STEVENS
INSTITUTE of TECHNOLOGY
THE INNOVATION UNIVERSITY

Machine learning to forecast time series in quantitative wealth and investment management

Group members:
Zhiheng Wu
Shuang Liu
Chengtao Li

Semester: 2021 SPRING

Table of Contents

Abstract

The target of our project is forecasting time series using machine learning methods from GluonTS and then evaluate the performance through metrics. Machine learning algorithms have achieved remarkable success in various areas over recent years. And the strength of Machine learning algorithms, and in fact the requirement for their successful use, is cross-learning, using many series to train a single model. In our project, we focus on: (1) Forecast performance difference between different machine learning models under the condition of same single time series. (2) Forecast performance difference between single time series and multiple time series under the condition of same model. (3) The influence of feature on time series prediction results. And the models selected are DeepAR, Deep State Space, Wavenet, N-BEATS, Deep State and Gaussian process. We find N-BEATS is the best model of forecasting future price of 20 days of 3 Time Series. We also observe the influence of adding Time Series (TS) and Feature.

Keywords: DeepAR, N-BEATS, Wavenet, Deep State Space, Gaussian process,machine learning.

1.Introduction

In the last few decades, machine learning algorithms have achieved remarkable success in various fields. Machine learning algorithms also has a very wide range of applications in the financial field. Compared with humans, machine learning is more efficient in handling business in the financial industry. It can accurately analyze thousands of stocks at the same time and draw conclusions in a short period of time. In terms of stock market trend prediction, machine learning algorithms are used to analyze the company's balance sheet, cash flow statement, other financial data and corporate operating data, extract features related to stock prices or indexes to make predictions. In addition, use enterprise-related third-party consultation, such as policies and regulations, news, or information in social networks, and analyze public opinions or sentiments through natural language processing technology to provide support for stock price forecasts, so that the forecast results are more accurate. Apply supervised learning methods to establish the relationship between two data sets, so as to use one data set to predict the results of another data set, such as using regression to analyze the impact of inflation on the stock market, etc.; unsupervised learning methods can be used in the stock market analysis of influencing factors reveals the main rules behind it; deep learning is suitable for the processing of unstructured large data sets, extracting features that are not easy to display and express; the goal of reinforcement learning is to find strategies to maximize revenue through algorithm exploration. Using methods such as LSTM, stock prices can be predicted in real time based on stock price characteristics and quantifiable market data, which can be used in the high-frequency trading field of the stock market.

In our paper, we will use machine learning method to forecast ETF index, a kind of time series. Actually, it is still difficult to forecast stock prices accurately through machine learning methods. We will apply GluonTS, is a python toolkit for building time series model based on deep learning, to work on it. With GluonTS, we can train and evaluate any of the built-in models on our own data, and quickly come up with a solution for time series tasks. Also, GluonTS allows us to Use the provided abstractions and building blocks to create custom time series models, and rapidly benchmark them against baseline algorithms. Specifically, we will use different kinds of deep learning models via GluonTS to forecast several ETF indexes.

2.Literature Review

Traditionally, time series modeling has focused on individual time series through local models, where free parameters are estimated separately for each time series. There are many commercial and open source toolkits for these local models (Hyndman and Khandakar, 2008; Taylor and Letham, 2018; Lning et al., 2019). We refer to (Januschowski et al., 2019) for a survey of open-source forecasting packages in Python. In recent years, advances in deep learning have led to accuracy improvements over the local approach by utilizing the large amounts of available data for estimating parameters of a single global model over an entire collection of time series (Flunkert et al., 2019; Wen et al., 2017). Deep learning frameworks (Chen et al., 2015; Paszke et al., 2017; Abadi et al., 2016) are growing in popularity, and have led to the emergence of more application-specific toolkits (Hieber et al., 2018; Dai et al., 2018; Bingham et al., 2018).

However, there were no dedicated deep learning toolkit for time series modeling currently exists until the GluonTS appears (Alexander Alexandrov, Konstantinos Benidis, Michael Bohlke-Schneider, Valentin Flunkert, Jan Gasthaus, Tim Januschowski, Danielle C. Maddix, Syama Rangapuram, David Salinas, Jasper Schulz, Lorenzo Stella, Ali Caner T¨urkmen, Yuyang Wang., 2020). GluonTS is a deep learning-based library based on the Gluon API of the MXNet deep learning framework. It provides utilities for loading and iterating over time series datasets, state of the art models ready to be trained, and building blocks to define our own models and quickly experiment with different solutions. GluonTS bundles components can be used to quickly assemble, train, and evaluate new models for time series applications such as forecasting and anomaly detection. It contains a number of deep learning models and probabilistic models for direct use or benchmarking of new algorithms.

3.Data

3.1 Data Source

3.1.1 Bloomberg L.P

Bloomberg L.P. is a privately held financial, software, data, and media company which provides financial software tools and enterprise applications such as analytics and equity trading platform, data services, and news to financial companies and organizations through the Bloomberg Terminal. From the Bloomberg, we downloaded the dataset of various index to train and test models.

3.1.2 Gluonts.dataset

GluonTS comes with several available datasets. And we use the datasets from GluonTS to simply test the efficiency of the models.

3.2 Data description

Data is composed of 10 ETFS, known as MSCI Japan Index (NDDUJN), iShares MSCI North America UCITS ETF (NDDUNA), Amundu MSCI Europe Ex UK Ucits ETF Dr (NDDUE15X), iShares Russell 2000 Growth ETF (RU20GRTR), iShares Russell 1000 Growth ETF (RU10GRTR), iShares MSCI UK ETF (NDDUUK), MSCI World ex USA total net return (NDDUWXUS), S&P 500 Consumer Staples Index (S5COND), S&P 500 Energy Index (S5ENRS) and MSCI Pacific ex Japan UCITS ETF (NDDUPXJ). The data was collected from March 17th, 2016 – April 5th, 2021. In general, a dataset should satisfy some minimum format requirements to be compatible with GluonTS. In particular, it should be an iterable collection of data entries (time series), and each entry should have at least a target field, which contains the actual values of the time series, and a start field, which denotes the starting date of the time series. All the datasets are good proxies for most representative asset and subasset classes so that we can ensure the datasets comprehensiveness. What is more, liquid ETFs were used because liquid data is more suitable for us to train and forecast. And In order to helping account for trends in the data, cyclic behavior, and seasonality, the data was collected at consistent intervals over the period being tracked. Meanwhile, compared to individual stocks or bonds, the datasets selected have “nicer” statistical properties. So, we satisfy the first requirement to use GluonTS.

In total, each sample of our datasets is 1318 daily observations. The data sample was segmented into a training or testing set which ran from March 17th, 2016 – April 5th, 2021. This allowed for a large enough sample (5 years) to see meaningful differential between models. This approach will be explained more in the methodology section.

