UNIVERSITY OF EDINBURGH
Business School

2022-23

FINANCIAL MACHINE LEARNING | CMSE11475

ASSET PRICE WITH AUTOENCODER + LSTM

B213801

1. Abstract

The goal of the project is to train an **LSTM + Autoencoder** model to predict various stock's next-month return based on its past returns and technical indicators. This is done by building an Models for **Asset Pricing** (Models for determining the required or expected rate of return on an asset).

In this report, Asset Pricing is done using machine learning and deep learning techniques such as :

- **Neural network - Long Short-Term Memory (LSTM) in Model 1 & 2,**
- **Autoencoder in Model 1,**
- **Cross Validation -K Fold (Grid Search CV) in Model 3,**
- **Neural network - Regression in SHAP (Interpretation) in Model 4.**

The models being implemented consist of the **LSTM + Autoencoder model**, **Simple LSTM Model** and **XG Boost model**. The "LSTM + Autoencoder Model" and the "Simple LSTM Model" had good but **similar results** for forecasting predictions, whereas the "XG Boost Model" comparatively showed **better predictions.**

2. Introduction

Asset pricing refers to exploring the factors that determine the prices of and returns on financial assets, including stocks, bonds, currencies, and real estate. In this project we focus solely on stock price returns.

Here, asset pricing is done to determine the stocks future "returns". This allows one to make informed investment decisions and build a strong portfolio backed by research and predictive forecasts. The inverse can also be done, where, stocks with potential downside can be dropped off from a portfolio. This tells us that the model built can help with understanding Portfolio Risks, investment Decisions and various Business Analytics.

The first model used is an LSTM + autoencoder, which is a type of neural network that can learn a compact representation of a sequence of data. The input data is first compressed into a lower-dimensional representation by an encoding LSTM layer, then reconstructed back to the original sequence by a decoding LSTM layer. The goal of this model is to learn a useful representation of the input data that can be used for downstream tasks, such as forecasting or classification.

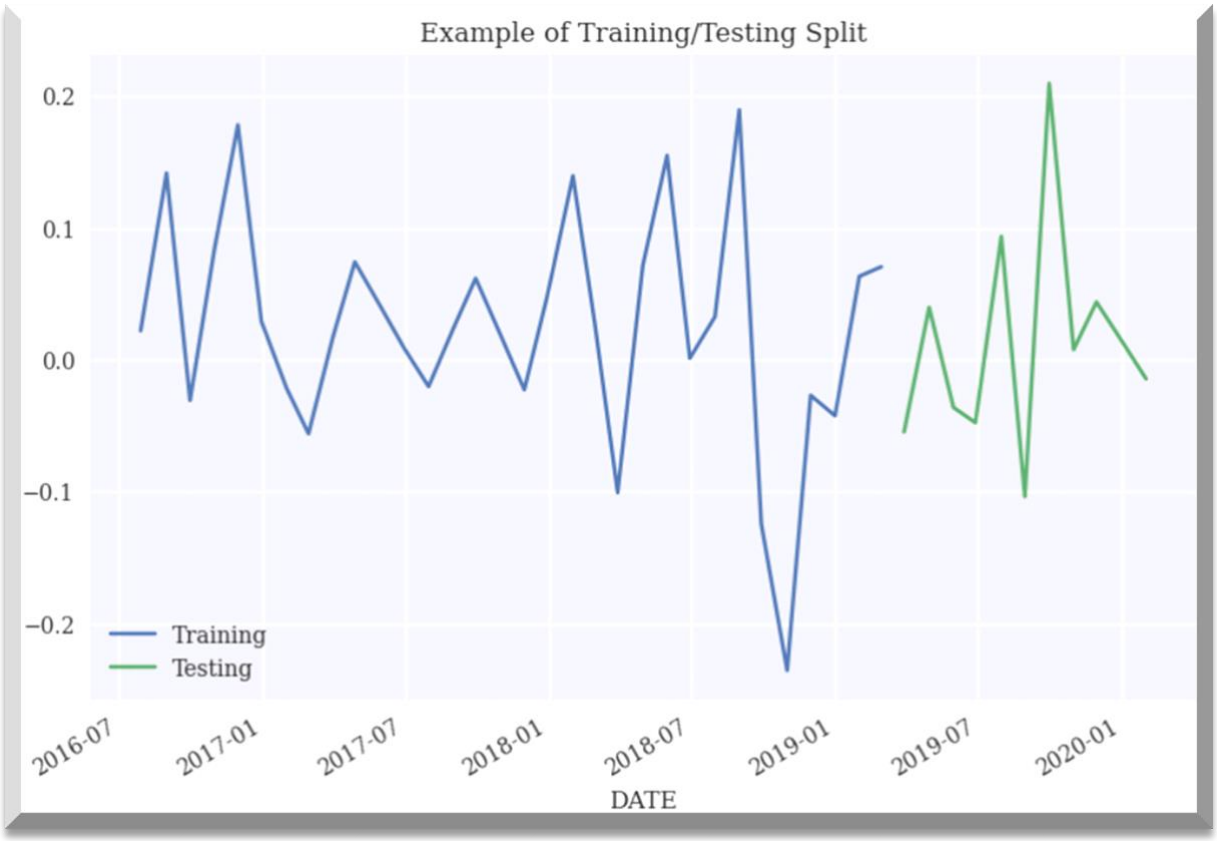
The second model used is a sequential model that uses Long Short-Term Memory (LSTM) neural networks for time series forecasting. LSTM is a type of **recurrent neural network (RNN)** that is capable of capturing long-term dependencies in sequential data. The model is trained to predict a single output value, given a sequence of input values.

