Machine Learning Engineer Nanodegree Capstone Project

Stock Price Prediction Using Machine Learning and Regression

Ho Jun Liang

December 2020

Definition

Project Overview

Domain and Project Background

Investors make investment decisions based on expectations of stock prices in the future. Often, this prediction is done by creating financial/valuation models or developing a qualitative expectation based on current affairs that may affect the stock prices. The former would more popularly be referred to as technical analysis, while the latter is popularly known as fundamental analysis. Here, an attempt will be made to tackle technical analysis using machine learning.

This project will be exploring the use of the Long Short-Term Memory (LSTM) model, a deep learning sequential model. Jia (2016) investigated the effectiveness of LSTM networks for the use case of stock price prediction and showed that it was effective. This is because LSTMs are a specific type of recurrent neural network which have a form of 'memory' that makes use of previous time events to help to predict the next time event. This makes it quite suitable for the use case of stock price prediction, where data is in the form of time series.

Recurrent neural networks have two distinct and well-known problems – vanishing gradients and exploding gradients (Bengio et al., 1994) – because these networks tend to be very deep. The use of LSTMs helps to alleviate these problems, and is "achieved by an efficient, gradient-based algorithm for an architecture enforcing constant … error flow through internal states of special units" (Hochreiter and Schmidhuber, 1997).

Another problem common to neural networks would be that of overfitting. This is exacerbated by the fact that stock data has limited data points. Overfitting can be caused by an overly complex model and by insufficient data used for training. Overfitting can be mitigated by applying the following strategies – making use of more training data, reducing the dimensionality of the model using Principal Component Analysis (PCA) and using regularisation techniques. Additional training data can be obtained by using features generated from other component stocks in the S&P 500 which may have correlated movements with the target stock. PCA has been shown to improve the accuracy of recurrent neural networks because it reduces dimensionality and hence the complexity of the model (Berradi and Lazaar, 2019). Dropout is a form of regularisation for dealing with the problem of overfitting by randomly dropping units and connections during training of the neural network (Srivastava et al., 2014). However, it may not be good for use in LSTM models as some 'memory' may be 'forgotten' if these units are dropped.

Related data sets or input data

Training and testing data will be generated using historical daily stock data for the component stocks of the S&P 500 as found on the Yahoo Finance website. Yahoo Finance has comprehensive historical data over decades, and it also has ready-made APIs that can be made use of to extract this data. This mitigates any overly tedious attempt at creating a scraping tool from scratch. However, it is also noted that the official Yahoo Finance API was decommissioned in 2017. Hence, this project makes use of unofficial, user-created APIs for extraction of the data.

For each component stock, relevant fields to be extracted are as follows:

(1) 'Open' – opening price for the day
(2) 'High' – highest price the stock traded at for the day
(3) 'Low' – lowest price the stock traded at for the day
(4) 'Volume' – how many stocks were traded for the day
(5) 'Adjusted Close' – closing price adjusted for stock splits and dividends

The independent variables (or features) will be Open, High, Low and Volume; the dependent variable (or target) will be Adjusted Close.

The period for the use for training and testing can be determined by the user of the project code – the user will be prompted to enter a start and end date for the period of extraction. For the purposes of this report, it has been run on a range of extraction from 1 January 2000 to 30 November 2020 to ensure that there are enough data points for effective model training. A sample of the data points for a 5-day period for a stock ticker 'MMM' is shown in Figure 1 below.

	MMM_open	MMM_high	MMM_low	MMM_close	MMM_adjclose	MMM_volume
0	0.025821	0.025843	0.025792	0.025813	0.026489	-0.009411
1	0.004039	0.003955	0.004142	0.004079	0.002834	0.005582
2	-0.016311	-0.015952	-0.016796	-0.016381	-0.009636	0.035670
3	-0.000315	-0.000336	-0.000408	-0.000399	-0.002631	0.010229
4	-0.014047	-0.013882	-0.014300	-0.014138	-0.010881	0.008620

Figure 1: Extract of data fields for stock ticker 'MMM' over a five-day period

Given that stock data is relatively sparse for the purposes of training a deep learning model (as there are limited data points collected), this project will make use of the data points of all other S&P 500 stocks for model training to help to ensure that the model has sufficient data to train on. For example, if we want to predict the adjusted close price for the next 5 days, we will use all the independent variables for all 504 other S&P 500 component stocks as training data for the model, subject to further pre-processing and transformation as specified in more detail later in this report.