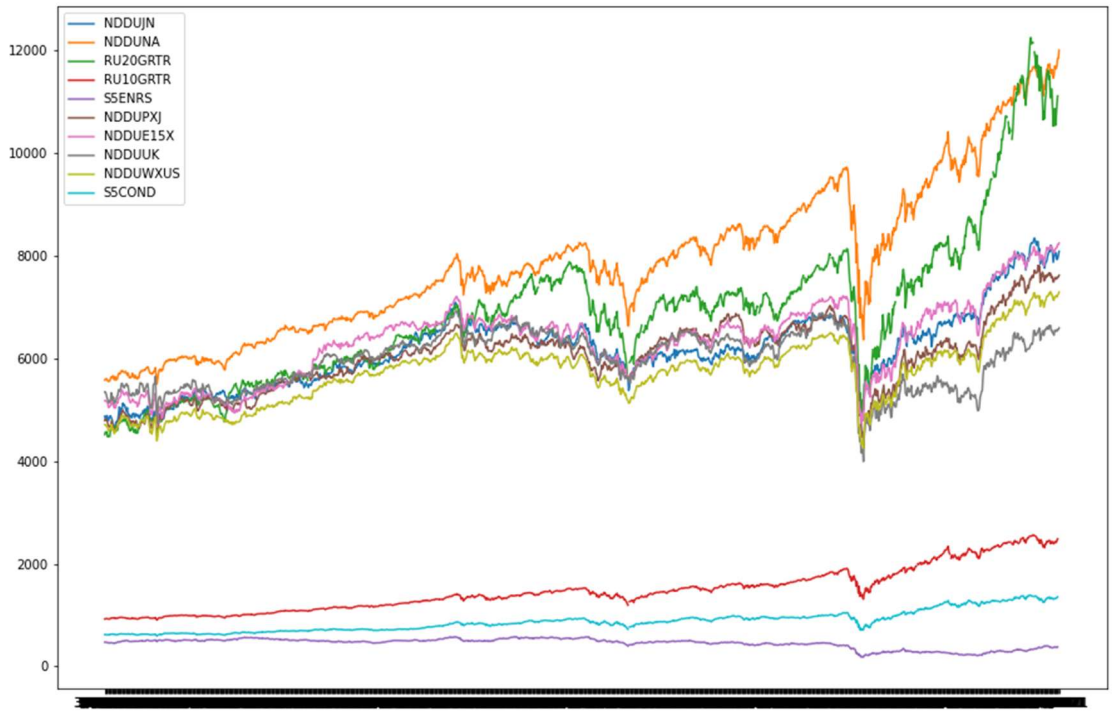


Figure 3.1 Overview of 10 Time Series

4.Methodology

4.1 Gluon Time Series Toolkit (GluonTS)

the Gluon Time Series Toolkit (GluonTS) is a Python library for deep learning-based time series modeling for ubiquitous tasks, such as forecasting and anomaly detection. GluonTS implies the time series modeling pipeline by providing the necessary components and tools for quick model development, efficient experimentation, and evaluation. In addition, it contains reference implementations of state-of-the-art time series models that enable simple benchmarking of new algorithms. GluonTS bundles components such as neural network architectures for sequences, feature processing steps, and experimentation and evaluation mechanisms. These components can be used to quickly assemble, train, and evaluate new models for time series applications, such as forecasting and anomaly detection. In addition, GluonTS includes a number of prebuilt state-of-the-art deep learning-based models and probabilistic models and components, including state-space models and Gaussian processes, for direct use or benchmarking of new algorithms.

4.2 Models

We choose six models to forecast time series from GluonTS. The six models are DeepAR, Deep State Space (DeepState), DeepFactor, Wavenet, Gaussian Process (GP), N-BEATS (NBEATS).

4.2.1 DeepAR

DeepAR used here is a forecasting method based on autoregressive recurrent neural networks, which learns a global model from historical data of all time series in the dataset. The method builds upon previous work on deep learning for time series data, and tailors a similar long short-term memory based recurrent neural network architecture to the probabilistic forecasting problem.

4.2.2 Deep State Space

Deep State Space model here is a novel approach to probabilistic time series forecasting that combines state space models with deep learning. By parametrizing a per-time-series linear state space model with a jointly-learned recurrent neural network, the method retains desired properties of state space models such as data efficiency and interpretability, while making use of the ability to learn complex patterns from raw data offered by deep learning approaches. The method scales

gracefully from regimes where little training data is available to regimes where data from large collection of time series can be leveraged to learn accurate models.

4.2.3 Gaussian Process

Gaussian Process Estimator here shows how to build a local time series model using Gaussian Processes (GP). Each time series has a GP with its own hyper-parameters. For the radial basis function (RBF) Kernel, the learnable hyper-parameters are the amplitude and lengthscale. The periodic kernel has those hyper-parameters with an additional learnable frequency parameter. The RBFKernel is the default, but either kernel can be used by inputting the desired KernelOutput object. The noise sigma in the model is another learnable hyper-parameter for both kernels. These parameters are fit using an Embedding of the integer time series indices (each time series has its set of hyper-parameter that is static in time). The observations are the time series values. In this model, the time features are hour of the day and day of the week.

4.2.4 Deepfactor

Deep Factor model here is a novel global-local method, Deep Factor Models with Random Effects. It is based on a global DNN backbone and local probabilistic graphical models for computational efficiency. The global-local structure extracts complex non-linear patterns globally while capturing individual random effects for each time series locally.

4.2.5 Wavenet

WaveNet here is an audio generative model based on the PixelCNN architecture. To deal with long-range temporal dependencies needed for raw audio generation, architectures are developed based on dilated causal convolutions, which exhibit very large receptive fields.

4.2.6 N-BEATS

N-BEATS model here focus on solving the univariate times series point forecasting problem using deep learning. We propose a deep neural architecture based on backward and forward residual links and a very deep stack of fully-connected layers. The architecture has several desirable properties, being interpretable, applicable without modification to a wide array of target domains, and fast to train. The model tests the proposed architecture on several well-known datasets, including M3, M4 and TOURISM competition datasets containing time series from diverse domains. The model demonstrates state-of-the-art performance for two configurations of N-BEATS for all the datasets, improving forecast accuracy by 11% over a statistical benchmark

and by 3% over last year's winner of the M4 competition, a domain-adjusted hand-crafted hybrid between neural network and statistical time series models.

4.3 forecast progress

4.3.1 Compare the models under different single Time Series (TS).

The first step of forecast progress is comparing the models under different single time series.

For the models in GluonTS, we need to evaluate the performance of the forecast result. As most of the models are local models, we just use single Time series each time for training and testing.

In this part, there are two questions that need answer. First, we need to know what Time series we want to forecast. Second, how we forecast the Time Series. For the first question, the data set provided has Time Series from different market. We choose three different Time Series from different market: NDDUIJN (MSCI Japan index), RU10GRTR (iShare Russell 1000 Growth ETF), E5ENRS (S&P Energy Index). For the second question, we use past 60 days to predict a period of future 20 days. We want to know the performance of the models in GluonTS when we use them to predict long sequence.

In our study, the parameters in the models are initiated randomly. Every time we train and test the model, we have different results. Here are two different methods that we can use. (1) We can set the seed at the beginning of our code. (2) We can train and test one model multiple times without setting seed and get the statistics information. After testing the two methods, we choose the method (2). The number we choose is 50 which is how many times we train and test one model. We also use the method in next two parts.

4.3.2 Adding other Time series in training set.

The second step of forecast progress is comparing the performance between single time series and multiple time series.

Although most of the models we use are local models, the package of GluonTS allows us to add Time series in the training set. In Part 1, when we set the seed at the beginning, running the model multiple times gives us the same result. The parameters are initiated with same values. We set the seed and add Time Series. Then we find that Adding Time Series has influence on the performance. The question is we don't know what the influence is.

The idea that comes to us is correlation. High correlated data may improve the performance. We need to choose one Time Series as the benchmark. And a set of Time Series we need to add. We need get the correlation between the benchmark and other TS first. Then we train the model use to TSs, one is the benchmark and the other is a TS we choose from the data set.

In this part, the models we choose DeepAR and DeepState. The benchmark we choose is NDDUIJN. The set of TSs: NDDUE15X, NDDUNA, NDDUPXJ, NDDUUK, NDDUWXUS, RU10GRTR, RU20GRTR. E5COND, E5ENRS.These TSs will be added to the training set (Each time we just add one Time Series).

4.3.3 adding features in training set.

The final step is comparing performance between single time series with single feature and single time series with multiple features.

GluonTS allows the user to add features in the training set. The features of our data set composed of the price of ETF, Volume of ETF and SMAVG (15) of ETF. In this part, we will group the price and other features to explore the relationship between final performance and features.

The model selected in this part is DeepAR and DeepState which are the models allowing us to add features in the training set. And the benchmark we choose is the performance of single time series with single feature price. The time series are NDDUNA, NDDUPXJ and RU20GRTR.

