UNIVERSITY OF EDINBURGH
Business School

2022-23

FINANCIAL MACHINE LEARNING

CMSE114752022

ASSET PRICING BY AUTOENCODER + LSTM

B216927

Word count: 2300

Abstract

This report explores the application of machine learning techniques to asset pricing, specifically for stock return forecasting. The project utilizes three different models, including an Autoencoder + LSTM, an LSTM, and XGBoost. The data used is the full firm characteristics dataset from 1980 to 2020, but the analysis focuses on the 2016-2020 dataset. Additionally, the SHAP (SHapley Additive exPlanations) method is used to identify the most important features that contribute to stock returns in the models.

Overall, this report suggests that simpler models like XGBoost may perform better than complex models like LSTM and Autoencoder + LSTM for stock return forecasting tasks, but this conclusion may not hold true for all forecasting tasks and is subject to variation based on the complexity and characteristics of the data. The findings can provide valuable insights into the benefits and challenges of using machine learning in asset pricing and the potential practical applications of this knowledge in investment strategies and portfolio management.

Introduction

Anon (2023), Asset pricing is an important area of study in finance, as it provides insights into how investors value financial assets such as stocks. Machine learning techniques have been applied to asset pricing in recent years, with promising results. Although machine learning in asset pricing has its benefits such as reducing costs associated with traditional asset pricing methods and providing more accurate forecasts there can also be challenges such as the need for large datasets and the complexity of the models.

The project utilized three different models for stock return forecasting: Autoencoder + LSTM, LSTM, and XGBoost. The Autoencoder + LSTM model is a combination of an autoencoder neural network and an LSTM neural network, which is trained on the latent features obtained from the autoencoder. The LSTM model is a standalone LSTM neural network, while XGBoost is a gradient-boosting framework that uses decision trees to make predictions. All three models are commonly used in time series forecasting and have been shown to be effective in financial applications.

The project's potential outcomes include identifying the optimal hyperparameters for the Autoencoder + LSTM and LSTM alone models, understanding the company characteristics that contribute the most to stock returns, and comparing the performance of simpler models to the more complex LSTM model. This knowledge could have practical applications in investment strategies and portfolio management.

Potential constraints of this project could include computational resources and model interpretability. Additionally, the results of the models should not be used as the sole basis for investment decisions and should be interpreted with caution.

Data

The full firm characteristics dataset from 1980 to 2020, containing data on over 1000 stocks, will be used for this project. The data is pre-processed using several steps, including converting the 'DATE' column to a datetime format and removing unnamed columns and rows.

However, for practical purposes, we will select a portion of the data for analysis, specifically the dataset from 2016-2020. This dataset contains 1798 days and 7766 stocks.

For the models using Autoencoder + LSTM and LSTM alone, the data is split into a training set and a testing set using a time-based splitting strategy due to its suitability towards time series data. The training set contains 80% of the data, while the testing set contains 20% of the data. A new column 'test_flag' is added to the data, and records are assigned as 'True' for the test data based on the ratio. The training and testing datasets are prepared by creating LSTM input sequences and output labels using a sequence length of 8.