Problem Statement

The problem under investigation is the question 'What is a stock's price in a certain specified future time?'. This project seeks to predict stock prices of a stock based on historical time-series data for all other component stocks of the S&P 500.

This project will make use of a type of recurrent neural network (RNN) called Long Short-Term Memory (LSTM) to predict the required target stock's adjusted closing price over a certain future time interval. RNNs are particularly suited for handling data with sequence dependence, especially LSTMs which have a form of 'memory' that is very useful for time-series data like the historical stock price data that is being used for this project.

This project is written in Python, using the Keras API on a Tensorflow backend together with other relevant Python libraries that can be seen in the Python Jupyter Notebook as uploaded on Github. Principal Component Analysis (PCA) will be used to reduce the dimensionality of the dataset, given that the model will be built on a significant number of features – which will be generated from the other component stocks of the S&P 500 as well as from the target stock's independent variables. This dimensionality reduction seeks to also capture the most amount of information from the dataset and is useful for the purposes of this project. The performance of the LSTM will be benchmarked against a simple multiple linear regression model generated on the same target variable and feature variables as used for the LSTM model.

Metrics

Given that this is a regression problem, a common metric (that this project will be making use of as well) is the Root Mean Square Error (RMSE) which is the square root of the variance of the residuals. This metric is a measure of the average deviation of the predicted value from the actual value, and can be used to evaluate the accuracy of the predictions generated from the model.

While R-squared is also a metric commonly used with regression, it is not used for time-series data as R-squared mainly reflects how well a model fits past values rather than how well the model predicts future values. The fact that R-squared also tends to increase (or rather not decrease) with each additional feature although these may be uncorrelated to the target variable. Therefore, the above lends credence to the fact that even with a low R-squared for a time-series model, predictions may be poor and hence the unsuitability of using R-squared as a metric here.

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 of the data or input that need to be addressed have been identified.

As mentioned above, the dataset will consist of the following features – Open, High, Low and Volume – for all component stocks, as well as the Adjusted Close for all component stocks except for the target stock (i.e. the stock picked for prediction). The target will be the Adjusted Close for the target stock. The user will be given the option to pick the stock they would like to predict prices for when they run the code using Jupyter Notebook.

For the purposes of this project, the target stock chosen will be the stock with ticker 'PVH'. The time period for extraction of data will be 1 January 2000 to 30 November 2020, to maximise the number of data points used for both training and testing data.

A sample of the data extracted for a particular stock is shown in Figure 1 in the previous section.

From the dataset extracted, there are columns with 'NaN' values (i.e. missing values) which may be due to companies within the S&P 500 that have not been listed as of 1 Jan 2000 or have been delisted before 30 November 2020. Regardless of reason, as the neural network cannot take in missing values, these columns are dropped.

After these columns with 'NaN' values are dropped, the remaining data consists of 2,255 columns and 5,261 rows. The rows represent the data points for the date range while the columns represent the features that can be used for prediction for each of the stocks.