5.Result and analysis

5.1 Compare the models

5.1.1 The influence of the seed

MSE	23800.068750
abs_error	2425.420898
abs_target_sum	160228.890625
abs_target_mean	8011.444531
seasonal_error	44.179769
MASE	2.744945
MAPE	0.015023
sMAPE	0.015190
OWA	NaN
MSIS	17.521464
QuantileLoss[0.5]	2425.420898
Coverage[0.5]	0.200000
RMSE	154.272709
NRMSE	0.019257
NrD	0.015137
wQuantileLoss[0.5]	0.015137
mean_absolute_QuantileLoss	2425.420898
mean_wQuantileLoss	0.015137
MAE_Coverage	0.300000

(a)The first time

MSE	17430.285938
abs_error	2051.059570
abs_target_sum	160228.890625
abs_target_mean	8011.444531
seasonal_error	44.179769
MASE	2.321266
MAPE	0.012842
sMAPE	0.012746
OWA	NaN
MSIS	19.279600
QuantileLoss[0.5]	2051.059570
Coverage[0.5]	0.650000
RMSE	132.023808
NRMSE	0.016479
ND	0.012801
wQuantileLoss[0.5]	0.012801
mean_absolute_QuantileLoss	2051.059570
mean_wQuantileLoss	0.012801
MAE_Coverage	0.150000

(b)The second time

Figure 5.1 Metrics from running DeepAR twice (No seed).

MSE	57714.575000
abs_error	4291.603516
abs_target_sum	160228.890625
abs_target_mean	8011.444531
seasonal_error	44.179769
MASE	4.291977
MAPE	0.023592
sMAPE	0.023847
OWA	NaN
MSIS	23.225948
QuantileLoss[0.5]	4291.603516
Coverage[0.5]	0.000000
RMSE	237.320501
NRMSE	0.029623
ND	0.026784
wQuantileLoss[0.5]	0.026784
mean_absolute_QuantileLoss	4291.603516
mean_wQuantileLoss	0.026784
MAE_Coverage	0.500000

(a)The first time

MSE	57714.575000
abs_error	4291.603516
abs_target_sum	160228.890625
abs_target_mean	8011.444531
seasonal_error	44.179769
MASE	4.291977
MAPE	0.023592
sMAPE	0.023847
OWA	NaN
MSIS	23.225948
QuantileLoss[0.5]	4291.603516
Coverage[0.5]	0.000000
RMSE	237.320501
NRMSE	0.029623
ND	0.026784
wQuantileLoss[0.5]	0.026784
mean_absolute_QuantileLoss	4291.603516
mean_wQuantileLoss	0.026784
MAE_Coverage	0.500000

(b)The second time

Figure 5.2 Metrics from running DeepAR twice (Use seed).

From **Figure 5.1** we can see, run the same code twice can get two different results. That is just an example. In our testing, we even found the error of one can be 5 times bigger than another. Figure 5.1 also shows that we could just use Mean Absolute Scaled Error (MASE) as the metrics. If MASE is great, other errors (MAPE, RMSE…) will also be great. After we set seed at the beginning of the code, the results can be the same. **Figure 5.2** shows the same two results when setting seed. In other results below, we will use MASE as our metrics.

Table 5.1 Performance (MASE) under different TSs and seeds. Red color is the best performance for a certain TS under different seeds.

Time Series / Seed	NDDUIJN	NDDUE15X	NDDUNA	NDDUPXJ	NDDUUK	NDDUWXUS	RU10FRTR	RU20GRTR	S5COND	S5ENRS
0	11.81208	1.07758	3.17761	3.82566	1.11726	8.36676	7.58775	12.95718	5.92834	9.3238
1	2.60189	1.31704	2.26512	0.92569	2.84145	3.42425	2.61165	5.3008	6.82533	2.23609
2	4.01131	9.62443	6.2318	4.18788	5.48698	2.45768	3.76057	6.66107	9.57303	1.79341
3	2.23718	2.93582	3.75692	2.13096	3.24371	2.10063	10.55811	6.37126	4.30953	7.79028
4	1.91306	1.9791	2.00285	7.75887	5.53524	3.80752	3.63587	13.4148	14.38263	2.4752
5	4.76836	2.66044	2.49196	1.09719	6.18399	1.91753	14.67527	7.61816	8.3455	3.38772
6	2.06983	1.36968	7.44527	3.87265	1.47225	6.03594	5.41067	7.01076	16.53943	1.97454
7	2.45726	10.24368	8.19486	2.14247	1.96828	3.95246	2.62503	13.15071	4.44147	2.02582
8	5.95886	6.42957	1.93229	2.59197	3.00189	4.28946	2.57875	12.21337	8.63674	4.69626
9	2.68174	3.06614	2.81373	2.58644	1.81136	2.99387	11.65346	23.39406	9.04888	2.25472

In **Table 5.1**, the best seed for NDDUIJN is 4 and the best seed for NDDUE15X is 0. The best seed of different TS is different. It is difficult and impracticable to find the best seed for each TS. As a result, setting the seed at the beginning is not a good way to solve our problem. To make the evaluation of the performance more precisely, we need the statistics information. All results after this are using the method that sampling MASE multiple times without setting seed. In our study, the model will be trained 50 times. Each time we will have a MASE. Then we have 50 MASE to learn the statistic information.

5.1.2 Performance of predicting NDDUJN

After solving the problem, we use TS to evaluate the performance of 6 models. As we have mentioned before, these 6 models are local models. It uses one TS to train the model at a time. So, using one TS to test the model is inappropriate. We choose three TSs that represent MSCI, iShare Russel and S&P. We will judge the performance of the model based on the comprehensive performance of each model under these three TSs. The first TS we use is NDDUJN

The error of Deepfactor is very big. In **Figure 5.3**, the minimum MASE of DeepFactor is about 20 when other models' maximum MASE are much smaller than 20. It is the worst model that we use to predict NDDUJN. We need to remove DeepFactor to find the best model.

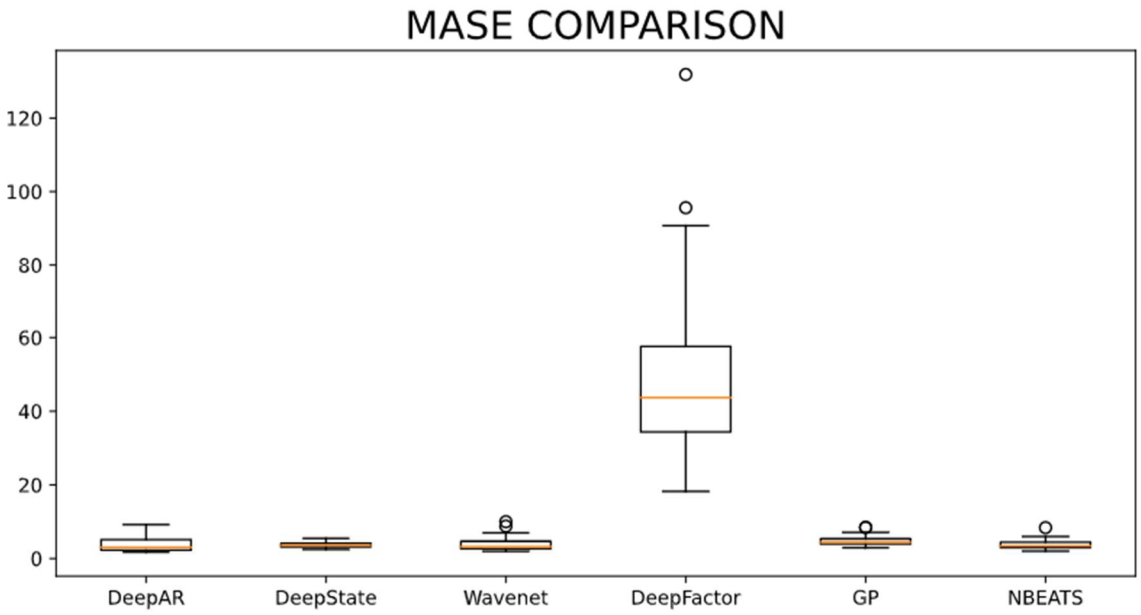


Figure 5.3 Comparison of MASE from different models.

