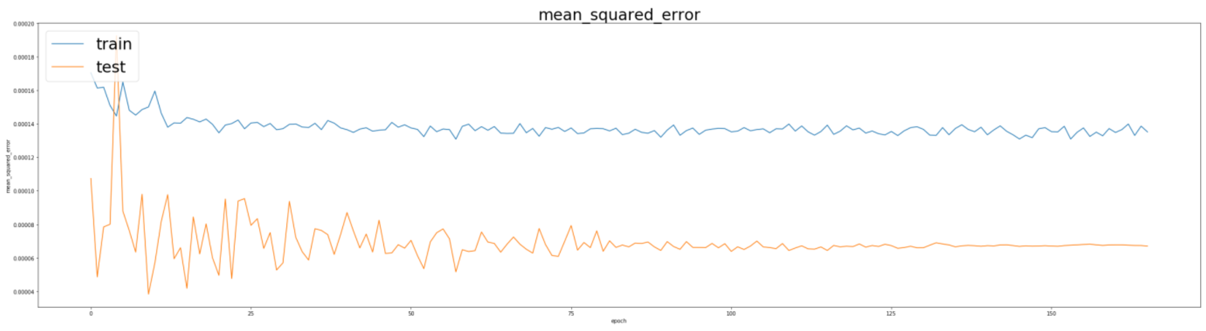
Results

Model Evaluation and Validation

The final model's qualities — such as parameters — are evaluated in detail. Some type of analysis is used to validate the robustness of the model's solution.

The final Model is an LSTM Model with a lookback of 5 periods, running for 500 epochs in batch sizes of 500. The robustness is confirmed by above attempts to search the parameters space in the vicinity of the final solution and obtaining worse MAE.

To validate robustness, one can look at the train and test error:



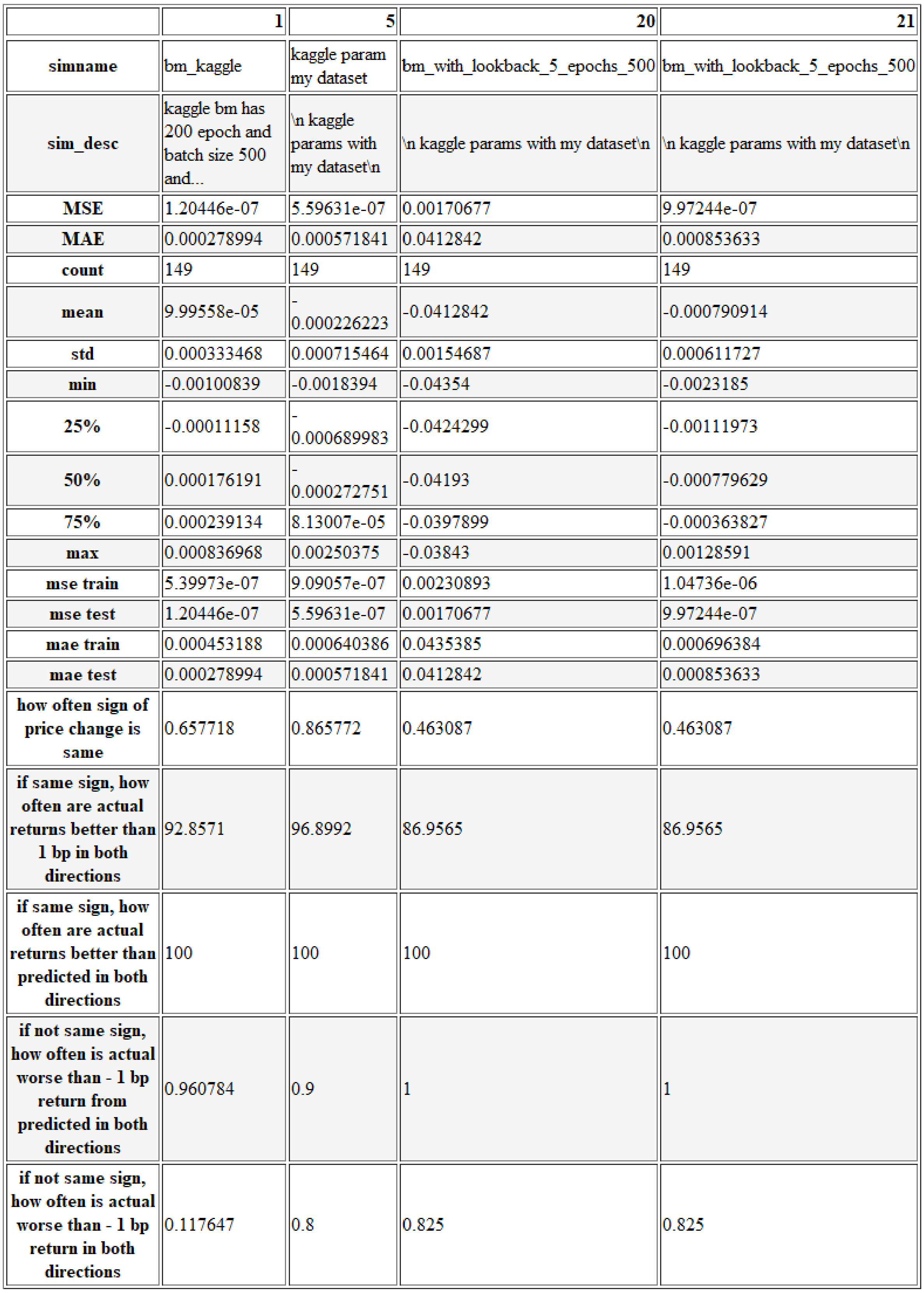
It can be seen that overfitting is not an issue, given testing error declines in line with training error. Further, we can see that testing error oscillates, indicating that the optimiser might be overshooting a bit when traveling along the gradients.