For the purposes of this project, a random target stock is picked to illustrate the model training and testing. The target stock's ticker symbol is 'PVH'. Figure 2 below shows the descriptive statistics for the target stock PVH as generated by the Python *pandas* module's 'describe' function. The mean of 62.26 is not far from the median (see '50%') of 53.26, indicating that there is only a relatively small skew. The minimum and maximum have a large variance, indicating that the stock is relatively volatile. This is understandable given that the time period selected is over a span of 20 years which is quite long and would encompass several financial crises (which are characterised by volatility in market prices).

```
In [7]: merged_data[chosen_adjclose].describe()

Out[7]: count    5261.000000
        mean       62.258707
        std        42.630146
        min         5.307689
        25%        22.210590
        50%        53.264847
        75%       102.160294
        max       167.603027
        Name: PVH_adjclose, dtype: float64
```

Figure 2: Descriptive statistics for adjusted closing price of PVH for selected date range

While another way of exploring the data would be by engineering new features like for example the simple moving average or obtaining the dollar value of the stocks traded daily by multiplying volume and the average price, this is not done as there are sufficient new features as generated if the other component stocks are used as features for the model. In fact, the opposite problem then surfaces – how to pick the correct features without losing the data represented by these features? This is where PCA comes in.

PCA is applied, with the number of principal components generated contingent on them explaining 99.9% of the variance. This percentage can be set by the user in the Python code. Upon application of PCA, the columns are transformed to the principal components created. A snapshot of the data has been included as in Figure 3 to illustrate the changes. Each component will be made up of a different composition of the original features. Figure 4 illustrates the contribution of each original feature to each component – rows here signify the component; columns here signify the original features; values are a measure of the contribution of each original feature to that component in the same row.

	0	1	2	3	4	5	\
2000-01-04	-35.960069	-23.552294	-4.597713	-17.453207	14.531658	-0.316768	
2000-01-05	-36.836905	-22.743318	-3.564498	-17.286059	13.297864	-0.338797	
2000-01-06	-37.487364	-22.279747	-2.514318	-16.556027	12.748698	1.037776	
2000-01-07	-37.299430	-21.751025	-2.901471	-16.130542	12.180725	1.006383	
2000-01-10	-36.589754	-21.807640	-3.366668	-16.198530	12.671839	1.904084	
...	...	...	...	...	...	...	
2020-11-20	90.183917	-29.257544	29.608164	1.440382	1.369148	-14.483319	
2020-11-23	90.293720	-29.171203	29.370437	1.104830	1.314000	-14.337519	
2020-11-24	91.138682	-28.987513	29.047628	1.801778	1.875908	-14.181955	
2020-11-25	93.325207	-26.612906	30.748859	3.270064	3.702518	-10.343187	
2020-11-27	92.693270	-29.498647	26.804316	1.978321	2.750035	-14.897862	

Figure 3: Snapshot of dataset after PCA

	MMM_open	MMM_high	MMM_low	MMM_close	MMM_adjclose	MMM_volume	ABT_open	ABT_high	ABT_low	ABT_close
0	0.025833	0.025854	0.025803	0.025825	0.026501	-0.009411	0.025723	0.025688	0.025770	0.025729
1	0.004184	0.004095	0.004292	0.004224	0.002885	0.005261	-0.012158	-0.012285	-0.012043	-0.012182
2	-0.016429	-0.016070	-0.016917	-0.016499	-0.009763	0.036224	0.021041	0.021366	0.020597	0.020982
3	0.000034	0.000007	-0.000050	-0.000049	-0.002404	0.009382	-0.000313	-0.000366	-0.000298	-0.000344
4	-0.014027	-0.013865	-0.014277	-0.014118	-0.010889	0.008232	0.000302	0.000395	0.000223	0.000325

Figure 4: Snapshot of the contribution of each original feature to the created principal components

Exploratory Visualisation

Exploratory visualisation is conducted to get a preliminary sense of the adjusted closing price for the stock over time. In Figure 5 below, we see the trend for the different variables for PVH from 1 Jan 2000 to 30 November 2020. The large dips in the prices for the PVH coincide with the 2008 financial crisis as well as the 2020 COVID-19 pandemic – both of which caused large market movements, and which explain the dip in prices. Looking at the overall trend from 2000 to 2020, we see that the prices track upwards. Also, the feature variables (Open, High, Low) tend to follow the same trend as the Adjusted Close target variable for PVH. This is intuitively reasonable as these different variables illustrate the variations within the short timespan of a day.