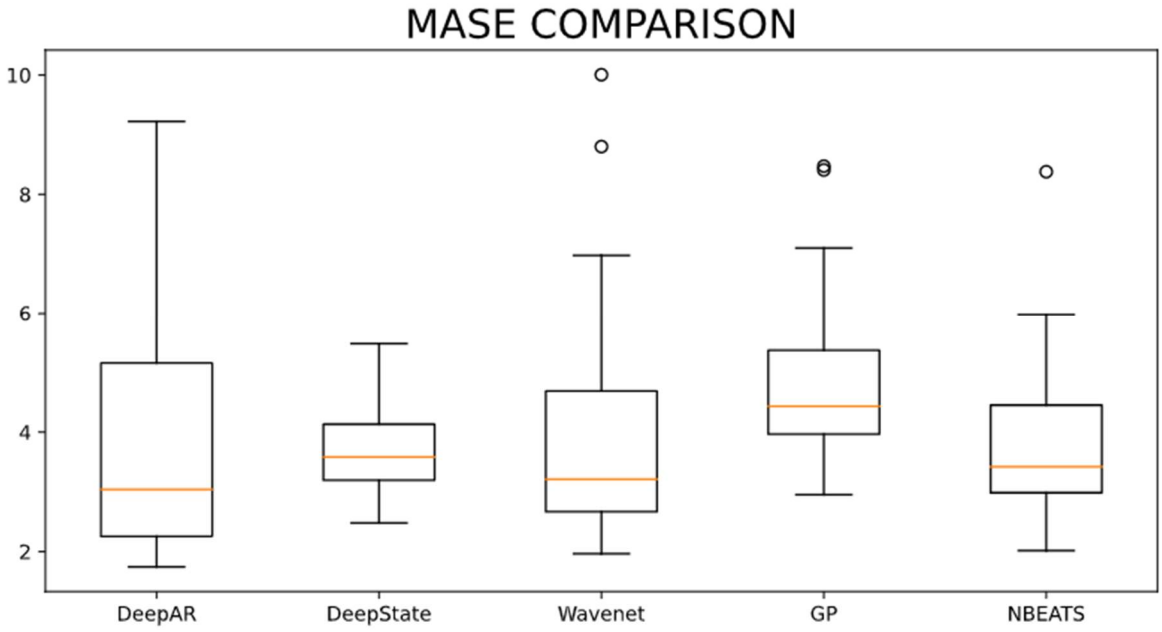


Figure 5.4 Comparison of MASE from different models. This removes DeepFactor

Table 5.2 The mean and median of MASE

Model	DeepAR	DeepState	Wavenet	DeepFactor	GP	NBEATS
Median	**3.0455**	3.5828	3.2106	43.6913	4.3254	3.4197
Mean	3.8782	5.0194	3.953	54.1514	4.7939	**3.8413**

After remove DeepFactor. We can compare the results of other 5 models. DeepAR has the smallest median as we can see in **Table 5.2**. Although NBEATS has the smallest mean values, the existence of outliers makes us to use median as metrics. Median of NBEATS and Wavenet are also close to DeepAR. So NBEATS and Wavenet are also very good models when predicting NDDUJN. The MASE of DeepState is more stable than other models as shown in **Figure 5.4**. But the median of DeepState is too high to make it the best model. As a result, the best model for the prediction of NDDUJN is DeepAR.

5.1.3 Performance of predicting RU10GRTR

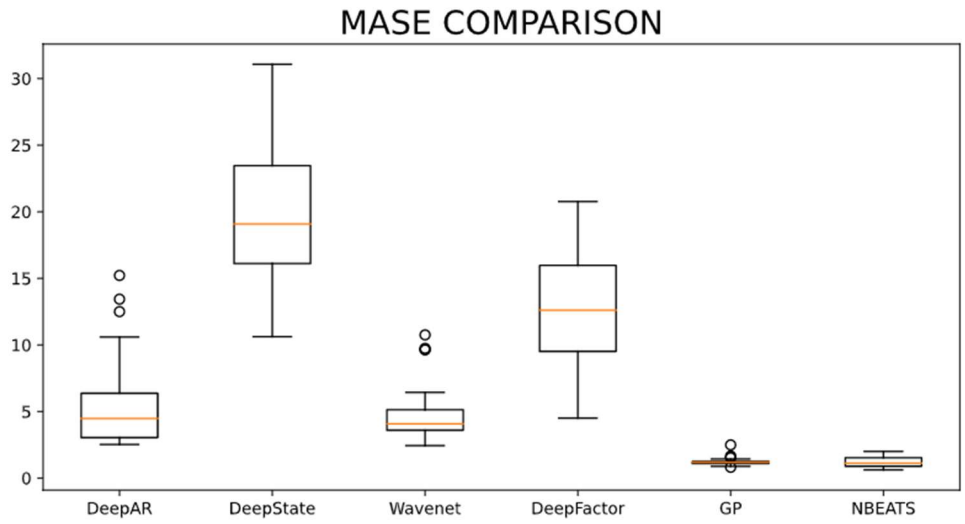


Figure 5.5 Comparison of MASE from different models.

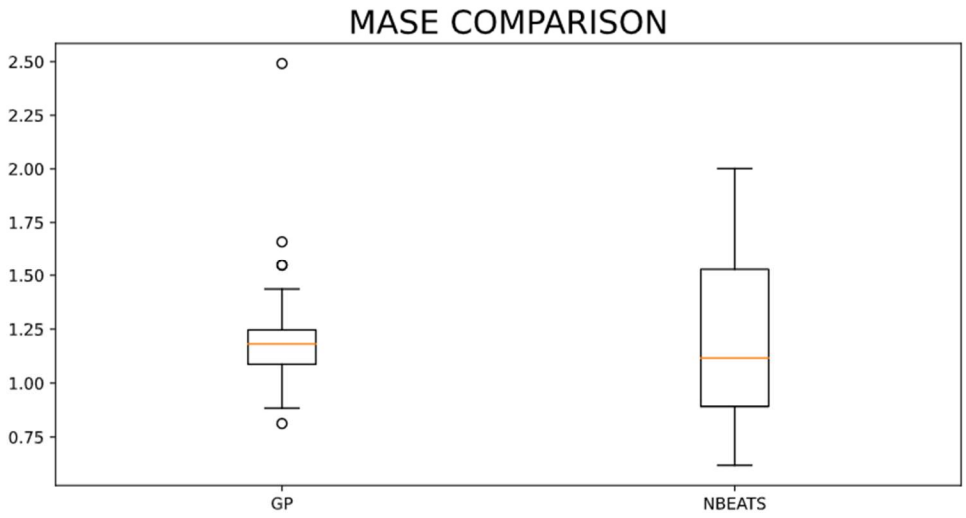


Figure 5.6 Comparison of MASE from different models. Only plot GP and NBEATS

Table 5.3 The mean and median of MASE

Model	DeepAR	DeepState	Wavenet	DeepFactor	GP	NBEATS
Median	4.4746	19.0809	4.0731	12.6056	1.1816	**1.1161**
Mean	5.2851	19.5809	4.8384	12.7823	1.2325	**1.1796**

Figure 5.5 shows a very different result. The simple model GP seems to have a very good performance on predicting RU10GRTR. DeepAR, DeepState, Wavenet and DeepFactor performs poorly. We need remove these four models to compare GP and NBEATS.

In **Figure 5.6**, on the one hand, the MASE of GP is more stable than NBEATS if we ignore outliers. On the other hand, GP has outliers but NBEATS does not have. In **Table 5.3**, NBEATS has the smallest median and mean of MASE. In our opinion, the best model for predicting RU10GRTR is NBEATS. GP is also a very effective model considering the outliers are still lower than 2.5. It depends on what we want to predict and how much risk we want to take to choose the best models.