The third model used is **XGBoost (Extreme Gradient Boosting)** is an ensemble method that combines multiple decision trees to make predictions. The XGBoost model uses a gradient boosting framework to iteratively improve the performance of the model by minimizing a loss function.

3. Data Set

The initial data contained above 9000+ stocks from 2010- 2020. The data was filtered down to ['2016-07-29':'2020-01-31']. The filtered data ranges through **1281 days with 7063 stocks** in total. Each stock has a unique number under the **"permno"** column and each stocks return is shown in the **"RET"** column. There are 98 predictors/indicators included in the dataset and "RET" is the **target variable** defined for the analysis.

To split the data for training, validation, and testing, the project uses a ratio of **0.25 (25%)** for the test set. The dataset is split into training and test sets based on the time index, with the first **75% of the data used for training and the last 25% used for testing**. A new column 'test flag' is created to mark the data rows as either training or test data.



The training data is fed to the all the models as sequences of **(8) eight consecutive months** of stock returns indicators, with the corresponding next-month return serving as the label. The feature engineering process involves selecting the relevant features from the raw data and normalizing them to improve the performance of the LSTM model.

The code first sets the seed (here, 50) for random number generation to ensure reproducibility. Then, it defines the length of the input sequence for the LSTM model as eight months. It creates two generator functions, one for generating input sequences and the other for generating labels, based on the selected features and the sequence length.

The code then creates empty lists to store the training and test data. It loops through each stock in the dataset and divides it into training and test data based on the test flag. For each sequence of eight months, the code appends it to the appropriate list based on the test flag. It also generates the corresponding label and appends it to the appropriate list.

Finally, the code converts the training and test data from lists to arrays and prints their shapes. It also prints the first three input sequences and corresponding labels from the training data.

4. Models

MODEL 1 – "Autoencoder + LSTM" :

The autoencoder is trained on a sequence of financial data, where the input sequence consists of 8 time steps (i.e., trading days) and 101 features (e.g., stock prices, volume, etc.). The output sequence is also 8 time steps long, and the model is trained to reconstruct the input sequence. The model architecture consists of two LSTM layers, each followed by dropout and batch normalization layers for regularization. The output layer is a time-distributed dense layer that outputs a single value for each feature at each time step.

After training the autoencoder, the encoder part of the model is extracted, and the input data is transformed into the encoded feature space. These encoded features are concatenated with the original input data and used as the input to a downstream regression model.

For this model, the **dropout rate used is 0.2**, and a **learning rate 0.001** of the optimizer. Sliding window is a technique that can be used to prepare the time series data for training the LSTM model. It involves sliding a window of fixed size over the time series and using the values within the window as input features to predict the next value. This allows the model to learn the patterns in the time series data and make accurate predictions. Input sequences are generated by sliding a window of a fixed size over the original sequence data, and the output sequence is the value immediately following the window. The window is then shifted by a certain number of steps, which is called the stride, to generate the next input-output pair. The

input shape of the LSTM autoencoder model here is **(4, 101),** which means that four consecutive rows of the original sequence data are used to forecast the output for the fifth row.