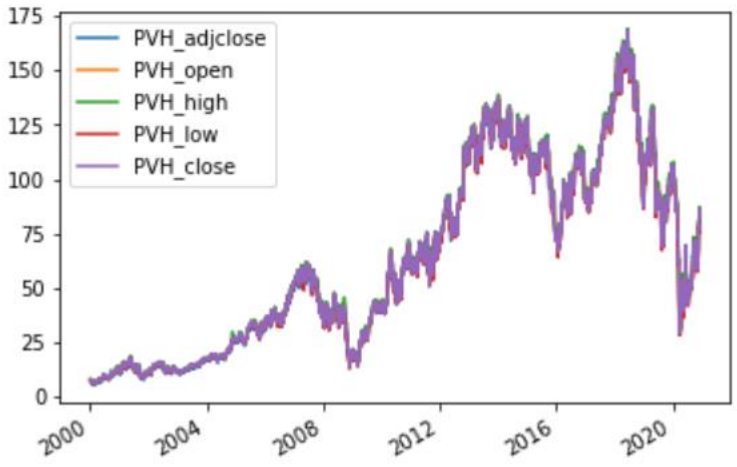


Figure 5: Plot of variables related to price against time for PVH

Figure 6 below illustrates the cumulative explained variance of each additional principal component after PCA is performed. The greatest increase in explained variance happens when there are approximately 10 components and gradually tapers to be negligible. This is reasonable as each component would have some correlation with each other which would increase with each incremental component, hence lending less explanatory power with more overlap.

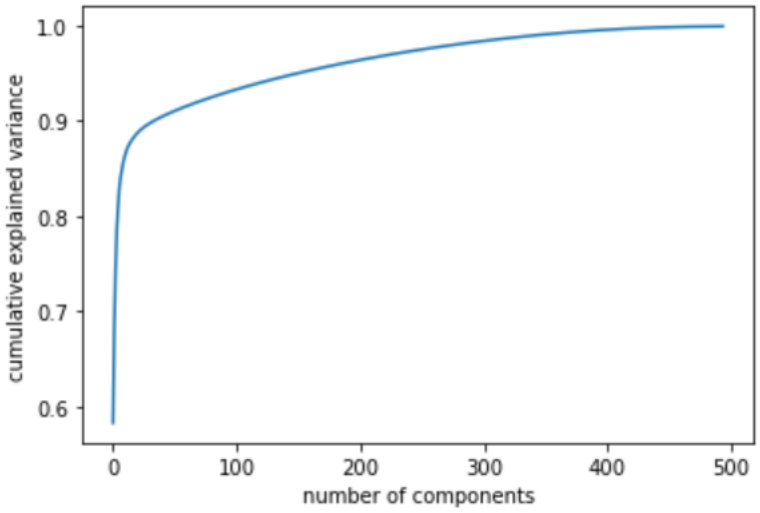


Figure 6: Plot of cumulative explained variance against the number of principal components

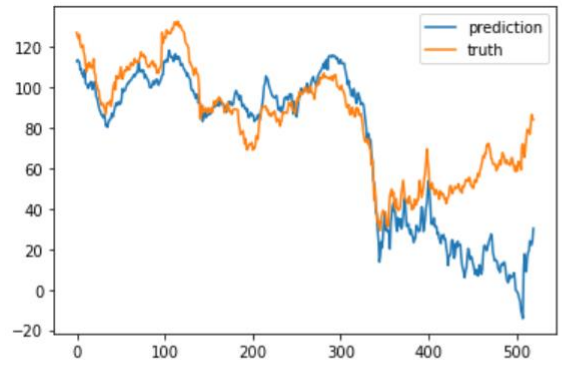
Algorithms and Techniques

A specific type of RNN called LSTM will be used here, with reasons as specified in the project overview above. The gist of why this is chosen is because LSTMs are highly suited for sequential models as they have the capacity to keep 'memory'. This means that stock data which is sequential data can theoretically be meaningfully explained with this model if parameters and the amount and type of data are appropriate.

