Project Work 2

Elias Eskelinen, Jarkko Komulainen, Matti Aalto, Vili Niemelä

November 30, 2025

1 Introduction

The aim of this project is to develop a model to predict the forward returns from the US stock market. To do this, we utilize a dataset (Hull et al. (2025)) containing US market data and daily returns data from buying the S&P 500 and selling it the following day.

2 Exploratory data analysis

The dataset consists of 9021 observations with 98 features. The features consists of a date id, which acts as an identifier for a single trading day, market dynamics and technical features, macro economic features, interest rate features, price/valuation features, volatility features, sentiment features, momentum features and dummy/binary features. There are also features for forward returns, i.e. the returns from buying the S&P 500 and selling it the following day, risk free rate i.e. the federal funds rate, and market forward excess returns i.e. forward returns relative to expectations.

The data contains plenty of missing data. Figure 1 shows that the first thousand datapoints have 85 missing features and only the last two thousand contain all the features. This is likely due to the data originating from a long period of time and data availability. Missing data must be taken into account in the model training phase and should be dealt with in data preprocessing.

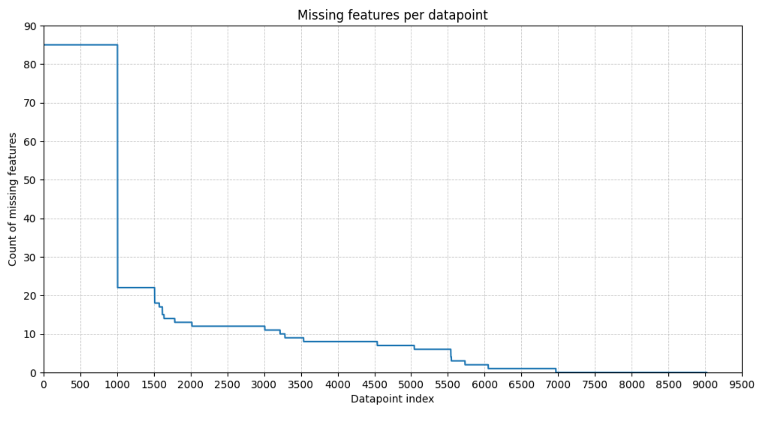


Figure 1: Missing features per datapoint.

Figure (2) shows the forward returns over time. From visual inspection, there does not seem to be any strong trends in the data, or even clear seasonality. However, the data shows that there are clear periods of higher volatility, where the variation in daily returns is high.

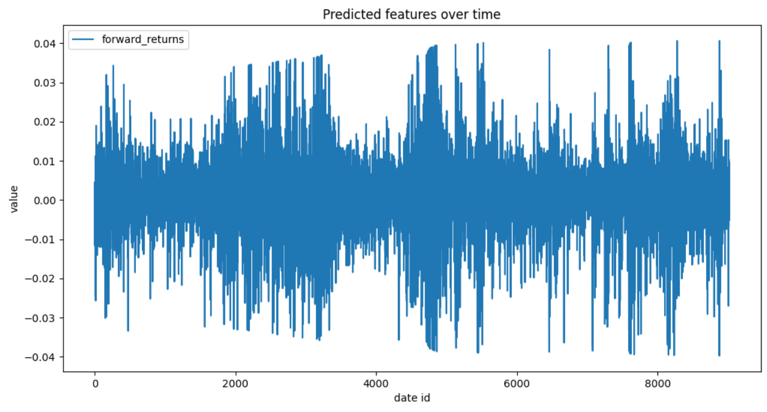


Figure 2: Forward returns time series.

The findings from figure (2) are supported by the distribution of forward returns observations, shown in the histogram (3). The values are distributed symmetrically around zero, with most of the density in the center. The high peak on the mean and symmetric spread around zero tells us, that most of the time the return is close to 0; high or low values are rarer, and getting positive returns is just as likely as getting negative returns.

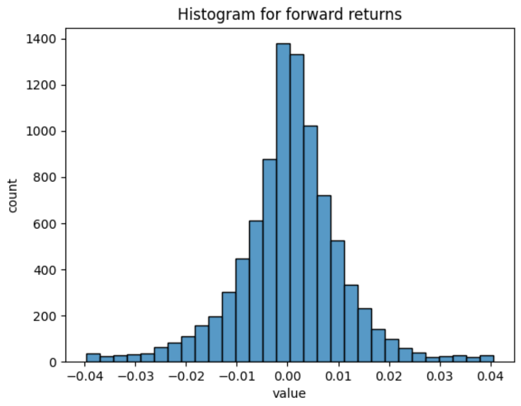


Figure 3: Forward returns histogram.

Figure (4) presents a correlation matrix of selected features in the data. Because of the high number of variables in the data, in order to keep the figure readable, the correlation matrix only shows the correlations for features which have strong ($|\text{corr}(x_i, x_j)| > 0.8,\ i \neq j$) positive or negative correlations with some other feature. The figure shows there are some strong correlations between the features, often between features of the same type. Notably, however, there are no strong correlations between the predictor features and forward returns.

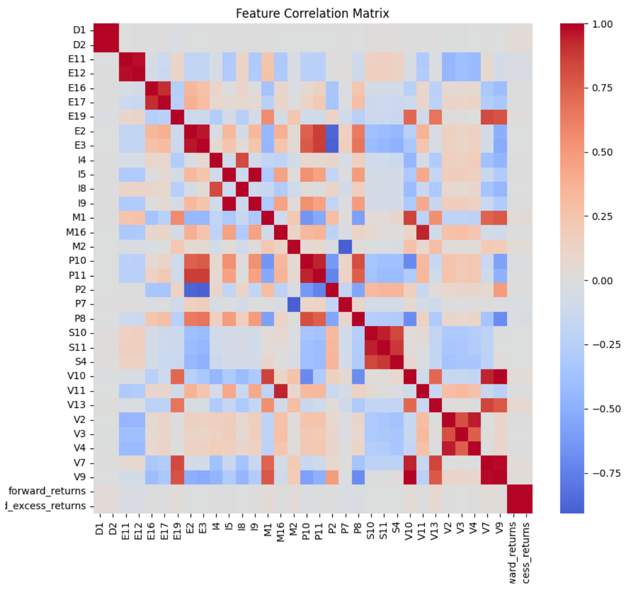


Figure 4: Correlation matrix for features with strong correlations.

2.1 Decomposition

Based on the Figure 2, the time series to predict does not seem to contain any clear long-term trend or seasonal pattern. In Figure 5 a zoomed trends of the time series can be seen. Based on the Figure, the time series does not seem to have either clear short-term trend or seasonal patterns. The time series seems to resemble a heteroscedastic Gaussian process.