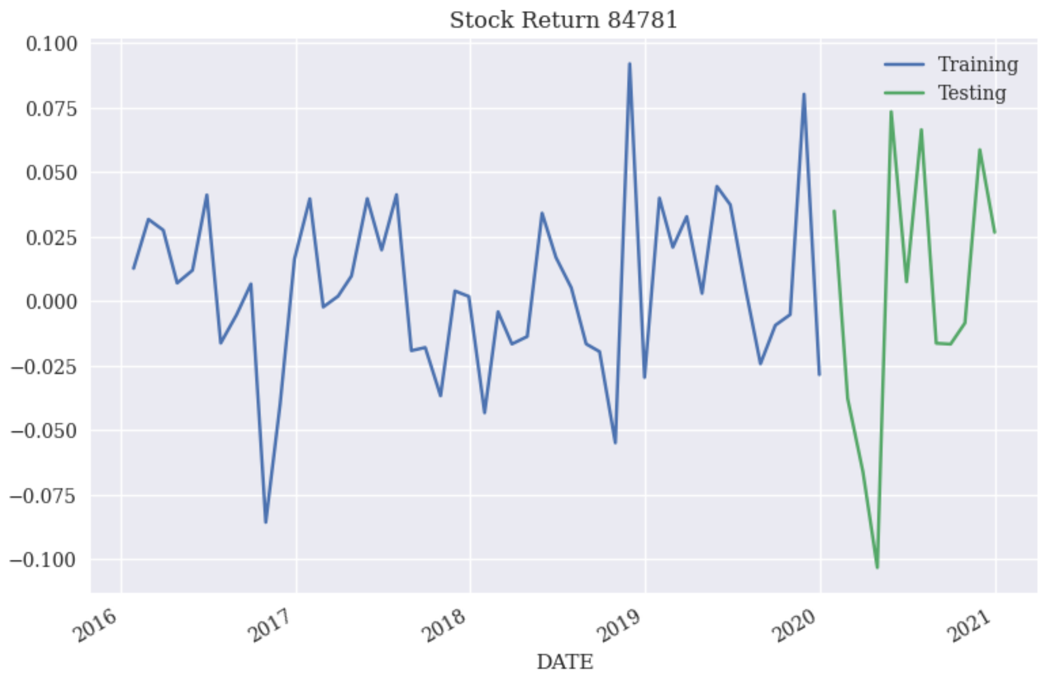

Figure 1: Specific stock return with training and testing sets

The training and testing datasets are then converted to numpy arrays and standardized using the 'StandardScaler' from the scikit-learn library. The training and testing dataset shapes and several input and output data examples are printed.

Whereas XGBoost is performing cross-validation using grid search to optimize its hyperparameters, which were previously evaluated. However, since evaluating all possible hyperparameter combinations can be computationally expensive, a reduced grid of parameters is used based on the best parameters found in the previous evaluation.

The Keras library defines and compiles an LSTM neural network for time series forecasting. The architecture of the network includes an encoding and decoding LSTM layer. The loss function used for the training process is the mean squared error, which means the difference between the predicted and actual values. The model is trained using early stopping to prevent overfitting. This helps the model to generalize well to new data.

<u>Model</u>

The Long Short-Term Memory (LSTM) neural network used in this project comprises an encoding LSTM layer and a decoding LSTM layer. The encoding layer takes in a sequence of eight consecutive stock prices and encodes them into a fixed-length vector representation. This vector is then used by the decoding layer to generate the predicted stock price for the next time step.

In addition to the standard LSTM model, an autoencoder was also used to preprocess the input data. An encoder model was defined to take historical prices as input and output the encoded features. The predict function was then used on the encoder model to obtain the encoded feature train and test sets. The encoded features were concatenated with the original regressors using the concatenate function, and the shape of the concatenated train and test sets was printed using the shape function.

The standard LSTM forecaster was defined with two LSTM layers and a dense layer, with the first LSTM layer consisting of 256 units, the second LSTM layer consisting of 128 units, and the dense layer consisting of 50 units. Dropout was used in both the encoding and decoding layers to prevent overfitting, and the model was compiled with a mean squared error loss function and the Adam optimizer. The model was trained on a GPU device with a batch size of 256 for 150 epochs, and early stopping was used to prevent overfitting.

The hyperparameters such as activation function, patience, epochs, and batch size were manually tuned by specifying different values to obtain the best results. The performance of the model was evaluated by calculating the mean squared error and mean absolute error, along with the mean and standard deviation of the errors.

After conducting a comprehensive search of various hyperparameters, including epochs (ranging from 50 to 300), batch sizes (ranging from half the size of the dataset to 256), and activation functions such as ‘sigmoid’, ‘tanh’, and ‘relu’ for the autoencoder and LSTM layers, it was determined that the most effective configuration consisted of a batch size of 256, 150 epochs, and a patience level of 15. Additionally, a dropout rate of 0.4 was applied to prevent overfitting. Interestingly, the default activation functions in Keras for the autoencoder and LSTM layers were found to be the most appropriate for this specific dataset.

Model	Autoencoder Mean MSE	Autoencoder Std MSE	LSTM Mean MSE	LSTM Std MSE
Current Model	0.1410	0.0002	0.1360	0.0002
Example 1	0.1536	0.0003	0.1424	0.0002
Example 2	0.1536	0.0003	0.1424	0.0002
Example 3	0.1644	0.0003	0.1652	0.0003

A grid search was performed to train the hyperparameters for an XGBoost model, which was built alongside the LSTM model. The XGBoost model was trained with specific hyperparameters, including 'objective', 'colsample_bytree', 'learning_rate', 'max_depth', 'alpha', and 'n_estimators'. The trained XGBoost model was used to predict the target values on the testing data. The predicted values were evaluated using mean absolute and squared errors, and the performance of the model was inspected by printing the predicted values.