5.1.4 Performance of predicting S5ENRS

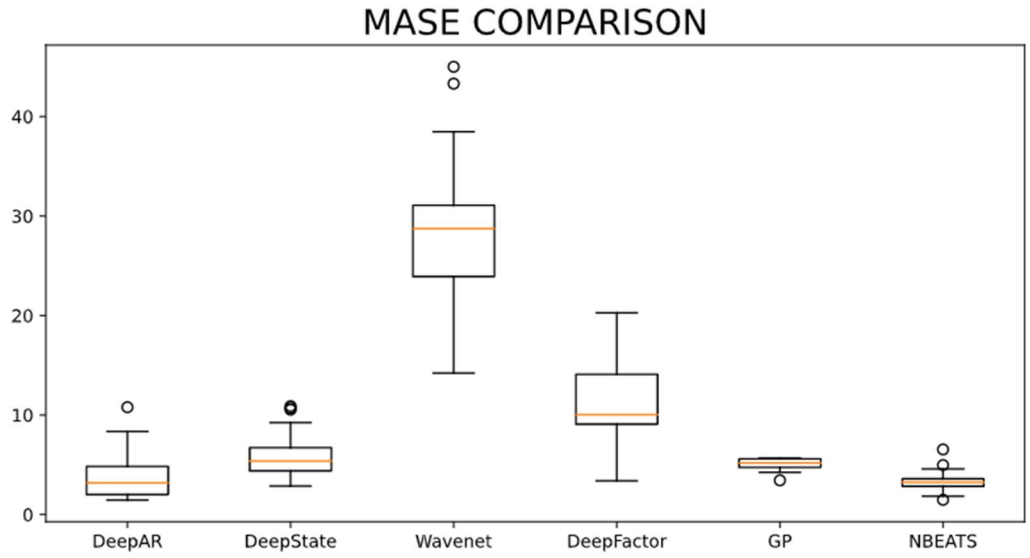


Figure 5.7 Comparison of MASE from different models.

We can see the two models Wavenet and DeepFactor are much worse than other four models from **Figure 5.7**. To compare other four models, we need remove Wavenet and DeepFactor from the plot.

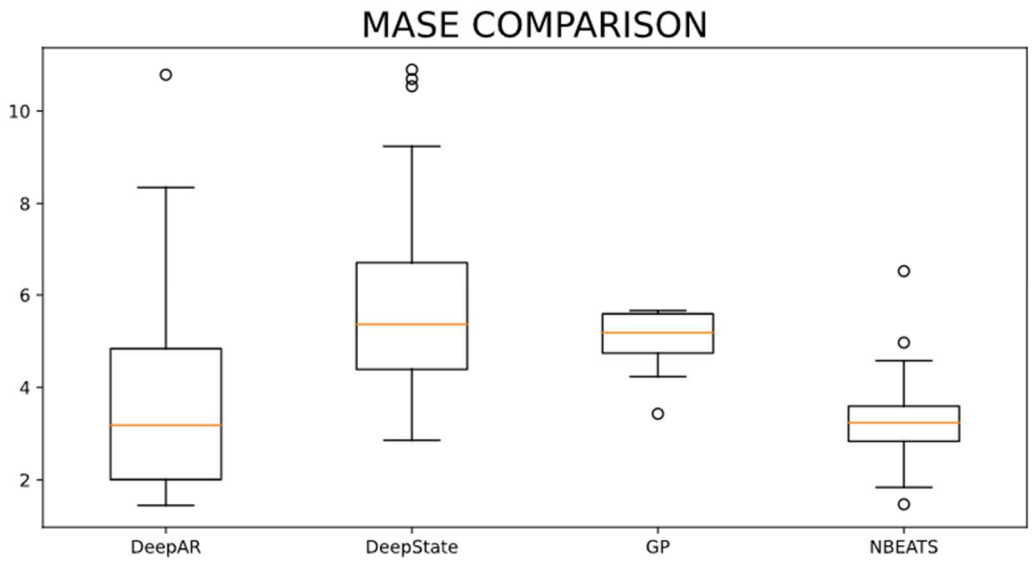


Figure 5.8 Comparison of MASE from different models. Wavenet and DeepFactor are removed.

Table 5.4 The mean and median of MASE

Model	DeepAR	DeepState	Wavenet	DeepFactor	GP	NBEATS
Median	**3.1818**	5.3647	28.7323	10.0324	5.1833	3.2364
Mean	3.6981	5.7289	27.7794	10.8059	5.0969	**3.3638**

From **Figure 5.8** we can see, model NABATS provide a very stable MASE. In **Table 5.4**, NBEATS has the second smallest median of MASE. The value of the median is 3.2364 which is also very close to the smallest one 3.1818. DeepAR has the smallest median but it has great chance to get a big error of prediction. MASE from GP is stable regardless of its value. If we want to use predicting model on S5ENRS, the NBEATS is the best choice.

Table 5.5 Best model of each TS

Time Series	NDDUJN	RU10GRTR	S5ENRS
Best Model	**DeepAR**	**NBEATS**	**NBEATS**
Good models	**NBEATS** **DeepState**	**GP**	**DeepAR**

As we mentioned before, NBEATS performs well on NDDUJN. Based on the performance on the three TS, NBEATS is the best model for predicting TS. The only problem for NBEATS is its cost of time to train the model.

5.2 Influence of adding TS

5.2.1 Using DeepAR and NDDUJN

The first model we choose is DeepAR. In **5.1**, DeepAR gives us a good performance on forecasting one TS. Another reason for choosing this model is the cost of time. It only takes only 20% of the time that be taken to train NBEATS. The correlation we use is Pearson product-moment correlation coefficient.

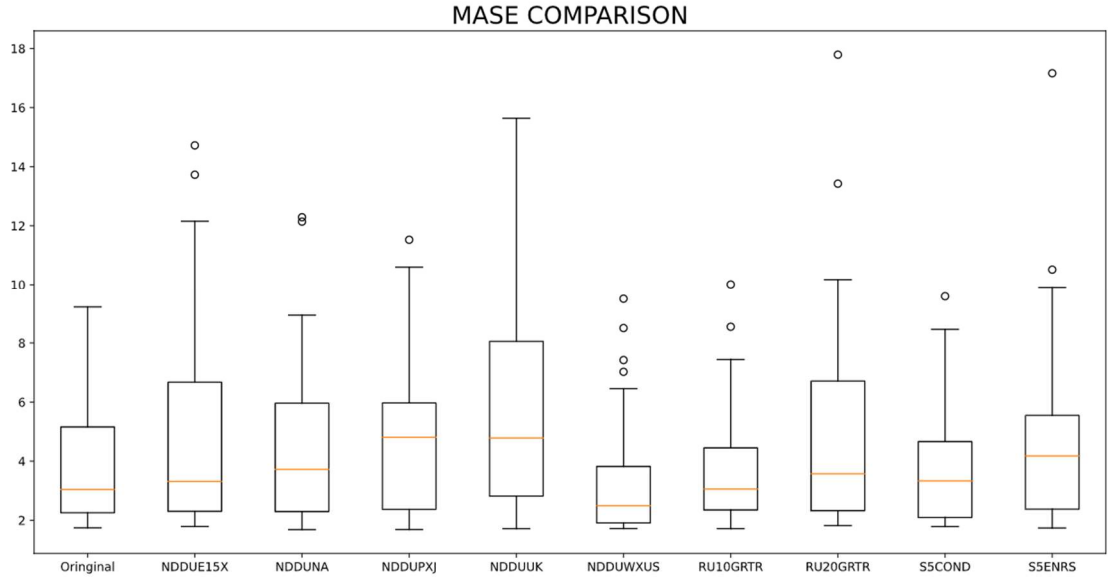


Figure 5.9 Boxplot of MASE that adding different TS. "Original" is the benchmark that only uses NDDUJN in training set. Other names represent the TS we add to the training set.

Table 5.6 Correlations between TSs and NDDUJN and median MASE. "No" is no TS adding.