Normal shuffled KFold cross validation might not work, as the data is sequential and predicting prices in the past with information from the future would not work in practice – thus randomly shuffling the dataset is not a good idea. For timeseries a different cross validation technique is used. One needs to increase the training set in step, and use the next fold for validation – thus the training set will increase across validation steps, which might have the tendency to make future validations more accurate and earlier ones less accurate as they will have less data to train on. (http://francescopochetti.com/pythonic-cross-validation-time-series-pandas-scikit-learn/)

Regarding random states, there is no direct way to run the model for several random states. This is a feature request currently, but not implemented yet (https://github.com/keras-team/keras/issues/2743). What is possible is setting a certain random state to obtain reproducible results: https://keras.io/getting-started/faq/#how-can-i-obtain-reproducible-results-using-keras-during-development.

We tried to fix the random state and run the model again – this obtains a MAE of 0.04, a lot worse than the 0.00027 obtained in the final simulation without fixing the random state. Thus the unfortunate conclusion is that the results were probably down to the random state and more work is required. However, it is likely that this holds true for the benchmark model as well, as the random state was not set there either.

Interestingly, it seems rerunning the entire workbook, compared to rerunning the workbook from section results in very different MAE, despite setting the random see using the recommended hack:



	1	5	20	21
simname	bm_kaggle	kaggle param my dataset	bm_with_lookback_5_epochs_500	bm_with_lookback_5_epochs_500
sim_desc	kaggle bm has 200 epoch and batch size 500 and...	\n kaggle params with my dataset\n	\n kaggle params with my dataset\n	\n kaggle params with my dataset\n
MSE	1.20446e-07	5.59631e-07	0.00170677	9.97244e-07
MAE	0.000278994	0.000571841	0.0412842	0.000853633
count	149	149	149	149
mean	9.99558e-05	-0.000226223	-0.0412842	-0.000790914
std	0.000333468	0.000715464	0.00154687	0.000611727
min	-0.00100839	-0.0018394	-0.04354	-0.0023185
25%	-0.00011158	-0.000689983	-0.0424299	-0.00111973
50%	0.000176191	-0.000272751	-0.04193	-0.000779629
75%	0.000239134	8.13007e-05	-0.0397899	-0.000363827
max	0.000836968	0.00250375	-0.03843	0.00128591
mse train	5.39973e-07	9.09057e-07	0.00230893	1.04736e-06
mse test	1.20446e-07	5.59631e-07	0.00170677	9.97244e-07
mae train	0.000453188	0.000640386	0.0435385	0.000696384
mae test	0.000278994	0.000571841	0.0412842	0.000853633
how often sign of price change is same	0.657718	0.865772	0.463087	0.463087
if same sign, how often are actual returns better than 1 bp in both directions	92.8571	96.8992	86.9565	86.9565
if same sign, how often are actual returns better than predicted in both directions	100	100	100	100
if not same sign, how often is actual worse than - 1 bp return from predicted in both directions	0.960784	0.9	1	1
if not same sign, how often is actual worse than - 1 bp return in both directions	0.117647	0.8	0.825	0.825

Column 20 is the simulation rerunning everything from start, Column 21 the simulation restarting in section 10.3.

Justification

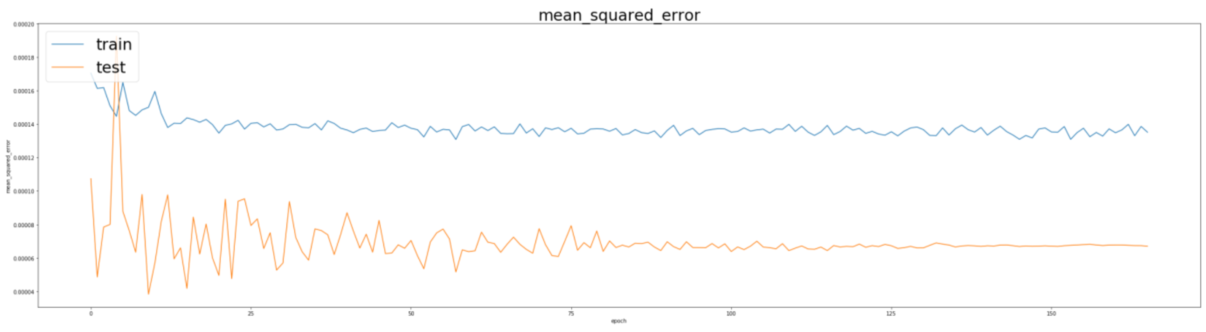
The final results are compared to the benchmark result or threshold with some type of statistical analysis. Justification is made as to whether the final model and solution is significant enough to have adequately solved the problem.