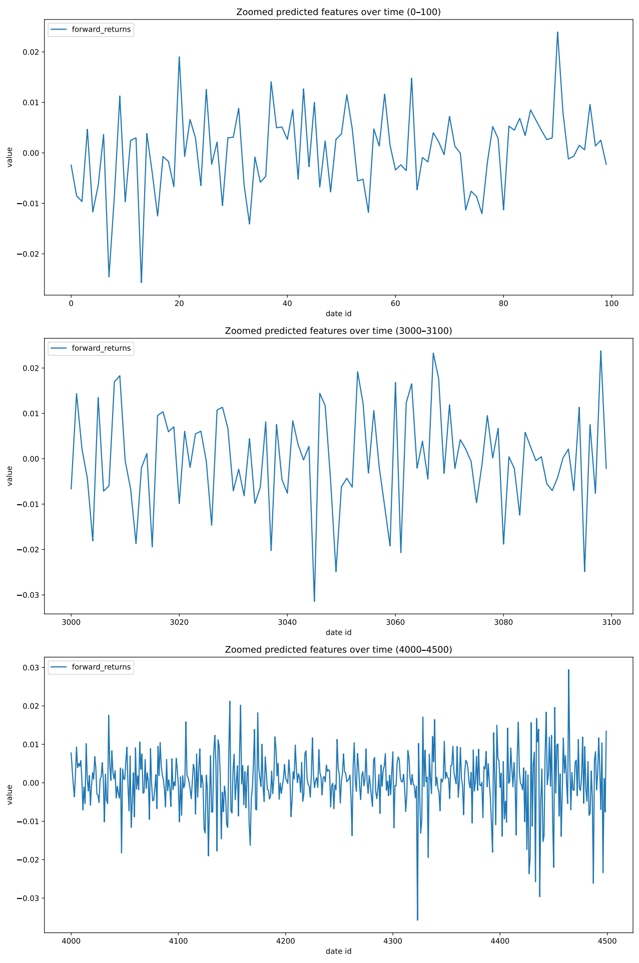


Figure 5: Zoomed forward returns.

We performed a regression analysis to the forward returns time series to confirm whether any linear trend can be identified. The estimated bias and trend coefficient terms are $[1.98082307 \times 10^{-8}, 3.82116136 \times 10^{-4}]$ which indicates that no long-term linear trend exists. As a conclusion, the forward return time series does not seem to have long-term trend or seasonal components to decompose.

2.2 Autocorrelation

In figure 6 we can see that the lag plot of forward returns looks like a round cloud and the ACF after lag 0 stays near zero, so yesterday's return doesn't help predict today's. When we fit a simple OLS model that uses yesterday's return to predict today's, the residuals look like noise. Their ACF is flat, so the mean part seems fine. But when we look at the absolute residuals, the ACF shows clear positive correlation that fades slowly. This could be interpreted to mean the volatility clusters. So larger moves and quieter periods tend to come in streaks. So the findings suggest that returns or their residuals themselves don't show autocorrelation, but the volatility does.

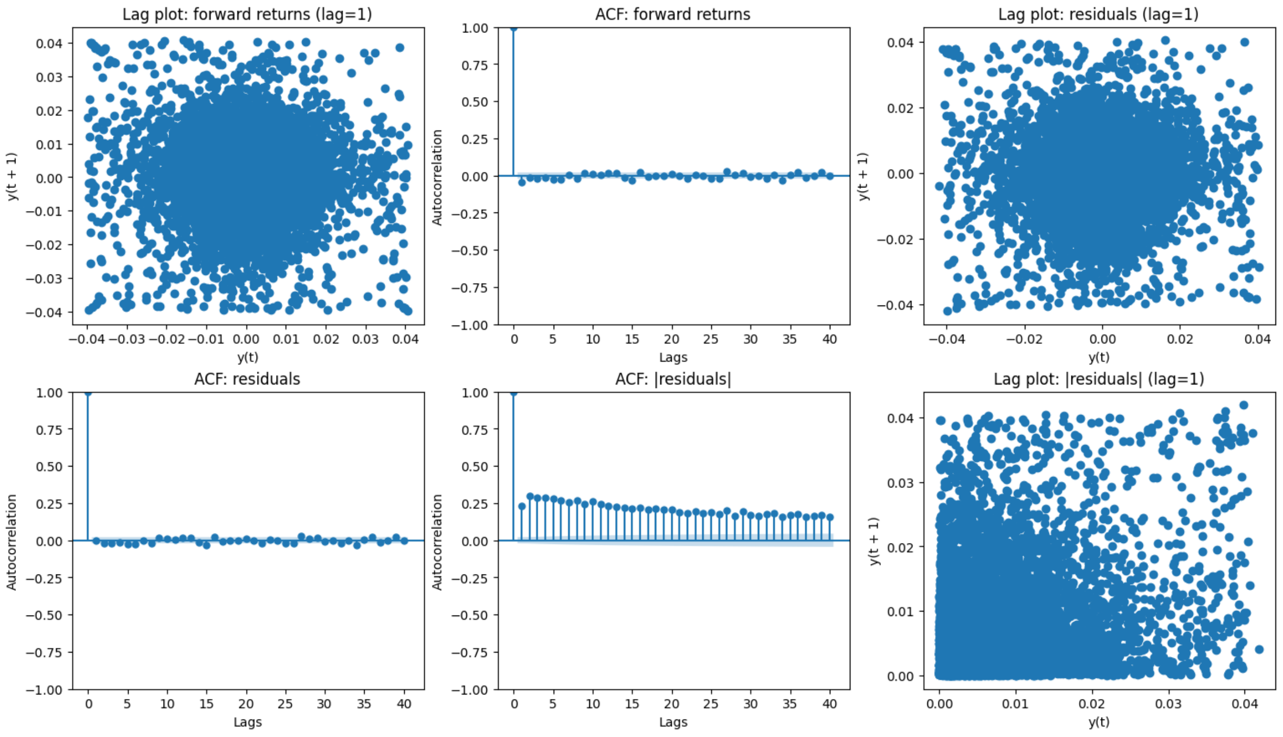


Figure 6: Autocorrelation plots.

2.3 Data partitioning for model formation

Partitioning a time-series dataset for model formation has its difficulties as the data has to keep its temporal structure. Our time-series is quite long and the variation seems to stay in a certain region. To perform robust training and validation for our model, we will use cross-validation in the form of a regular time-series split.

In a time-series split, we start with a smaller training data and validate on the next fixed amount of datapoints. Then we add the last validation set to our training set and validate on the next points. In this way, we use the most training data and we mimic the real life situation where we use all historical data to predict the future. The number of validation datapoints will be decided later, but we will start with 500. And the size of the first training data will be around a 1000 datapoints. These can change depending on the computational cost of the model and feature analysis. With this method, there is a possibility that the model learns patterns in the validation data and overperforms. If this seems to happen, we will test a slightly different form of splitting called the blocked time-series split, where the training data is always a fixed size and moves along with the validation data.

Our time-series has 9020 datapoints but many missing values in the beginning as we can see in figure 1. Because of this, we might have to ignore the first thousand datapoints, at least in the beginning. After we have performed an analysis on the features, we can better decide if we can use the first thousand datapoints and if we need to ignore more datapoints or predict some missing features. A little over two thousand of the last datapoints contain all the features and those datapoint will be used in the feture analysis.

3 Data pretreatment & baseline models

3.1 Data pretreatment

The data we have in our project, is in many ways very easy to work with. since it is not observations from any sensors etc. and the stock market's movement is tracked by numerous different parties, the data is in many way very complete. The absolute first thing we did was to remove the 2 other optional target variables that were provided in the dataset, which were *risk free rate* and *market forward excess returns.*

3.1.1 Continuity

The aforementioned facts apply especially continuity-wise. There are no gaps in the time-series, only consecutive days with records for each.

3.1.2 Asynchronicity

Again for the previously mentioned facts, the data is synchronous. We did not detect any asynchronicity in the features, everything is 1 observation per day.

3.1.3 Missing value handling

The data did have quite a lot of missing values. We observed that the first 1005 observations are missing pretty much all feature data. We assume these were not recorded yet by that time, as the data spans around 35 years. Then additionally there are features, which are missing a lot of consecutive observations, and some that do not have any records prior to around 500 or 1000 newest observations. We assume these are features that were begun to track only recently and cannot be deduced for the historical data.