Given that we have a large proportion of features as compared to the actual number of data points (2,255 versus 5,260), dimensionality reduction is applied using PCA so as to avoid overfitting. PCA is designed to maximise the explained variance by combining each individual feature into various components and yet to minimise the number of components so that there are fewer dimensions (or features). The literature review for this has also been explained in the project overview above.

Benchmark

The benchmark model to compare against would be the multiple linear regression model built on the principal components created after PCA as the independent variables/ features. The model makes use of lagged variables as input to predict the current period target variable. The R-squared after PCA is shown to be surprisingly good as can be seen in Figure 7 below. The intuition behind this might be that PCA helps to incorporate the complexity of the model into each component, and linear regression is run on the components which have already simplified the model by reducing the number of dimensions involved. The RMSE for this model is 23.61 (rounded to 2 decimal places) as seen in Figure 7.




```
from sklearn.metrics import r2_score, mean_squared_error
r2_score(label_array_test,prediction),np.sqrt(mean_squared_error(label_array_test,prediction))
```

```
(0.1547596269069199, 23.613411707770045)
```

Figure 7: Plot for linear regression with RMSE results for model evaluation

Methodology

Data Pre-processing

Extract and import data

The first step to be done was to extract and import the data. This was done using a Python library called *yahoo_fin*. To get all the component stocks' ticker symbols, the Wikipedia list of S&P 500 stocks was scraped using the *pandas* library. Two of the stocks' tickers were replaced as there were errors in extraction using these tickers later in the process. With this list of stock tickers, *yahoo_fin* was used to scrape the relevant data fields into a dictionary of dataframes.

Format data – remove missing values, merge, rename columns

This dictionary was then merged column-wise (by date).

Columns with missing values (denoted with 'NaN' by *pandas*) were removed as the LSTM would not be able to run with these missing values. Column names were transformed to the format <ticker symbol>_<relevant variable name>. An example of this would be 'MMM_open'.

Create target and feature variables

The user will be able to pick the target stock from a list of displayed stock tickers left in the dataset. The Adjusted Close column for this target stock will then be removed from the dataset and used as the target variable. The remaining features will be lagged by 1 day.

Implement PCA

PCA is implemented to reduce dimensionality. This will transform the dataset, a snapshot of which can be seen in Figure 3 above. Some data pre-processing is involved before PCA can be implemented.

Firstly, given that the data are all on different scales for different fields, they must be normalised so that they can be comparable and have the same mean and variance without distorting the differences in the range of values within each feature. The most obvious difference without normalisation would be between the Volume variable against all other variables which are related to the stock price. A

fit_transform function from the *StandardScaler* class in the *sklearn* library is implemented to help transform the dataset.

Then, the PCA is fit to generate the principal components. The *fit_transform* function above helps to ensure that each feature will not be weighted differently purely because of the absolute scale of the original feature, which would skew the composition of the components.

Split into training and testing datasets

The dataset is then split into training and testing datasets, with 90% of the original data set as the training data and the remaining 10% as the testing data. It seems that the usual default setting for this is 80% as training data and 20% as testing data in practice. However, some liberty has been taken to vary this because the relative number of data points for a deep learning model is small here. Hence, there is incentive to attempt to obtain as much training data as possible – increasing the percentage of the dataset set aside as training data is one way of doing so. It might also be good to note that in extremely large data sets, an even smaller proportion of data can be set as the testing data or even a specific numeric value for the number of data points can be set as the testing data.

An important note for the split into training and testing data sets is that this split is chronological and not random as the sequence matters in this use case of stock price prediction.

Transform the split datasets

The datasets above are transformed using the *MinMaxScaler* class from the *sklearn* library. This transformation is necessary to scale the inputs into the LSTM models in the range of 0 to 1. A fit_transform function is applied on the training features, while a transform function is applied on the testing features. The reason for this is that the fit_transform function scales the data and also learns the parameters used to scale that data; the transform function simply scales the data without learning another set of parameters. This helps to ensure that the model does not learn on the testing data, which is supposed to be kept independent from the model to simulate real-world input.