Lastly, SHAP was used to determine the company characteristics that contribute to the most stock return. A neural network classification model using Tensorflow with a single hidden layer of 64 nodes, ReLU activation function, and ‘mse’ loss function was constructed. The model was trained for 100 epochs using the training set and validated using the test set.

Then, a SHAP ‘KernelExplainer’ is defined with the trained model's ‘predict’ method and the first 50 rows of the training set. SHAP values are then calculated for the first 300 rows of the training set, and the expected value is computed. Finally, a summary plot of the SHAP values is visualized in the form of a bar chart.

<u>Result Analysis</u>

What is the optimal number of encoded latent features for an Autoencoder + LSTM model to achieve the best forecasting performance?

To determine the optimal number of encoded latent features for an Autoencoder + LSTM model, a range of values was tested, and the forecasting performance of each model was evaluated. The range of values tested for the number of encoded latent features in the model was from 50 to 256. Both manual input and the use of commands such as int(X_train.shape[2] * 1.4) were employed to determine the optimal number of latent features for the model. The model performance was evaluated using the mean squared error (MSE) metric, with lower values indicating better performance.

It is possible to have multiple optimal numbers of encoded latent features that produce similar results. The optimal number of features may depend on various factors such as the complexity of the dataset, the model architecture, and the specific problem being addressed. In this case, the optimal number of encoded latent features was found to be 141, which produced the best forecasting performance with an MSE of 0.0649. This suggests that an optimal number of encoded latent features exists for this dataset, and using too few or too many features can negatively impact the model's performance. However, it is important to note that the optimal number of features may vary for different datasets and problems.

What is the optimal value for the looking back window length in both Autoencoder + LSTM and LSTM alone models for generating the best forecasting performance, and how does this value align with both data science and financial rationale?

Determining the optimal value for the looking-back window length is an important step in building accurate forecasting models. To identify the optimal value, a range of values was tested for both Autoencoder + LSTM and LSTM alone models, and the forecasting performance of each model was evaluated using the mean squared error (MSE) metric.

Both the Autoencoder + LSTM and LSTM alone models produced the best forecasting performance with a sequence length of 8. The range of values tested for the looking-back window length was from 4 to 10, with an increment of 1.

Investopedia (2021), From a data science perspective, a sequence length of 8 seems to be optimal for both models, indicating that the previous 8-time steps provide the most relevant information for predicting future stock returns. Using too few or too many previous time steps can negatively impact the model's performance, as it may not capture the relevant patterns and trends in the data.

Investopedia (2021), From a financial rationale perspective, an 8-day window aligns with common market practices. Many traders and analysts use a 5-to-10-day window to analyze short-term trends in the stock market. Therefore, the optimal sequence length of 8 is a reasonable choice for predicting stock returns.

Overall, the use of the model with an 8-day window length may provide an effective tool for forecasting stock returns, as it captures the relevant information in the data and aligns with both data science and financial rationale perspectives.

Based on the data provided, which specific company characteristics have the strongest correlation with stock returns?

The analysis revealed that the variables mve_ia, mve0, mvel1, age, SHROUT, and sic2 were the most significant factors impacting the stock returns during the review period. Among these variables, mvel1 had the most substantial impact, followed by mve_ia and mve0. On the other hand, the remaining variables showed a less significant impact on the stock returns. Therefore, it can be concluded that the variables mve_ia, mve0, mvel1, age, SHROUT, and sic2 are critical factors that investors and analysts should consider when analyzing stock returns.