	NDDUE15X	NDDUNA	NDDUPXJ	NDDUUK	NDDUWXUS
Correlation	0.9601	0.9113	0.9178	0.6040	0.9677
Median	3.3154	3.7215	4.8077	4.7838	2.4913

	RU10GRTR	RU20GRTR	S5COND	S5ENRS	No
Correlation	0.7988	0.7252	0.8066	0.4039	1.0000
Median	3.0579	3.5731	3.3329	4.1776	3.04

In **Figure 5.9**, it is obviously that adding NDDUWXUS has the best performance. In **Table 5.6**, what we can know is that adding NDDUWXUS decrease the median. When not adding any TS, the median is 3.0400. After adding NDDUWXUS, the median decrease to 2.4913. **Table 5.6**

also shows that the correlation between NDDUWXUS and NDDUJN is the highest: 0.9677. Other TSs with lower correlation decrease the accuracy.

What we learn from the result directly is when using DeepAR to forecast NDDUJN, we can add NDDUWXUS to the training set to improve the accuracy. We can also propose two hypotheses based on the results. (1) We can find some TS to improve the performance when forecasting one certain TS. (2) There is relationship between correlation and the influence. For the first hypothesis, we think some TSs can help the model to learning knowledge of other TS. But it may be very difficult to find the right TS that can improve the accuracy. For the second one, the TSs we have used is not enough to draw a conclusion. One way to test the hypothesis is using hundreds of TSs. Then set correlation as the X axis and median as the Y axis to see if there is relationship.

5.2.2 Using DeepState and NDDUJN

Another model we use is DeepState.

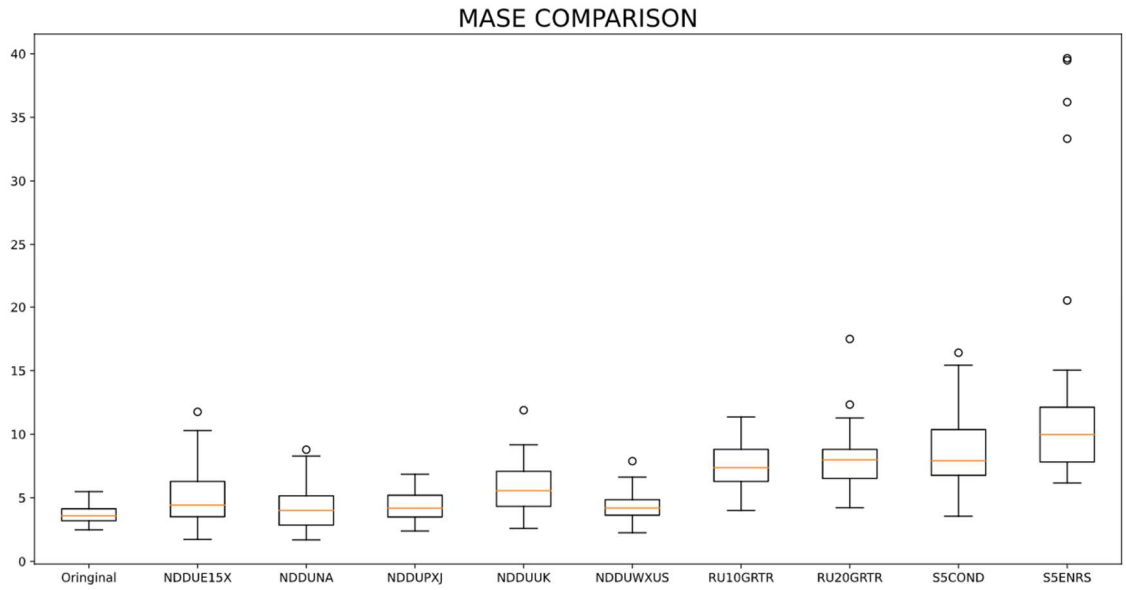


Figure 5.10 MASE of adding different TS. "Original" is the one that only use the benchmark. Other names represent the TS we add to the training set.

Figure 5.10 tells us adding RU10GRTR, RU20GRT, S5COND and S5ENRS is not a good idea. These four TS increase the error of forecasting greatly. We need to remove the four TS to see the performance of adding other TS clearly.

Table 5.7 Correlations between TSs and NDDUJN and median of MASE. "No" is no TS adding.

	NDDUE15X	NDDUNA	NDDUPXJ	NDDUUK	NDDUWXUS
Correlation	0.9601	0.9113	0.9178	0.6040	0.9677
Median	4.4239	4.0058	4.1766	5.5552	4.189

	RU10GRTR	RU20GRTR	S5COND	S5ENRS	NO
Correlation	0.7988	0.7252	0.8066	-0.4039	1.0000
Median	7.365	7.9807	7.908	9.9754	3.5828

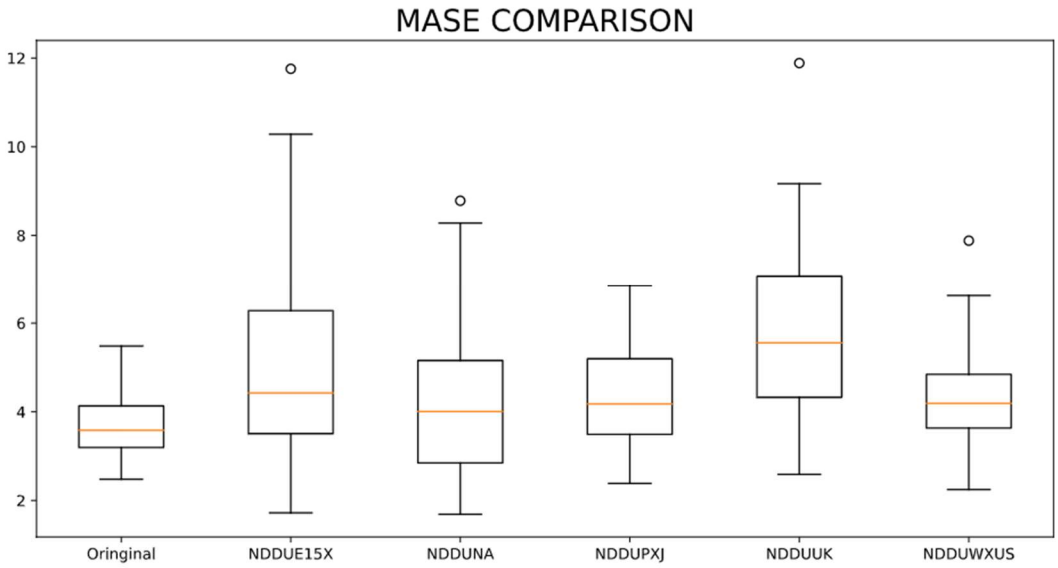


Figure 5.11 The boxplot from **Figure 5.10** but remove RU10GRTR, RU20GRTR, S5COND, S5ENRS.

The results of DeepState are very different from the results of DeepAR. In **Figure 5.11**, adding TS makes the forecasting unstable. When adding TS, the forecasting starts to have outliers. The **Table 5.7** also shows us the case adding no TS has the smallest median of MASE. None of the TS can help improve the accuracy of the forecasting. If we want to use DeepState to forecast TS, there is no need to add other TS. But we still have interesting observation.