As our initial approach, we chose to remove the first 1005 observations, and also remove the variables that were missing values. After this process, the data was left with zero missing values and 74 feature variables and 8013 observations.

3.2 STL decomposition for outlier detection

There are approximately 252 days in a year where the stock markets are open, so we decided to treat the observations in yearly cycles of 252 observations. We performed STL decomposition for the forward returns variable, to separate the seasonality, trend and residuals from each other.

STL decomposition separates the trend, seasonal and residual parts from the data, as seen in figure 7

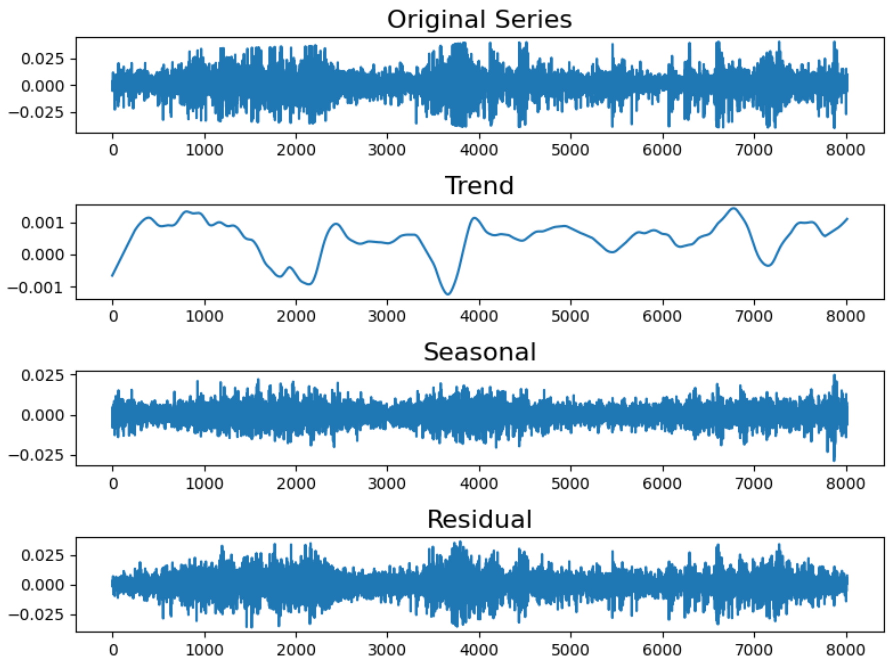


Figure 7: STL decomposition.

From the residuals, we performed anomaly detection by selecting the residuals that surpassed a limit of 3 times the standard deviation of the residuals. From this analysis, 97 anomalies were detected, which represents around 1.21% of all observations. In figure 8 we can see the the detected anomalies plotted in the original data.

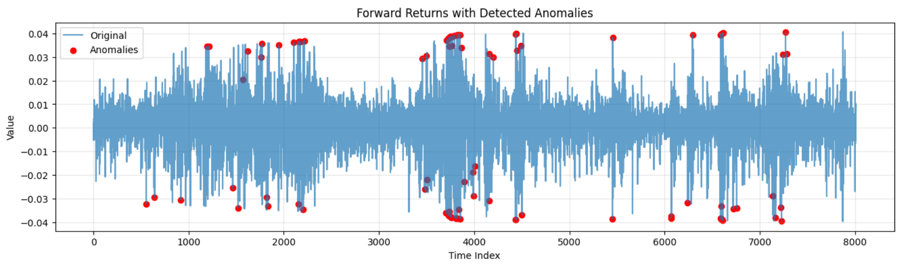


Figure 8: Detected anomalies.

3.3 Dimensionality reduction

The preprocessed dataset contains a high number of features (74), therefore we apply dimensionality reduction techniques to try and decrease the dimensionality in the data. We use two different dimensionality reduction to produce two different reduced datasets: linear dimensionality reduction method Principal Component Analysis (PCA), and the non-linear dimensionality reduction method kernel-PCA.

PCA decomposites the data matrix $\mathbf{X} \in \mathbb{R}^{n\times m}$ to matrices $\mathbf{T} \in \mathbb{R}^{n\times k}$ i.e. the *variable loadings*, $\mathbf{P} \in \mathbb{R}^{m\times k}$ i.e. the *object scores* and $\mathbf{E} \in \mathbb{R}^{n\times m}$ i.e. the error and noise according to equation (1). In essence, the method projects the data to a new k-dimensional subspace such that the first component vector of $\mathbf{P}$ corresponds to the dimension in the data with the most variance, the second vector to the dimension with the second highest amount of variance etc. In other words, the matrix $\mathbf{P}$ is the dimensionally reduced k-dimensional dataset which maximizes variance. (Salem and Hussein (2019))

$$\mathbf{X} = \mathbf{T}\mathbf{P}^\top + \mathbf{E} \tag{1}$$

Figure (9) shows the values of the principal components and the cumulative explained variance by principal components. According to the results, the dataset could be reduced to the first 30-40 component vectors while still retaining approximately 90-95 % of the variance.

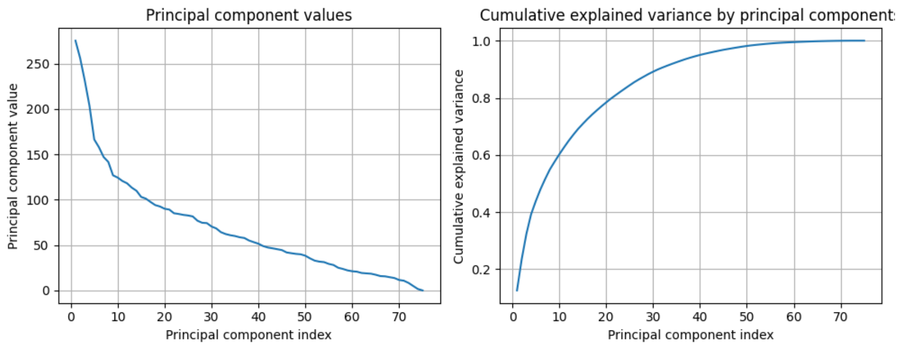


Figure 9: Principal components and cumulative explained variance.

Kernel-PCA (KPCA) extends standard PCA to non-linear spaces by using a kernel function $k(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_i) \rangle$ to calculate the covariance matrix (Gram matrix) $\mathbf{K}$ where $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$. Eigenvalue decomposition is then applied to extract principal components λ and component vectors $\mathbf{V}$. (Rosipal et al. (2001)).

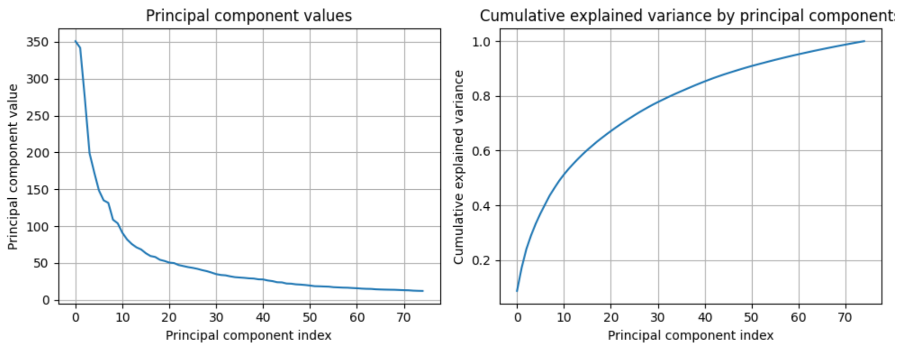


Figure 10: KPCA Principal components and cumulative explained variance.