Sequence generation

Difference sequence lengths were tested, with the sequence length of 5 returning the best results. This sequence length of 5 moves along the stock data, with each step in the data recorded as one row in the sequence input. The dimensions of the data are also transformed such that they can be input into the LSTM model using Keras with Tensorflow backend. This process is implemented on both training and testing data.

Implementation and Refinement

Implementation of the model is done using Keras with Tensorflow backend. The use of such Python libraries greatly simplifies the technical implementation of the complicated LSTM and allows one to focus one's efforts on the iteration of the different hyperparameters to optimise the performance of the model.

Hyperparameter tuning was done manually, with multiple iterations and comparisons of the RMSE of the testing data to determine which is the best model. Hyperparameters that were tuned, and the range of values attempted are as follows:

(1) Number of hidden layers – 1 to 2

(2) Number of neurons/units in each hidden layer – 50 to 128
(3) Batch size – 4 to 128
(4) Epochs to run – 100 to 1000

Batch size was varied in powers of 2 so as to optimise the training speed. This was mentioned in Coursera's Deep Learning Specialisation by the instructor Andrew Ng. The number of epochs was also decided based on the shape of the curve of the chosen loss function against the number of epochs as seen in Figure 8 below.

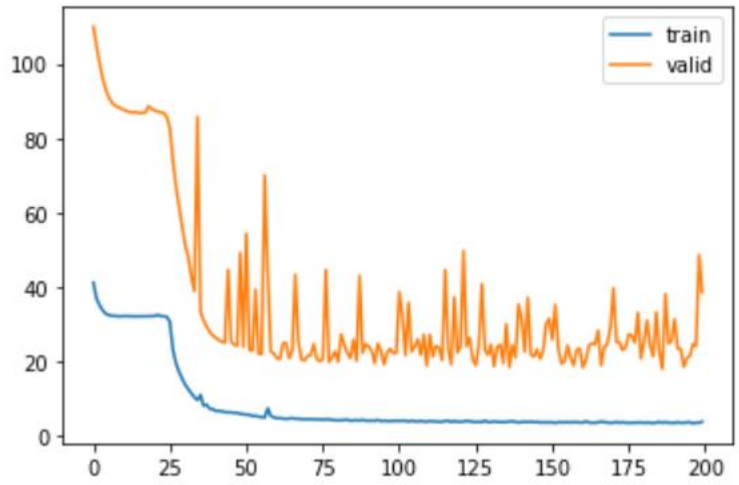


Figure 8: Plot of loss function against number of epochs

As can be seen in the figure above, the largest fall in the loss occurred within the first 25 epochs and gradually tapered off as the number of epochs approached 200. The optimal number of epochs to stop at would be at the elbow of the curve, however the decision was made to run for 200 epochs as there seemed to still be marginal loss reduction. The fluctuation for the validation curve is mainly because the validation set is chosen randomly.

One decision made was to set a seed number for the initialised weights. This is because weights are initialised randomly, which would make each run using the same parameters yield different results. For the sake of fair comparison between each set of hyperparameters, the weights are set with a seed number. This also makes results reproducible. For future improvement, the average of a few runs with each set of hyperparameters may be done instead of arbitrarily setting a seed number which may not be optimal.

The final hyperparameters used for the purposes of this report are as follows.

(1) Number of hidden layers – 2
(2) Number of neurons/units in each hidden layer – 75-50-50
(3) Batch size – 32
(4) Epochs to run – 200

Ideas for refinement for future improvements are as follows.