We can focus on the results from adding TS. Those four TSs we removed from **Figure 5.10** are from Russel and S&P while NDDUJN is from MSCI. But the remaining 5 TSs are from the same market as NDDUJN. The correlations between the four TSs and NDDUJN are 0.7988, 0.7252, 0.8066 and -0.4039. These are lower than the correlations between NDDUE15X, NDDUNA, NDDUPXJ, NDDUWXUS and NDDUJN. And in **Table 5.7** we can see, those with higher

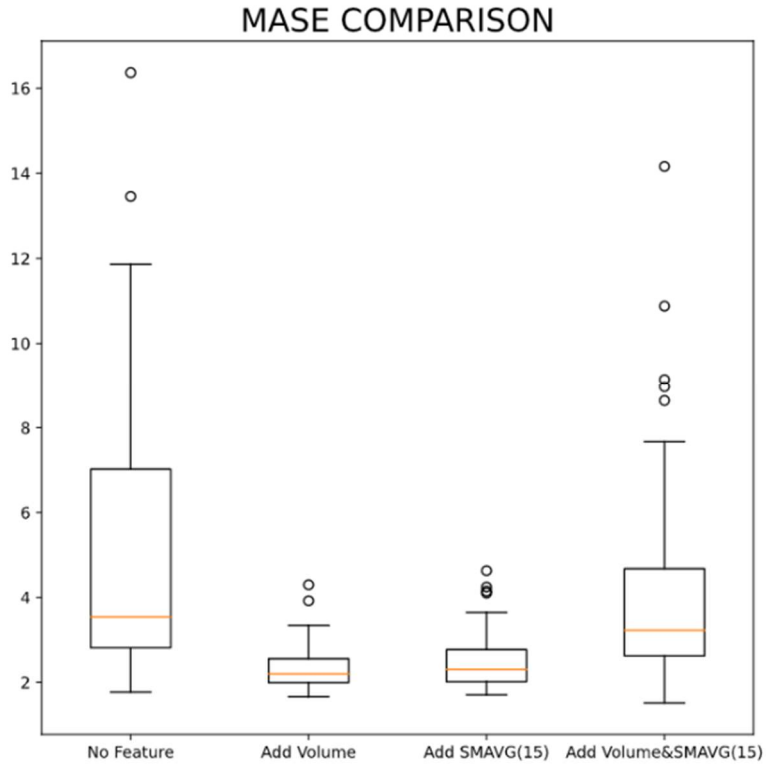
correlations have smaller medians. The result from this model gives us evidence on the second hypothesis in **5.2.1**.

5.3 Influence of features

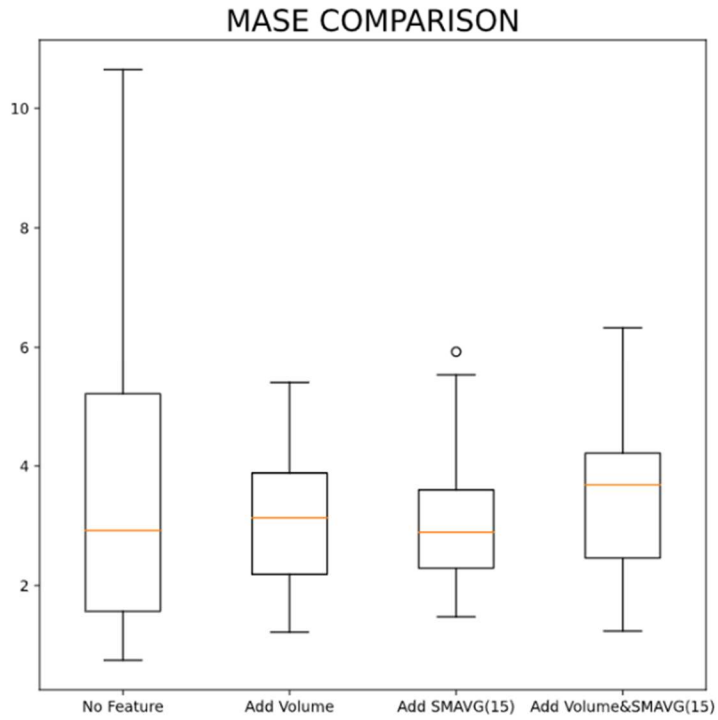
In this part, we used DeepAR and Deepstate to test the influence of features. DeepAR and DeepState allow us adding feature part in training set and testing set. The input of DeepAR and DeepState accepts the feature of the TS. Other four models have no such function. The TS we choose is NDDUNA, NDDUPXJ and RU20GRTR. They have two features: Volume and 15 days simple moving average (SMAVG (15)). There are four situation we need to compare: No feature, Adding Volume, Adding SMAVG (15), Adding Volume and SMAVG (15)

5.3.1 Using DeepAR

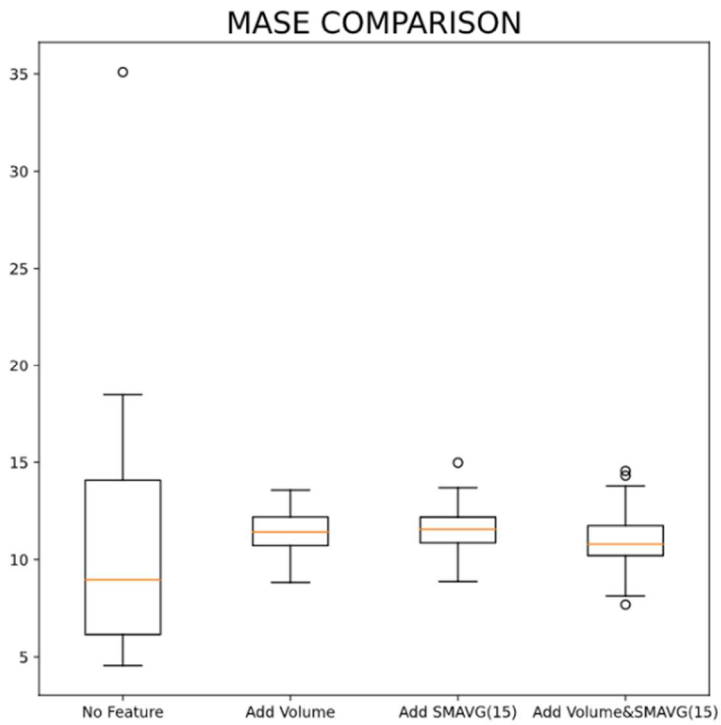
The first model we use is DeepAR.



(a)NDDUNA



(b)NDDUPXJ



(c)RU20GRTR

Figure 5.12 MASE for different situation and TS

Table 5.8 Median of MASE for different situation and TS

	No feature	Adding Volume	Adding SMAVG(15)	Adding Volume and SMAVG(15)
NDDUNA	3.5399	**2.1998**	2.3074	3.2261
NDDUPXJ	2.9186	3.1324	**2.8917**	3.6821
RU20GRTR	**8.9586**	11.4161	11.5543	10.7919

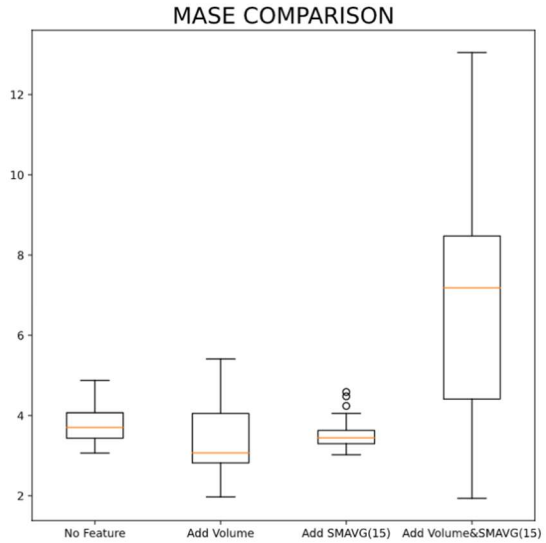
Adding features has remarkable influence on forecasting TS. In **Figure 5.12**, no matter what features we add, the forecasting becomes more stable especially for NDDUNA and RU20GRTR. But in **Table 5.8**, adding features may not improve the overall performance of forecasting. For NDDUNA, the median of MASE is 3.5399 and adding one feature decrease it to 2.1998 and 2.3074. But for RU20GRTR, adding features increase the median of MASE from 8.9586 to about 11.5, although the results are stabilized.

Also in **Table 5.8**, adding two features is worse than adding one features when using NDDUNA and NDDUPXJ. It is better when using RU20GRTR. When we check the data of NDDUNA and NDDUPXJ, we find that there are too many missing values under Volume and SMAVG (15). This may be the reason that cause poor performance of adding two features.