Figure (10) shows the values of KPCA principal components and the cumulative explained variance by principal components. According to the results, the dataset could be reduced to the first 50-60 component vectors while still retaining approximately 90-95 % of the variance.

3.4 Literature review

Subsampling a long time series to smaller sequences is needed for robust results. Many of the methods used for subsampling are based on the fixed-origin and rolling-origin methods. In the fixed-origin method we use a fixed training set up to a certain timestep and validate the model on samples after that timestep. In rolling-origin we add more timesteps to the training data as we predict further along the timeline, as in if we are predicting sample from t+1 we can use all samples up to t to predict that. In a rolling-window method we drop out older samples from the test set to keep a fixed length set as we move forward. (Tashman, 2000).

Research by Li et al.(2023) suggests that dividing the seasonal timeseries to subsamples which each contain one season, could provide better results. This should be done with different seasonalities on the data, for example weekly, monthly and yearly.

Recurrent neural networks like long short-term memory can model some seasonality as they as they capture long term patterns. However deseasonalization should be used unless the seasonality is clearly homogeneous (Hewamalage et al., 2021). For deseasonalization another model, like seasonal trend decomposition, can be used together with the LSTM to form a more robust hybrid model (Jaiswal et al., 2025).

LSTMs are sensitive to input scale, so the inputs should be standardized or normalized. Some basic methods are min-max scaling and z-score scaling. More advanced methods are normalizing even between the hidden layers of the LSTM. This has can provide better generalization and faster model training times. (Cooijmans et al., 2017).

3.5 Baseline model

As a baseline model we decided to use a dynamic autoregressive model (AR-model) which uses 15 previous return values to predict next returns with a rolling window. The AR(15) model can expressed as follows

$$X_t = \beta_0 + \sum_{i=1}^{15} \beta_i X_{t-i} + \epsilon_t, \tag{2}$$

where X_t is the forward return at time index t and X_{t-i} is the i:th lag of the forward return. β_0 is the estimated constant term and β_i is the regression coefficient for the i:th lagged forward return value.

AR-model assumes that the time series is stationary meaning mean zero and constant variance. From the Figures above, it can be quite clearly seen that the forward returns time series is not stationary since the variance clearly changes with respect to time. However, our assumption is that the forward returns time series is locally stationary, meaning that small sample time series are stationary. We decided to select 100 length time series to test whether randomly selected such time series from the forward returns are stationary. If that is true when training the dynamic autoregressive models with 100 samples, we do not have to stationarize the time series always before training. To study whether our assumption is correct we used Augmented Dickey-Fuller test, which is a statistical stationary test for time series. Based on the test results, randomly selected 100 length sample time series from the forward returns are stationary. Because of the stationary conclusion, we decided to train the dynamic AR-models with 100 length time windows.

The training was done with the rolling time windows in a way that first the model is trained using 100 samples and then tested with the next 50 samples. The next training data starts from the index after the last index of the last training dataset. In Figure 11, the visualization of the used time series splitting technique is presented.

Two baseline models were implemented, one which predicted the forward returns one day ahead and another which predicted the returns 50 days ahead. For the both models, Python's statsmodels toolbox was used. From the Figures 12 and 13, the baseline model predictions one and 50 days ahead can be seen. Based on the Figures, a significant difference in the predictions accuracies cannot be seen which is why the models were evaluated with the root mean squared

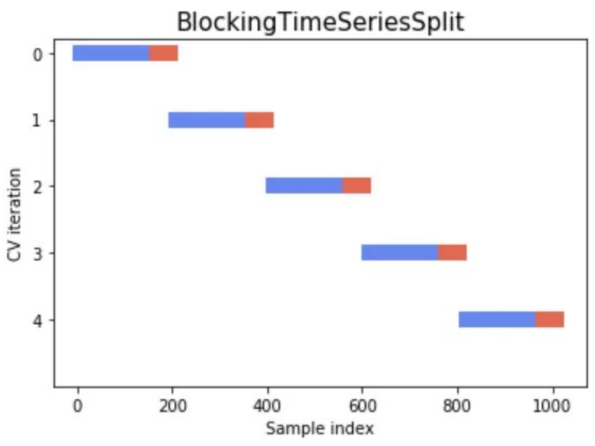


Figure 11: The method to split time series into training and validation datasets visualized.

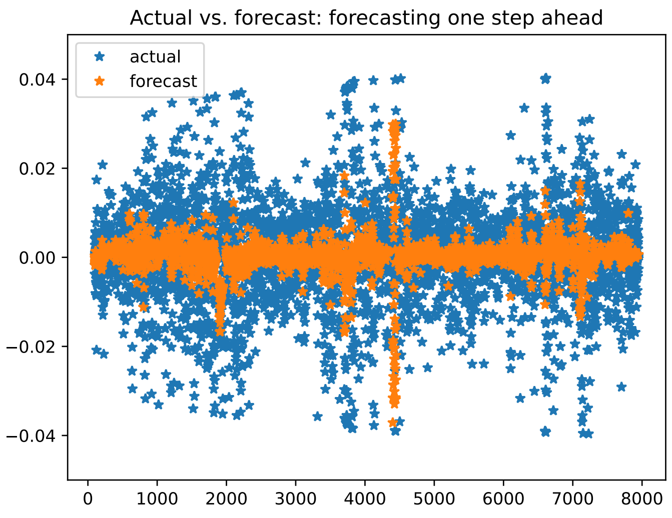


Figure 12: The one day ahead prediction results of the baseline model.

error metric which can be expressed as follows

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{i}^{n}(y_{\mathrm{true}} - y_{\mathrm{pred}})^2}, \tag{3}$$

where n is the number of test samples. The corresponding error metrics for the one day and 50 days ahead predictions were 0.0115227 and 0.014618 respectively. Based on the RMSE values, more accurate predictions can be obtained when predicting only one day ahead which was quite predictable.

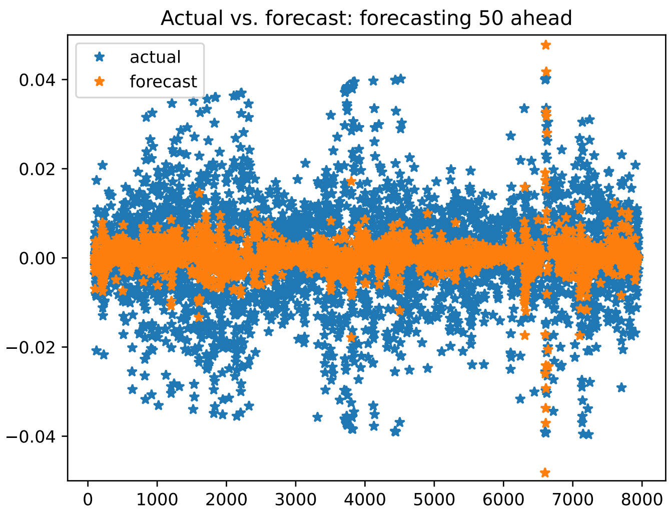


Figure 13: The 50 days ahead prediction results of the baseline model.