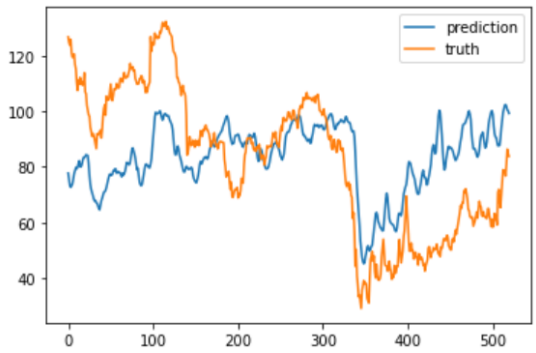
(1) Automatically tune hyperparameters – this can be done using Keras Tuner instead of manually selecting hyperparameters. A more systematic approach may be used instead of arbitrarily attempting to set hyperparameters.

(2) Create and transform additional variables like a weighted moving average which may conceptually make more sense than making use of features of other component stocks which may not be very correlated with the target stock.
(3) Use the average of the metrics for each set of hyperparameters instead of setting the seed number arbitrarily. This allows for randomness and also provides a certain additional degree of robustness to the model.

Results

Model Evaluation and Validation

Using the hyperparameters specified above, a plot of the value of the prediction and truth labels for the test data is generated as shown in Figure 9 below. The evaluation metric RMSE is relatively low – 23.59 (rounded to 2 dp). This is relatively low, and it can be seen visually from Figure 9 that the prediction results track closely with the actual ground truth labels. The results seem to track the closest approximately between the 150-day to 350-day mark, with the worst divergence approximately between the 1-day to 100-day mark and the 400-day to 500-day marks.




```
pd.DataFrame([np.squeeze(prediction),label_array_test]).T\
.rename({0:'prediction',1:'truth'},axis=1)\
.set_index(target.index[target.index> target.index[int(len(target) * 0.9) + sequence_length]])\
.to_csv('LSTM_prediction.csv')
```

```
r2_score(label_array_test,prediction),np.sqrt(mean_squared_error(label_array_test,prediction))
```

```
(0.15611925802638693, 23.594412112012265)
```

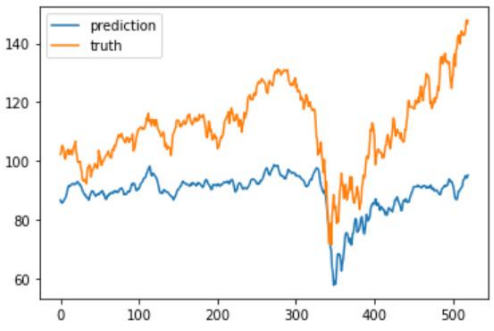
```
print(mean_squared_error (label_array_test,prediction))
```

```
556.696282911471
```

Figure 9: Plot of prediction and truth against the testing data dates for PVH (LSTM)

From the manual changing of hyperparameters, it was noted that additional epochs or additional hidden layers may not contribute much or may even be detrimental to the performance of the model. Models run on 1000 epochs performed worse than models run on 200 epochs in some instances, and this would have been visible on the graph of loss function against the number of epochs. Also, it was noted that in some cases, 1 hidden layer performed better than 2 hidden layers. This may indicate that the 1 hidden layer model was sufficiently complex to model the problem.

To test the robustness of the model, the model is re-run on a different target stock, PPG. The results for this can be seen in Figure 10 below. While the RMSE of PPG of 24.76 (rounded to 2 dp) is not as low as that for PVH, it is still sufficiently small for the model to be judged as good. This can be seen in comparison to the data range of minimum 10.86 and maximum 147.92993 for the target Adjusted Close variable seen below in Figure 11. This RMSE is also less than half of the mean as shown in Figure 11, further indicating that the model is a good fit.




```
In [32]: pd.DataFrame([np.squeeze(prediction),label_array_test]).T\
         .rename({0:'prediction',1:'truth'},axis=1)\
         .set_index(target.index[target.index> target.index[int(len(target) * 0.9) + sequence_length]])\
         .to_csv('LSTM_prediction.csv')

In [33]: r2_score(label_array_test,prediction),np.sqrt(mean_squared_error(label_array_test,prediction))

Out[33]: (-2.358749986470324, 24.735971676074776)
```

