Time Series Classification

Gabriel Lucchini, Nassim Lattab, Florian Posez, Mohamed Azzaoui

Professor : Themis Palpanas, Lucrezia Tosato

Abstract—This report presents a comprehensive exploration of time series classification in two distinct domains: oil transformer temperature prediction and a multivariate time series task encompassing eight features about exchange rate forecasting with missing value handling. The primary objective of this research is to develop accurate predictive models capable of forecasting the next 100 values in each dataset.

keywords—*Time Series Prediction, Data Science, Oil Temperature, Exchange Rate, Multivariate Time Series, Forecasting Techniques, Missing Data Imputation*

Contents

1. Introduction

Time series forecasting is a classical learning problem that consists of analyzing time series [7] to predict future trends based on historical information. In particular, long-term forecasting [6] is notoriously challenging due to feature correlations and long-term temporal dependencies in time series. This learning problem is prevalent in real-world applications where observations are gathered sequentially, such as medical data, electricity consumption temperatures, or stock prices.

2. Problem Formulation

The problem we aim to address in this project involves two main tasks in time series forecasting.

2.1. First part: Univariate Time Series Forecasting

The first task focuses on training a predictor to forecast the next 100 values of a univariate time series. Specifically, we aim to predict the 'OT' feature, representing the temperature of the oil in a transformer. This feature is critical for cooling and operational efficiency. The dataset for this task consists of a single univariate time series with no missing values.

Univariate time series forecasting [2] has been extensively studied in the literature. Traditional methods include the use of auto-regressive (AR) models, moving averages (MA), or combinations thereof such as ARMA and ARIMA models (Box et al., 2015). These models have been successfully used to predict univariate time series in various domains, such as weather forecasting, product sales, and industrial equipment monitoring.

2.2. Second part: Multivariate Time Series Forecasting

The second task involves forecasting the next 100 values of a daily exchange rate, which is a multivariate time series problem. This task requires preprocessing the data, segmenting the dataset, and handling missing values. The training data for this task consists of one multivariate time series with 8 features. The objective is to forecast the '6'th feature, which represents different currencies or countries' exchange rates relative to a base currency. Additionally, 'OT' in this context represents an additional variable or feature influencing exchange rates, and the dataset contains 25% of missing values.

Forecasting multivariate time series [1] is an active research area, and several approaches have been proposed. These include vector auto-regressive (VAR) models, recurrent neural networks (RNNs), hidden Markov models (HMMs), and deep learning methods such as recurrent neural networks (RNNs) and convolutional neural networks (CNNs). These methods have been successfully applied to forecasting problems in various domains, including financial markets, meteorology, and healthcare.

These two tasks present distinct challenges in time series forecasting and require different approaches for model training and evaluation.

3. Part 1 : Electricity Transform Temperature prediction on time series data

In this part, we will focus on training a predictor to forecast the next 100 values of a univariate time series. Specifically, our goal is to predict the 'OT' feature, representing the temperature of the oil in a transformer. This feature is critical for cooling and operational efficiency. The dataset for this task consists of a single univariate time series with no missing values.

3.1. Data description

The provided dataset, named 'ETTh1', comprises timestamps indicating the date and hour of measurement for the variable 'OT' (Oil Temperature). The 'OT' values are recorded hourly, spanning from July 1, 2016, 00:00:00, to June 22, 2018, 15:00:00. In total, the dataset encompasses 17,320 observations of 'OT'.

Notably, this dataset exhibits complete data integrity, with no missing values, facilitating a comprehensive analysis of oil temperature trends over the specified time period. In Fig 1, we can see an overall representation of 'OT'.

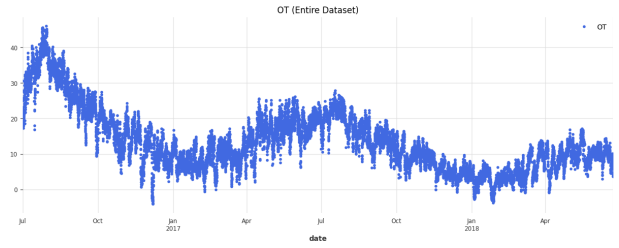

Figure 1. Plot of 'OT'

In the analysis of transformer oil temperature data between 2016 and 2018, it is observed that temperatures recorded during the period 2016-2017 are generally higher than those recorded during the period 2017-2018. This trend may have significant implications in terms of operational efficiency and heat management in transformers.