4 Modeling

4.1 Recurrent Neural Network (RNN)

Recurrent neural network is the basic neural network designed for processing sequential data, like text, speech and time-series data. It has more advanced variants, e.g. the latter model implemented here, the Long-Short Term Memory Neural Network.

4.1.1 Model architecture and used data

The RNN model we used has input layer, 1 RNN hidden layer, 1 dense layer with RELU activation and output layer that gives 1 output. The sizes of these layers were experimented with and are explained more in the following chapter.

The way the model uses the sequentiality of the data is that the model uses a sequence window of a defined step size, to predict the following 1 value. So for example the with a step size of 40, the model predicts the 41:st target value based on the first 40 samples. We used overlapping windows, so the model will progress a single index each time, predicting the 42:nd target value next.

We had multiple different feature datasets to choose from, but for this current version of the implementation we chose to use the PCA-processed data, and choose from those result latent variables.

The target variable y was normalized with Standardscaler and the calculated error was rescaled back to full size in the end.

4.1.2 Training

For our limited hyperparameter testing, we tried out the following values with the following results for errors:

Hidden units	Dense units	Batch size	Step size	Latent variables	MSE
32	8	64	50	30	0.217
64	16	64	50	30	0.406
128	32	64	50	30	0.525
64	16	64	10	30	0.506
64	16	64	100	30	0.493
64	16	64	50	10	0.320
64	16	64	50	70	0.351

Table 1: Results for different combinations of parameters

4.1.3 RNN visual results

Here are the results of the trained models, in visual terms. The training versus validation loss, and the fit on a part of the test set.

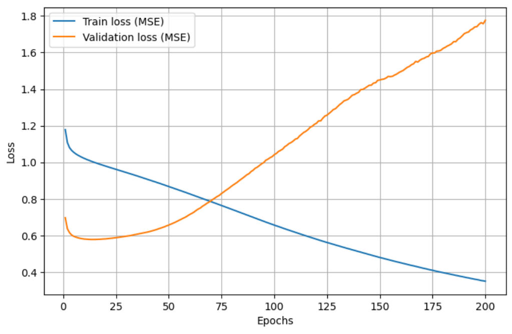
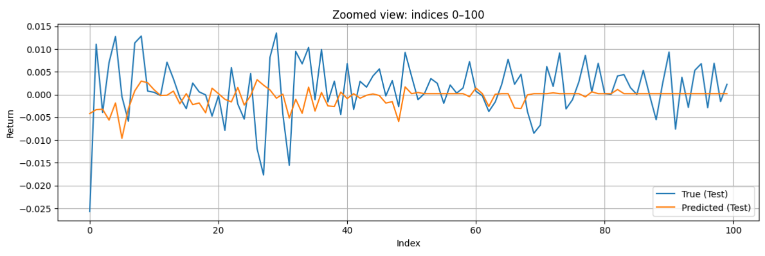
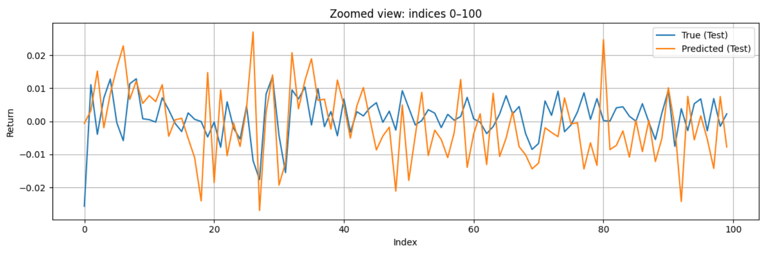


Figure 14: Validation and training losses of a trained RNN model



(a)



(b)

Figure 15: Predictions against actual values from a model with (a) 32 hidden units and 30 latent variables (b) 64 hidden units and 70 latent variables.

The train and validation losses in shown figure 14 were very similar for all tested models and it tells us that the models are somewhat overfitting to the training data and don't seem to learn many general patterns in the data. Together from the table 1 and figure 15 we see that a low mean squared error doesn't necessarily mean that the model makes good predictions as the model (a) had the lowest MSE but the predictions were in most places very off and just close to zero. The model (b) in figure 15 had almost all available features of the data in use and it was visually one of the better ones but it was still off by a lot in many points.

4.1.4 Improving the model

We had quite many ideas for improving the RNN, as the model in its current form is quite bad performing and simple. Firstly, proper cross-validation is an important one for choosing the latent variables. For this we planned to use time series cross-validation, where the data is split into sections, and the temporally next window will be used for validation, and then added into the training data for the next iteration.

Another really important improvement is proper in depth hyperparameter tuning. We tried out some different values for each, but there is still much room for improvement. To seek the best parameters and detect the best combinations, further testing is required.

As we only used the PCA-processed features, one improvement idea is to try out the other dimensionality reduction tool products too.

The final improvement we thought of, is removing the detected anomalies from the data, that we detected with STL. Possibility for this is to generate an average value for them from the adjacent values, or remove them completely and resample.

4.2 Long-Short Term Memory (LSTM) Neural Network

A LSTM neural network is a more advanced version of the recurrent neural network, which implements a more fine-grained control over the hidden state along with long-term memory. The network selectively inputs data to long-term memory, forgets data in long term memory and leaks long-term memory to the hidden state. (Aggarwal (2023))

4.2.1 LSTM results

In this section the results of LSTM model are presented. We implemented three different LSTM models: LSTM without dimensionality reduction, kernel-PCA LSTM, and PCA LSTM. We decided to use 70 sequences of length 100, and prediction horizon of 20. For the training we used the first 80% of the sequences. As a LSTM architecture we used two LSTM layers with 256 units in both. As an activation functions we used tanh-function and sigmoid-function for recurrent activation.

4.2.2 LSTM without dimensionality reduction

In Figure 16, training and validation losses with different epochs can be seen. Based on the Figure, the losses seems to converge to their levels already after 25 epochs meaning that more training does not seem to increase the accuracy of the model. In Figure 17, the model estimations on five training sequences are presented. When comparing to the prediction results on the testing data in Figure 18, it can be clearly seen that the model is overfitting to the training data since it is able to estimate the training data very accurately but the testing data quite poorly. The root mean squared error for the testing data is 0.058 which is worse than the baseline models' root mean squared errors.

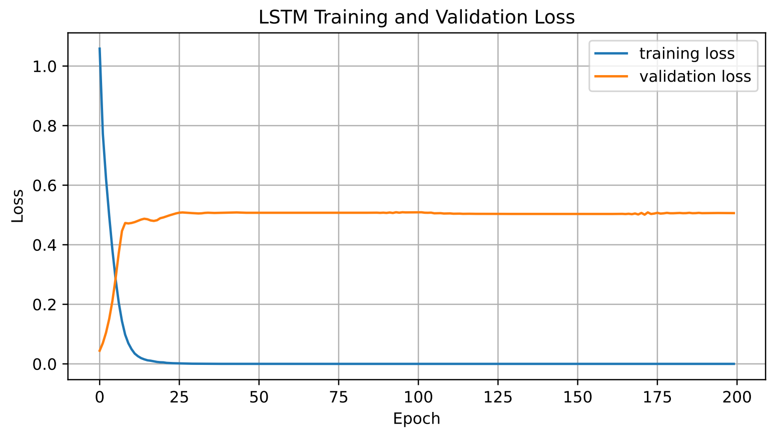


Figure 16: Validation and training loss of different epochs with LSTM without dimensionality reduction.