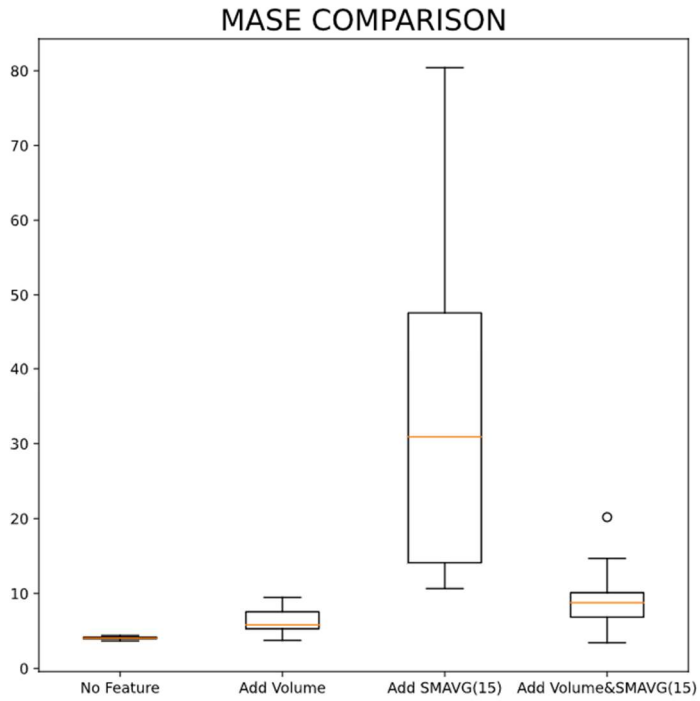
Although adding features may increase the error, the error is increased by a small percentage. As a result, adding features is a good way to improve the forecasting when using DeepAR.

5.3.1 Using DeepState

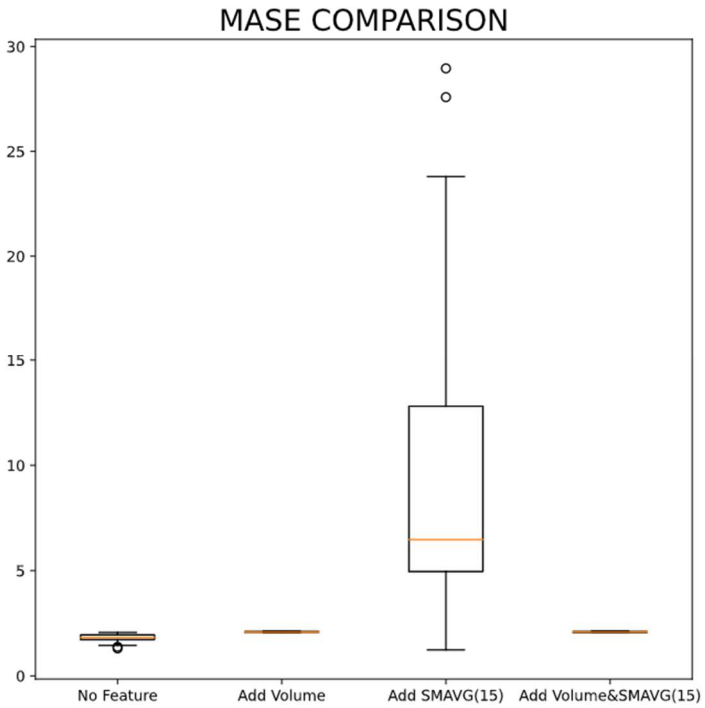
The second model is DeepState.



(a)NDDUNA



(b)NDDUPXJ



(c)RU20GRTR

Figure 5.13 MASE for different situation and TS

Table 5.9 Median of MASE for different situation and TS

	No feature	Adding Volume	Adding SMAVG(15)	Adding Volume and SMAVG(15)
NDDUNA	3.7046	**3.0691**	3.4443	7.183
NDDUPXJ	**4.0508**	5.8118	30.9075	8.7715
RU20GRTR	**1.8221**	2.0803	6.471	2.0830

Figure 5.13 and **Table 5.9** show a very different result from DeepAR. For NDDUNA, adding one feature improve the performance. But adding SMAVG (15) can be a disaster for NDDUPXJ and RU20GRTR. Unlike DeepAR, there is no clue that adding features can stabilize the forecasting of DeepState. This can be observed in Figure 5.13(a) and (b). In **Table 5.9** we can see, adding two features is not a good idea. This is the same as DeepAR.

In summary, adding features is not recommended when using DeepState.

6.Conclusions

Forecast TS is very difficult, especially forecast TS in financial market. Some models give the probabilistic prediction like DeepAR. The prediction has confidence interval. The question is, even the model learns the distribution perfectly, it does not know what will happen certainly. Also, forecasting a long period of time makes it harder. The package GluonTS provide us pre-build models that can be used easily. We can just use few lines of code to train and test the model. But it takes time to figure out how to use the package more efficiently.

There are many things that have impact on the forecasting of model. The first thing is the seed. The models in GluonTS initiate the value of the parameters randomly. DeepAR use uniform random number to initiate the weights in each net. It causes a very big problem in forecasting: the result of the training differs every time when use same TS and structure of model to train. The classic way to solve the problem is getting statistics information. In our study, the model has been trained 50 times. But this number is still not enough to get a precise result. The expected number is 1000 or more. The reason why we do not use it is the cost of time. It will take months to finish all the problems. For example, NBEATS takes 20 minutes to train and forecast at a time. When we train and forecast it 1000 times, it cost us 20000 minutes which is about two weeks. This is just for one TS.

The best model of the six models we use is NBEATS. This is based on the performance under three different TS. It is reasonable because of its structure and cost of time. DeepAR is also a usable model compared to other four. What we are forecasting is a long period of time like few days. Unlike high frequency trading, we do not need to get most recent result when we use daily data. The time it takes to train NBEATS can be ignored. Other models also take minutes each time.

Another thing that may change the performance is adding other TS. The models we compare is local models, which means it accepts only one TS as the input. But the package allows us to add other TSs as the extension in the training set. The result shows us adding TS can improve the performance of DeepAR. It also shows that the existence of some relationship between correlation and the influence. In future study of this, we need a big set of TSs. The correlations have a wider range just like [-1,1].

Feature plays a very important part when we use Machine learning model. Many Deep Learning model use features to improve the accuracy. It gives the model more information about

the data. When we use feature in DeepAR, the feature helps the model to get a more stable result. It is attractive for those who want a very stable forecasting. Although DeepState can have features in input, it does not show the advantage of adding feature. Different models have different function to capture the feature. As we mentioned in 5.3.1, the data of the feature has missing values. DeepAR may be good at handle this situation while DeepState may be not. We need be careful if we want to add features.

The models have their advantage and disadvantage. It depends on our needs to choose the best one.

References

1. Rangapuram, Syama Sundar, Matthias W. Seeger, Jan Gasthaus, Lorenzo Stella, Yuyang Wang, and Tim Januschowski. "Deep state space models for time series forecasting." *Advances in neural information processing systems* 31 (2018): 7785-7794.
2. Salinas, David, Valentin Flunkert, Jan Gasthaus, and Tim Januschowski. "DeepAR: Probabilistic forecasting with autoregressive recurrent networks." *International Journal of Forecasting* 36, no. 3 (2020): 1181-1191.
3. Oreshkin, Boris N., Dmitri Carpov, Nicolas Chapados, and Yoshua Bengio. "N-BEATS: Neural basis expansion analysis for interpretable time series forecasting." *arXiv preprint arXiv:1905.10437* (2019).
4. Wang, Yuyang, Alex Smola, Danielle Maddix, Jan Gasthaus, Dean Foster, and Tim Januschowski. "Deep factors for forecasting." In *International Conference on Machine Learning*, pp. 6607-6617. PMLR, 2019.
5. Alexandrov, Alexander, Konstantinos Benidis, Michael Bohlke-Schneider, Valentin Flunkert, Jan Gasthaus, Tim Januschowski, Danielle C. Maddix et al. "GluonTS: Probabilistic and Neural Time Series Modeling in Python." *Journal of Machine Learning Research* 21, no. 116 (2020): 1-6.