The final solution represents a small improvement over the benchmark model. This is possibly down to a lack of explanatory parameters in the price history only – one might have to take into account other financial variables such as news. The MAE improved and the directional accuracy is better as well.

Conclusion

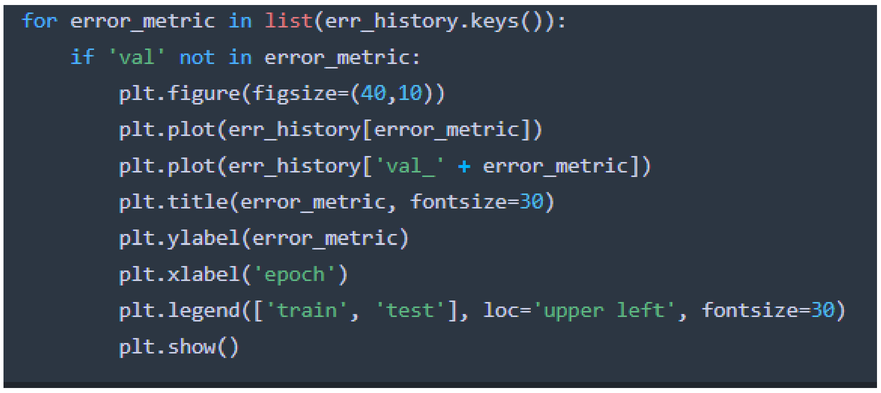
Free-Form Visualization

A visualization has been provided that emphasizes an important quality about the project with thorough discussion. Visual cues are clearly defined.



The above visualisation shows the MAE history for the final model. One can see that it initially oscillates and then settles at its final value long before we reach the last epoch. Thus one conclusion could be that the model is stuck in a minimum and cannot get out, or that I simply do not have enough information in my features to improve the MAE.

It does seem strange that test is always below train, but looking into the error history of the Keras model these are exactly the labels:


```
for error_metric in list(err_history.keys()):
    if 'val' not in error_metric:
        plt.figure(figsize=(40,10))
        plt.plot(err_history[error_metric])
        plt.plot(err_history['val_' + error_metric])
        plt.title(error_metric, fontsize=30)
        plt.ylabel(error_metric)
        plt.xlabel('epoch')
        plt.legend(['train', 'test'], loc='upper left', fontsize=30)
        plt.show()
```

It seems that this is a feature of LSTM models: https://keras.io/getting-started/faq/#is-the-data-shuffled-during-training. It has to do with the fact that loss on testing will be quite low, as it uses the final model at the end of an epoch, whereas training loss decreases during an epoch, and is an average of batches during an epoch, thus initial (not so well fitted) models will have an influence on the loss figure, resulting in a higher training loss reported per epoch than testing loss.

Reflection

Student adequately summarizes the end-to-end problem solution and discusses one or two particular aspects of the project they found interesting or difficult.

It was very interesting to investigate different properties of the dataset. It was difficult to get down the prediction error, possibly because I am solving for the wrong measure. A much more practical measure would be to predict whether I get the direction of the next price movement right, rather than its value. This would give less importance to small errors in value.

It was interesting to create my own error function and logging system, as well as a notebook than can run through in one go without needing human attention. This is very useful to quickly test a new parameter combination and search the parameter space.

My end to end problem solutions was the following:

- load data into sql server database
- compute 15 minute windows, with new features
- load data into notebook
- run linear regression
- add PCA as a feature
- feed updated dataset to LSTM model
- compare various parameters selections on LSTM model and select the best

The final result was that we improved marginally on the Benchmark Model in terms of MAE (from 0.00028 to 0.00027), but improved the directional error quite a lot (from 65% to 81% accuracy).

Improvement

Discussion is made as to how one aspect of the implementation could be improved. Potential solutions resulting from these improvements are considered and compared/contrasted to the current solution.

An improvement could be made by changing the problem to a classification, which would focus more on whether a move is positive or negative rather than getting the absolute value right.

I attempted a classification to predict the return class (positive, negative or zero), instead of the exact price. However, I might have the wrong understanding of inputs to the model for the binary classification case, as my attempt always predicts the same thing. Given that in my project proposal, binary classification was out of scope and I proposed this as a regression problem, I did not investigate further.

Quality

Presentation

Project report follows a well-organized structure and would be readily understood by its intended audience. Each section is written in a clear, concise and specific manner. Few grammatical and spelling mistakes are present. All resources used to complete the project are cited and referenced.

Functionality

Code is formatted neatly with comments that effectively explain complex implementations. Output produces similar results and solutions as to those discussed in the project.