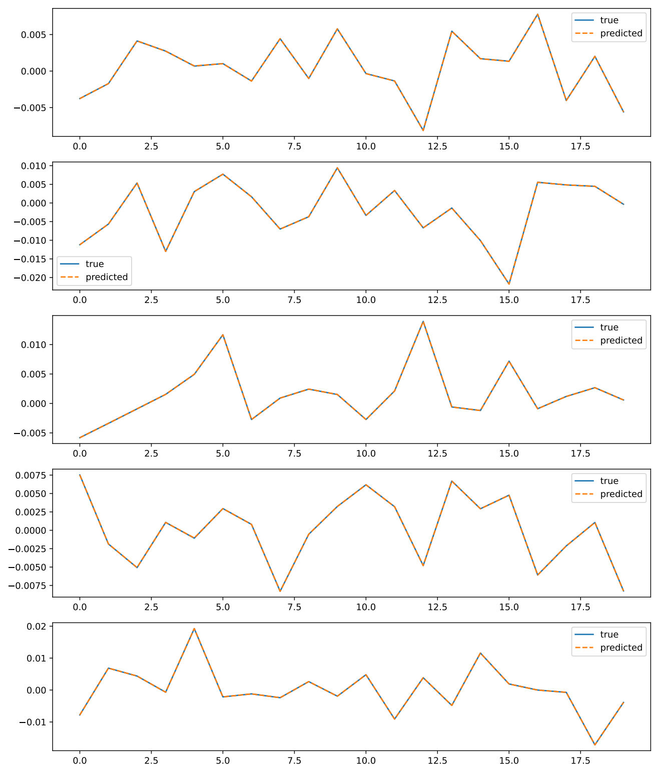


Figure 17: LSTM without dimensionality reduction predictions on training data.

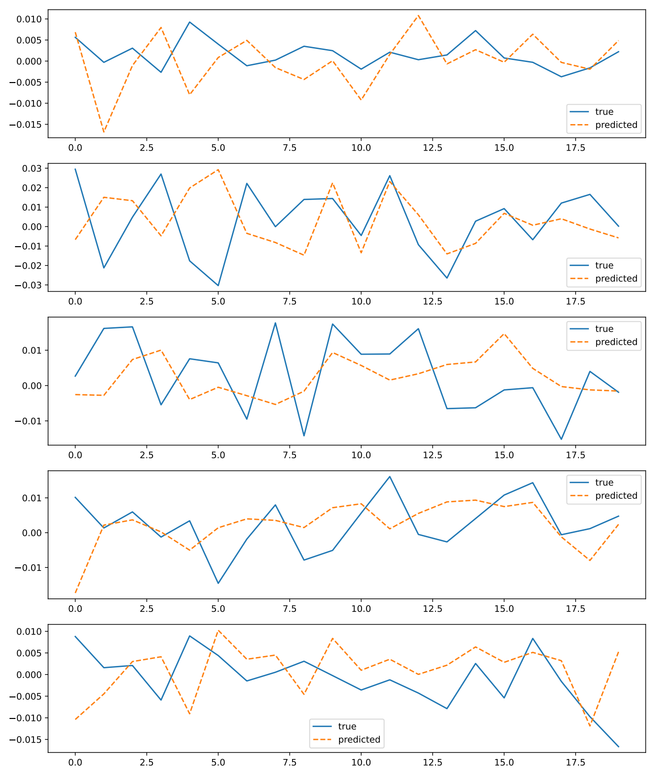


Figure 18: LSTM without dimensionality reduction predictions on testing data.

4.2.3 PCA LSTM

We fit the LSTM model to the first 30 components vectors from the PCA-dataset, which explain approximately 90 % of the variance in the original data. Model validation and training loss and error (mean absolute error) can be seen from figure (19); the figure shows that both training loss and accuracy improve during training, but validation loss and error get worse. This means that while the model does fit to the dataset, it does not generalize to the unseen test set.

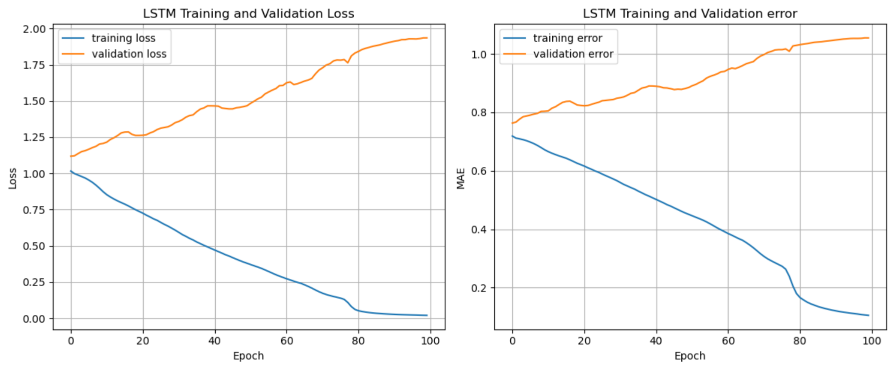


Figure 19: PCA-LSTM training and validation loss and error.

Figure (20) show the model's predictions on 50 observations from the training set. The results show that the model fits to the training set very well.

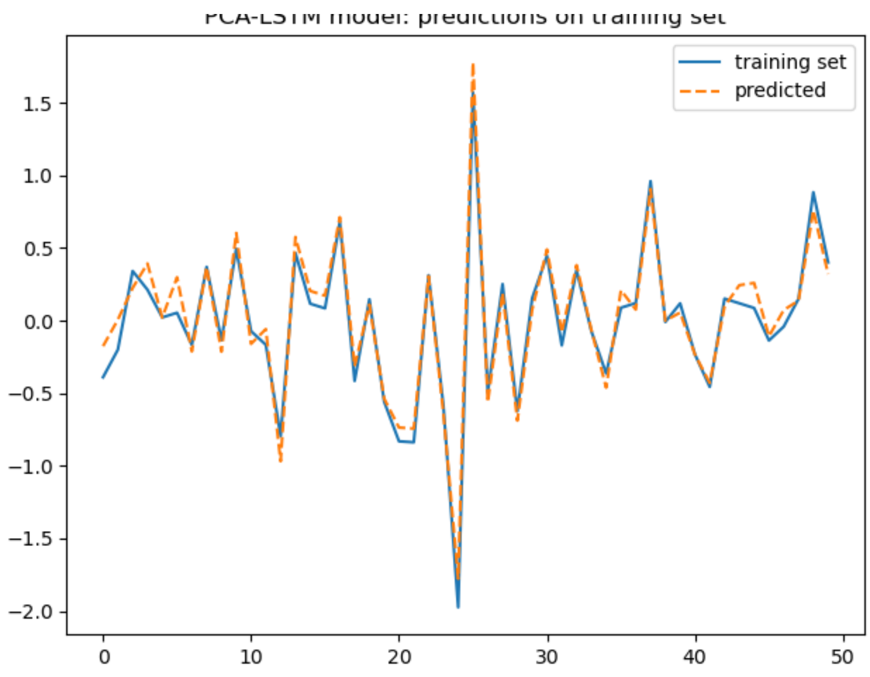


Figure 20: PCA-LSTM predictions on training set.

Figure (21) show the model's predictions on the first 50 observations from the data; as can be seen, the first couple of predictions are very accurate, but the predictions get progressively worse as the model predicts observations further away from the training set. A possible solution to this is proposed in subsection (4.2.5).