MODEL 2 – "Simple LSTM" :

This is a sequential model that uses Long Short-Term Memory (LSTM) neural networks for time series forecasting. The model is trained to predict a single output value, given a sequence of input values.

The model architecture consists of two LSTM layers, each with a **dropout rate of 0.5 to prevent overfitting**. **The first LSTM layer has 256 units** and returns a sequence of outputs for each time step, while **the second LSTM layer has 128 units** and returns a single output for the entire sequence. The output of the second LSTM layer is then passed through a dense layer with 50 units before being fed into the final output layer, which has a single unit to predict the target value.

The model is compiled with **mean squared error (MSE)** loss and the **Adam optimizer**. The early stopping call-back is used to monitor the validation loss and stop training if the loss does not improve for a certain number of epochs.

MODEL 3 – "XG Boost" :

XG Boost stands for "Extreme Gradient Boosting".It is trained using the gradient boosting method, where weak models (decision trees in this case) are added iteratively to the ensemble, and each subsequent model is trained to reduce the error of the previous models. The objective function used in the training process is mean squared error (MSE), which is minimized by adjusting the weights of the weak models.

To optimize the hyperparameters of the XG Boost regressor, the **Grid Search CV** function from the scikit-learn library is used. The hyperparameters that are optimized are 'colsample_bytree', 'learning_rate', 'n_estimators', 'subsample', and 'max_depth'. Grid Search CV performs a grid search over these parameter values specified in 'grid_search_params'. The best set of hyperparameters are selected based on the lowest negative mean squared error (MSE) score obtained from this **cross-validation.**

Once the best hyperparameters are found, the XG Boost regressor is trained on the training set using the optimized hyperparameters. The trained model is then used to make predictions on the test set, and the performance of the model is evaluated using the mean squared error (MSE) and mean absolute error (MAE) metrics.

The model uses a sliding window approach, where a portion of the historical data is used to train the model, and the model is then tested on a subsequent portion of the data. This process is repeated until all the data has been used for testing.

MODEL 4 – "SHAP":

A neural network regression model is constructed using TensorFlow's Keras API. The model has two layers, the first layer has 64 nodes with the 'relu' activation function and input dimension of the number of features in the training data. The second layer has 2 nodes, one for the stock ID and one for the stock returns. The model is compiled with mean squared error (mse) as the loss function and the 'adam' as the optimizer. Early Stopping call-back is used to stop the model training if the validation loss does not improve for 10 epochs.

Then the Kernel Explainer from the SHAP (SHapley Additive Explanations) library is used to define an explainer object that can be used to calculate the SHAP values of the features. The SHAP values are then calculated using the explainer object and the first 300 rows of the training data.

Finally, the SHAP values can be used to interpret the model and understand which features contribute the most to the stock return.

5. Results

- **If using Autoencoder + LSTM model, how many latent features yield the best forecasting performance?**

In the given Autoencoder + LSTM and Simple LSTM models, different numbers of latent features were tested to evaluate their impact on the forecasting performance. The models were trained and tested on a given dataset, and the number of latent features was varied to see which value would yield the best performance.

According to the results, using **256 latent features** yielded the best performance compared to 128 or 512 latent features. When 128 latent features were used, the model stopped learning after the patience level was breached, which suggests that the model was unable to learn and improve further. On the other hand, using 512 latent features resulted in the model learning, but , it did not perform as well as the model with 256 latent features.

- **What looking back window length generates the best forecasting performance in Autoencoder + LSTM, and LSTM alone models?**