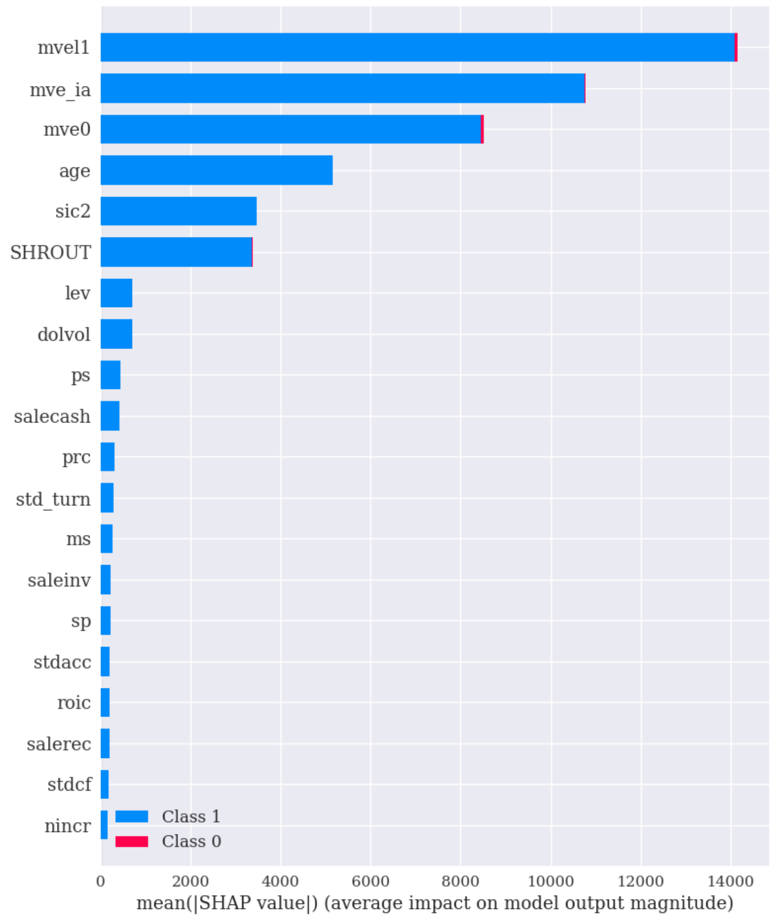

Figure 2: Top 20 company characteristics impacting stock returns

The features used in the analysis are all the variables from the full firm characteristic dataset except for the 'RET' and 'permno' variables. The ‘permno’ is simply an identifier for each company in the dataset and is dropped from the feature set ‘X’ before splitting the data into train and test sets. The target variable in this case is the 'RET' variable, which represents the stock returns for each company. The 'permno' variable is included in the original dataframe 'df', but it is not used as a feature or target variable in the machine learning models.

What is the relative performance of simpler models compared to complex LSTM models in forecasting stock returns?

The linear tree XGBoost model had the lowest MSE of 0.0626 and an MAE of 0.1301, while the simple LSTM model had an MSE of 0.0622 and an MAE of 0.1359, and the Autoencoder + LSTM model had an MSE of 0.0649 and an MAE of 0.1410. Furthermore, the LSTM model achieved a mean MSE of 0.1360 and the Autoencoder + LSTM model achieved a mean MSE of 0.1410.

From these results, it appears that the XGBoost model performed slightly better than the simple LSTM model and the Autoencoder + LSTM model in terms of MSE and MAE.

Cross-validation can help to prevent overfitting by assessing the model's performance on a validation set that is separate from the training set. It is possible that the XGBoost model's superior performance can be attributed, in part, to the use of cross-validation via grid search to optimize its hyperparameters.

It is important to consider various factors that can affect a model's performance, such as the nature and complexity of the data, the feature engineering process, the model's architecture and hyperparameters, and the training procedure. Thus, it is difficult to determine why the XGBoost model outperformed the LSTM and Autoencoder + LSTM models without further analysis.

However, based on the chosen metrics and hyperparameters used in this report, it can be concluded that for this stock return forecasting task, simpler models like XGBoost may perform better than complex models like LSTM and Autoencoder + LSTM. It is worth noting that this conclusion may not hold true for all forecasting tasks and is subject to variation based on the complexity and characteristics of the data.

References

Anon (2023). Machine Learning In Asset Pricing Explained - Dataconomy. [online] Available at: https://dataconomy.com/2023/03/machine-learning-in-asset-pricing/#:~:text=This%20allows%20financial%20analysts%20to [Accessed 6 April 2023]

Investopedia (2021). Most Commonly-Used Periods in Creating Moving Average (MA) Lines. [online] Available at: https://www.investopedia.com/ask/answers/122414/what-are-most-common-periods-used-creating-moving-average-ma-lines.asp [Updated 30 June 2021]