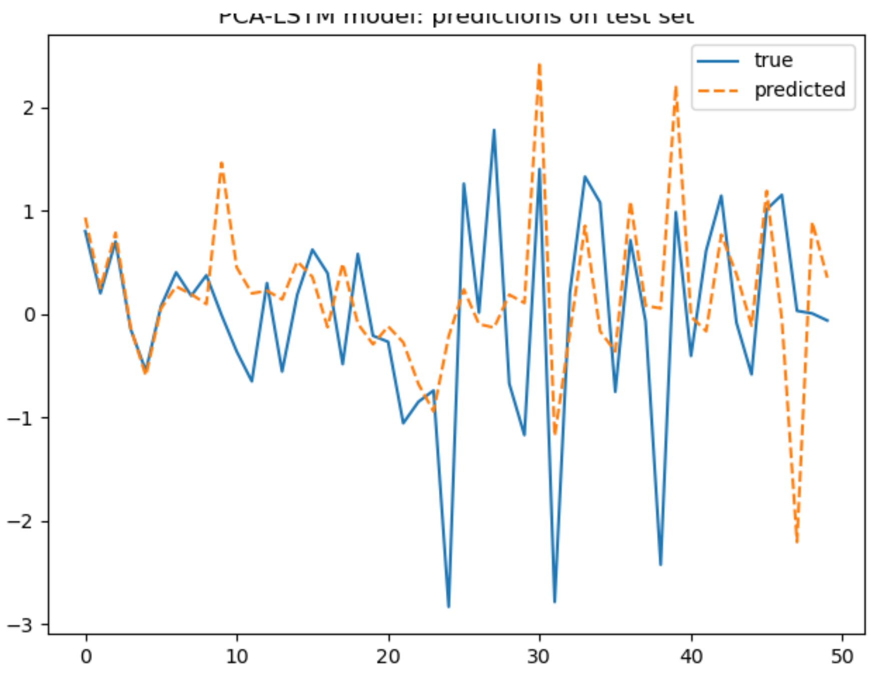


Figure 21: PCA-LSTM predictions on test set.

4.2.4 Kernel-PCA LSTM

Based on the Figure 10, we decided to use 50 kernel principal compounds, since with 50 principal compounds the selected cumulative variance is over 90%. As a kernel we used radial basis function kernel.

Training and validation losses of different epochs can be seen in Figure 22 where it can be seen that the model seems to overfit to the data since the training loss is much lower than the validation loss. Increasing the number of epochs seems not to help since the losses seems to stabilize to their levels after 75 epochs. In Figure 23, the estimation results of the model on five training sequences can be seen. When comparing to the prediction results on the testing data in Figure 24, it can be clearly seen that the model is able to estimate the training data very accurately but testing data quite poorly. The root mean squared error for the model on the test data is 0.055, which is worse than for the baseline models.

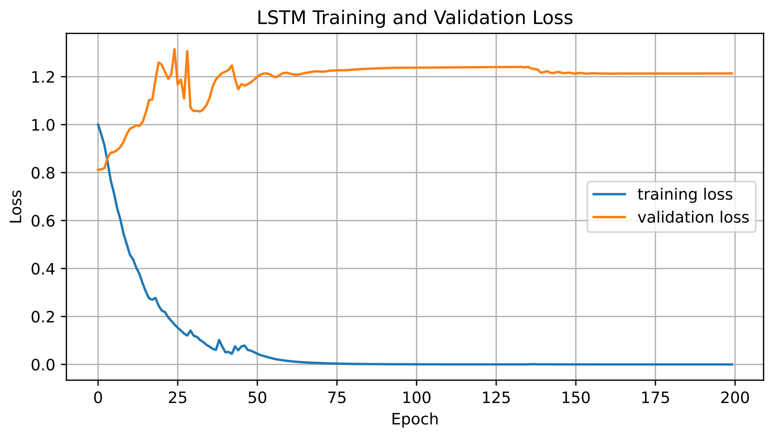


Figure 22: Validation and training loss of different epochs, kernel-PCA LSTM.

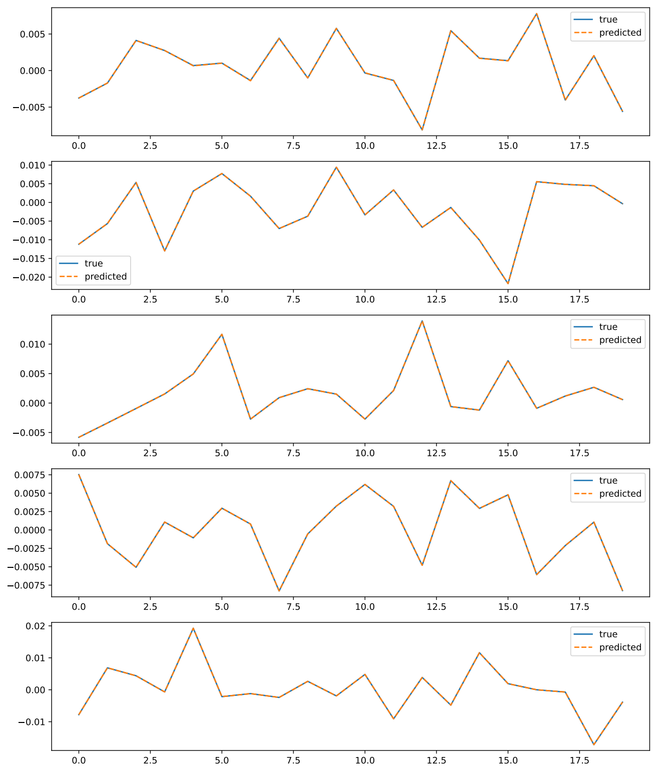


Figure 23: Kernel-PCA LSTM predictions on training data.

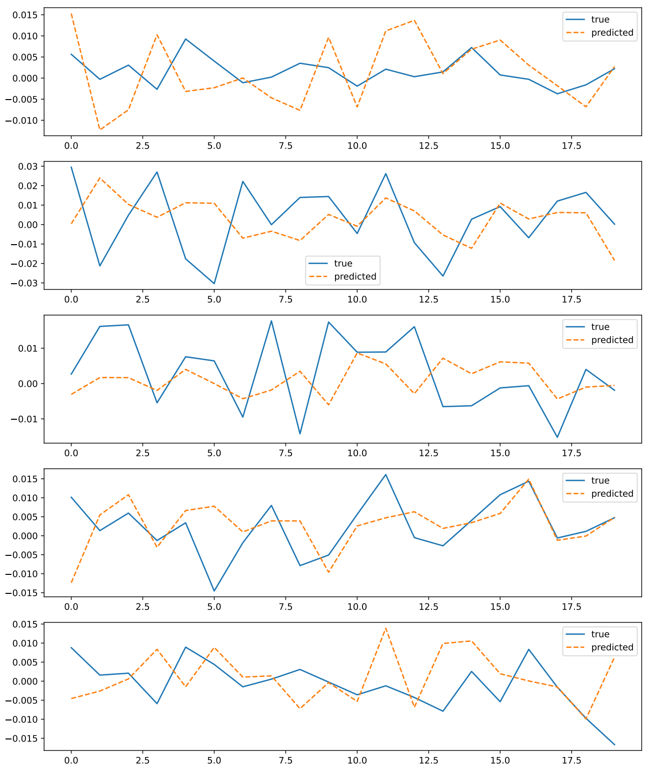


Figure 24: Kernel-PCA LSTM predictions on testing data.