Figure 10: Plot of prediction and truth against the testing data dates for PPG (LSTM)

```
merged_data[chosen_adjclose].describe()
```

```
count    5261.000000
mean       51.795033
std        37.803501
min        10.860846
25%        20.305990
50%        28.547291
75%        93.142494
max       147.929993
Name: PPG_adjclose, dtype: float64
```

Figure 11: Descriptive statistics for PPG

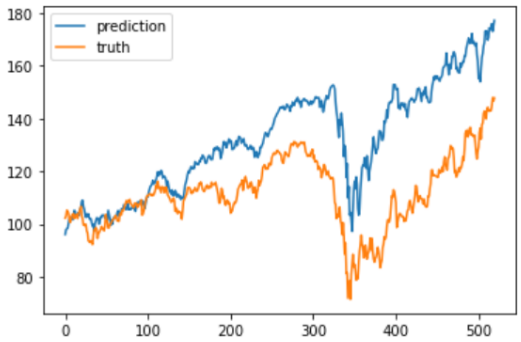
In conclusion, this model is robust and accurate enough to be used for stock market prediction, although it depends on the user's specifications as to the degree of accuracy needed. That being said, stock market prediction is not likely to be very accurate, as there are simply too many factors that have to be considered and included into the model.

Justification

The benchmark linear regression for PVH performed relatively well as can be seen in Figure 7 above, with RMSE of 23.61. The prediction results closely track the actual ground truth labels up to approximately the 400-day mark, and this is reasonable given that model is likely to be more inaccurate the further into the future it tries to predict. For PVH, the benchmark RMSE of 23.61 compared to the LSTM RMSE of 23.59 shows that the LSTM performs better than the benchmark model. However, it is also astonishing that a simple multiple linear regression model can perform so

well. The intuition for this has been specified above – it may be that PCA has helped to simplify the model such that even a 'simpler' model like linear regression is able to have good predictive power.

The robustness of the LSTM model can also be seen by comparing Figure 10 and Figure 12, from which we are able to compare the LSTM model against the benchmark model for PPG. We see the LSTM RMSE of 24.76 is still better than the benchmark RMSE of 24.88 (rounded to 2 dp).




```
]: from sklearn.metrics import r2_score, mean_squared_error
   r2_score(label_array_test,prediction),np.sqrt(mean_squared_error(label_array_test,prediction))

]: (-2.398249512825723, 24.880996363469674)
```

Figure 12: Plot of prediction and truth against number of days for PPG (benchmark model)

Conclusion

This project attempted to predict stock prices by making use of LSTMs. The number of data points and features were limited given the nature of the stock data and were hence supplemented with data points and features from other component stocks in the S&P 500 index. This created the problem of having too many features with not enough data points, which was then mitigated by implementing PCA. Both LSTM and the benchmark linear regression model performed well after PCA, and both proved to be robust. While the prediction of stock prices is not a simple problem, with many factors qualitative and quantitative that affect it, these models have shown to be good according to the chosen metric RMSE.

Future improvements would include making use of automatic or at least a more systematic approach to hyperparameter tuning. The current method of manual tuning requires a lot of human intervention and may not be comprehensive enough to obtain the actual best model.

The most difficult and time-consuming parts of the project are the data extraction and pre-processing, as many trial and error attempts had to be made before they could be correctly fed into the model for training. Similarly, the tuning of hyperparameters had to be narrowed down to a reasonable range of values through trial and error.

References

1. Jia H. (2016). Investigation into the effectiveness of long short-term memory networks for stock price prediction. arXiv preprint arXiv :1603.07893
2. Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. IEEE Transactions on Neural Networks, 5(2), 157-166. doi:10.1109/72.279181
3. Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. Neural Computation, 9(8), 1735-1780. doi:10.1162/neco.1997.9.8.1735
4. Berradi, Z., & Lazaar, M. (2019). Integration of Principal Component Analysis and Recurrent Neural Network to Forecast the Stock Price of Casablanca Stock Exchange. Procedia Computer Science, 148, 55-61. doi:10.1016/j.procs.2019.01.008
5. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research, 15(1).