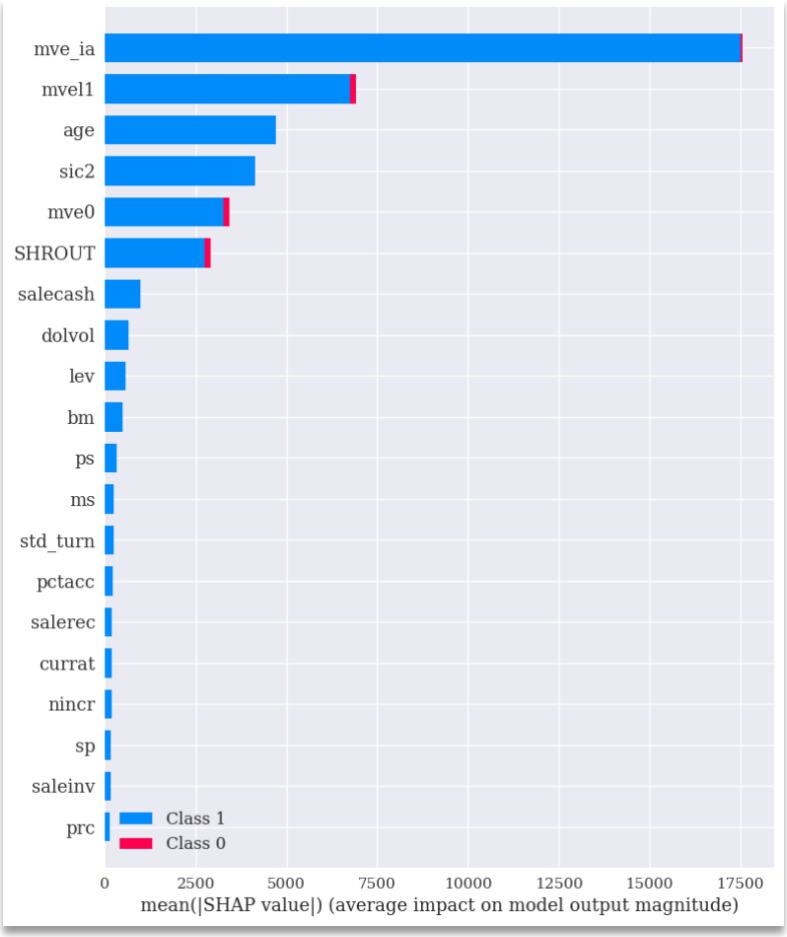
The looking back window length is a hyperparameter that determines how many time steps in the past the model will use to predict the next value.

From a data science perspective, the choice of the window length depends on the time series patterns present in the data. If the patterns have a long-term dependency, a larger window length might be more appropriate. On the other hand, if the patterns are more short-term, a smaller window length might work better.

From a financial perspective, the choice of the window length depends on the level of granularity and frequency of the data, as well as the investment horizon of the portfolio manager. For instance, if the portfolio manager is interested in making short-term trades, a smaller window length might be more appropriate. Conversely, if the manager is interested in longer-term investments, a larger window length might work better.

In this case, the experiments were conducted using different window lengths of 4, 6, 8, 10, and 15. The **window length of 8 resulted in the best performance.**

- **What company characteristics contribute to the stock return the most?**



The above graph depicts **the 20 company features that affect a stock's return the most**, these were ascertained to be the 20 most influential characteristics out of a possible 98. They were calculated using SHAP. These features provide insights into the financial health of a company and its ability to generate profits.

- **Do simpler models generate worse, or same or even better stock return?**

The model used to compare the models was Linear tree XG Boost. It had a lower MSE score than the Autoencoder model shown in the results below. **This shows us that the XG Boost generates better stock returns than the Autoencoder + LSTM Model.**

Grid Search Cross-Validation (CV) is used to tune the hyperparameters of the XG Boost model. Cross-validation is a technique to evaluate models by splitting the data into **k-folds** (subsets), training the model on k-1 folds, and evaluating it on the remaining 1 fold. This process is repeated k times, with each fold being used as the validation set exactly once.

Using Cross Validation on this model helped it perform better than both the Autoencoder + LSTM and the Simple LSTM models.

6. Appendix

Evaluation Results:

	MSE SCORE	MAE SCORE	MEAN	STD
MODEL 1	0.0440	0.0994	0.09961	0.00029
MODEL 2	0.0427	0.0985	0.09823	0.00022
MODEL 3	0.0259	0.0881		

7. References

Mullins, Jr., D.W. (2014) *Does the capital asset pricing model work?*, *Harvard Business Review*. Available at: https://hbr.org/1982/01/does-the-capital-asset-pricing-model-work (Accessed: April 6, 2023).

Colah (2015) *Understanding LSTM Networks*, 27 August. Available at: https://colah.github.io/posts/2015-08-Understanding-LSTMs/ (Accessed: April 2, 2023).