4.2.5 Improving the model

Notes (remove) - anomaly removal - expanding window time-series cross-validation

As discussed in section (4.2.3), the model can accurately predict the first couple of observations after the training data, but the performance gets progressively worse as the model predicts observations from the test set that are further away in time from the training set. In other words, the global model can't generalize to unseen data. A possible solution to this could be to use an expanding window timeseries training to train and cross-validate model performance, and then further train the model as new data becomes available. Another solution would be to implement a rolling window model, which is trained repeatedly on a constant length time-window to predict the next few observations.

4.3 Transformer model

The transformer model is a powerful network architecture that processes sequences by utilizing attention mechanisms. The model consists of stacked encoder and decoder layers equipped with a multi-head self-attention layer and a feedforward neural network. Residual connections and layer normalization are used to stabilize training and positional encoding is used to retain positional information of the inputs. (Vaswani et al. (2023)). The architecture enables the model to capture long-range dependencies (Vaswani et al. (2023)), making it a good candidate for our time-series forecasting task.

4.3.1 Model implementation

We used a transformer model optimized for time series prediction tasks, which predicts 1 future observation from a sequence of 50 past observations. The model is configured with 2 encoder and decoder layers of size 128 and uses lagged features with 1 lag.

4.3.2 Model training

The dataset was split to a test set with the last 1000 observations and training set with the rest of the data. Model was trained over 10 epochs, while model loss on both training set and test set was recorded. Model loss during training can be seen from figure (25).

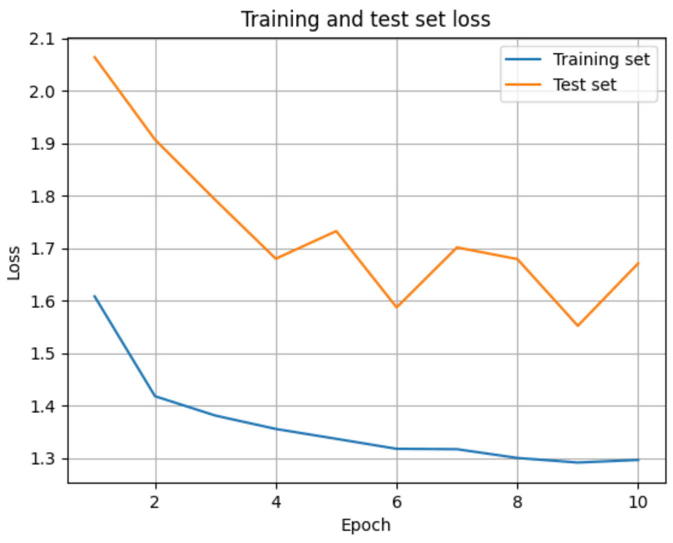


Figure 25: Kernel-PCA LSTM predictions on testing data.

4.3.3 Results

Model predictions on a time window of 1000 observations can be seen in figure (26). The model seems to be able to predict some of the variation in the data, but suffers from underfitting.

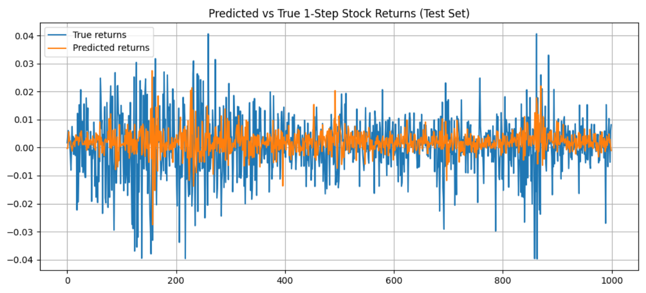


Figure 26: Kernel-PCA LSTM predictions on testing data.

A zoomed in version of the predictions can be seen in figure (27). As in figure (26), the model seems to predict some of the variation, but underfits the data.

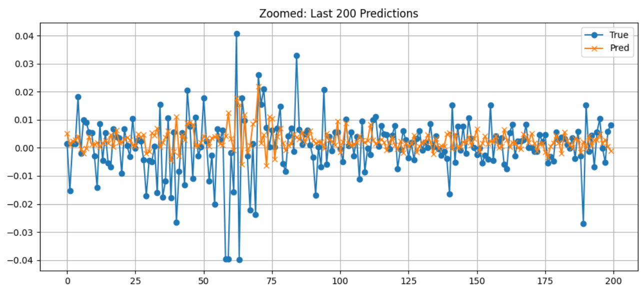


Figure 27: Kernel-PCA LSTM predictions on testing data.

4.3.4 Plan for model optimization

As it can be seen from the results above, the prediction performances of the models are not good, meaning that there truly exists a need for a optimization and tuning of the models. Luckily, there is many different ways to optimize neural networks since the neural networks' architectures are quite complex and the structures are large. In this section we explain some approach for a neural networks' optimization.

Especially with LSTM and RNN models we had problems with overfitting to the data since during the training of the models, the validation losses were pretty high when compared to the training losses. For such problems, one possible solution would be to add dropout layers to the models, where the layer removes some learned information by replacing some values with zeros.

Another method to enhance the performance of the models, would be to rethink the sizes of the prediction horizon and number of time points used for predictions. When considering the application area, forward returns of the stock, it may be that the returns of the stock for example three months ago does not contain any information about tomorrows returns. If that is true, there is no need to use for example 100 time points long sequences for predictions. Also whether we should predict one or ten days ahead is another area to be rethought.

By increasing the size and complexity of the models, enhanced performances could be also achieved. For example with the transformer model, numbers of decoder and encoder layers could be increased. Moreover, the dimensionality of the model could be also increased to 256 or even up to 1024 if required. Also by combining the transformer with another predicting neural network in a way that the output of the transformer would be given as an input to another more simpler neural network could be one possible approach.

References

Aggarwal, C. C. (2023). *Neural networks and deep learning : a textbook.* Springer, Cham, second edition edition.

Cooijmans, T., Ballas, N., Laurent, C., Çağlar Gülçehre, and Courville, A. (2017). Recurrent batch normalization. *arXiv.org.*

Hewamalage, H., Bergmeir, C., and Bandara, K. (2021). Recurrent neural networks for time series forecasting: Current status and future directions. *International journal of forecasting*, 37(1):388–427.

Hull, B., Bakosova, P., Lanteigne, L., Shah, A., Sinclair, E. C., Fast, P., Raj, W., Janecek, H., Dane, S., and Howard, A. (2025). Hull tactical - market prediction. https://kaggle.com/competitions/hull-tactical-market-prediction. Kaggle.

Jaiswal, R., Jha, G. K., Kumar, R. R., and Choudhary, K. (2025). Stl-lstm hybrid model for forecasting seasonal agricultural price series. *Annals of data science.*

Li, X., Petropoulos, F., and Kang, Y. (2023). Improving forecasting by subsampling seasonal time series. *International journal of production research*, 61(3):976–992.

Rosipal, R., Girolami, M., Trejo, L. J., and Cichocki, A. (2001). Kernel pca for feature extraction and de-noising in nonlinear regression. *Neural computing applications*, 10(3):231–243.

Salem, N. and Hussein, S. (2019). Data dimensional reduction and principal components analysis. *Procedia Computer Science*, 163:292–299. 16th Learning and Technology Conference 2019Artificial Intelligence and Machine Learning: Embedding the Intelligence.

Tashman, L. J. (2000). Out-of-sample tests of forecasting accuracy: an analysis and review. *International journal of forecasting*, 16(4):437–450.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2023). Attention is all you need.