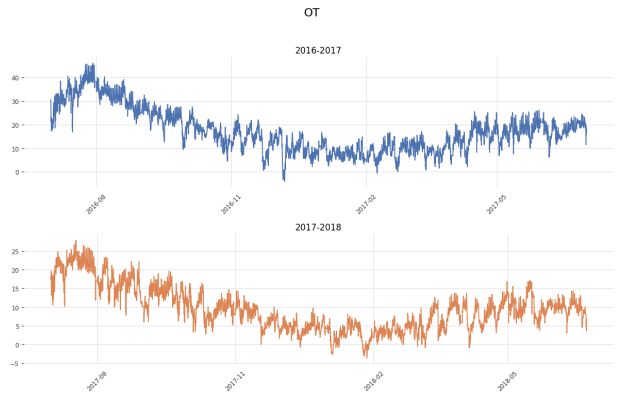

Figure 2. Comparison of curves for the period 2016-2017 and 2017-2018

When examining the data across different months, there seems to be a lack of consistent similarity in their overall structure. Each month exhibits distinct patterns and fluctuations, suggesting unique environmental conditions and influences at play during different times of the year. Interestingly, while the overall structure varies between months, the oil temperature (OT) values demonstrate a notable alignment on a daily and hourly basis.

3.2. Models and implementation

To address the task of predicting the next 100 values of oil temperature ('OT') in the provided dataset ('ETTh1'), we initiated our modeling approach by partitioning the dataset into training, validation, and test sets (see Fig 3). This step ensured an effective evaluation of model performance while avoiding overfitting and assessing generalization capabilities.

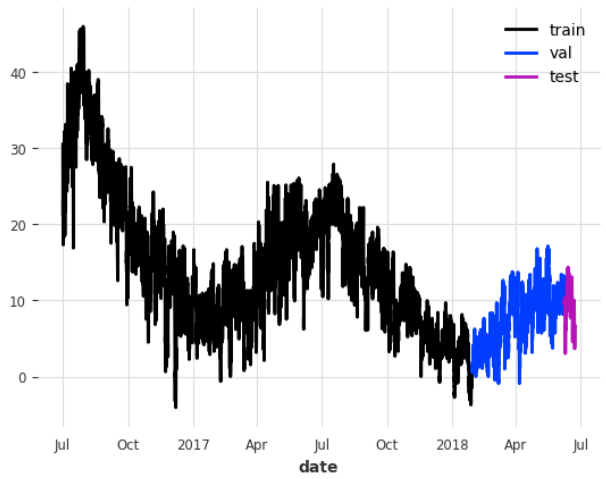

Figure 3. Partitioning of the dataset into training, validation, and test sets

We began by selecting a diverse set of forecasting models known for their efficacy in time series prediction tasks. These included the TFTModel, NBEATSModel, TiDEModel, and TiDE+RinModel. Each model offers unique architectures and capabilities, allowing for a comprehensive exploration of different modeling approaches.

- TFTModel: A Temporal Fusion Transformer model, leveraging self-attention mechanisms for capturing temporal dependencies in the data [5].
- NBEATSModel: A Neural Basis Expansion Analysis Time Series model, renowned for its interpretability and ability to capture complex patterns in time series data [4].
- TiDEModel: A Temporal Inductive Deep Learning model, designed to exploit both spatial and temporal information in time series data [3].
- TiDE+RinModel: A variant of the TiDE model enhanced with Recurrent Integration Network (RIN) components, aimed at capturing long-term dependencies and improving predictive performance.

Once the models were chosen, we conducted hyperparameter tuning (see Fig 1) to optimize their performance. This involved systematically adjusting key hyperparameters, such as input/output chunk lengths, hidden layer sizes, and dropout rates, to find the configurations that yielded the best predictive performance on the validation set.

Table 1. Hyper-parameter tuning range for our 4 models

Parameters	Range
inputChunkLength	[256,512,1024]
outputChunkLength	[8,16,32,64,100,128]
hiddenSize	[256,512,1024]
numEncoderLayers	[1,2,3]
numDecoderLayers	[1,2,3]
decoderOutputDim	[16,32,64]
temporalDecoderHidden	[16,32,64,128]
useLayerNorm	[True,False]
dropout	[0.0,0.1,0.2,0.3,0.5]

See the documentation for each parameters here [8].

Each model was trained and evaluated using the same dataset partitioning strategy, comprising training, validation, and test sets, to ensure consistency and fairness in the comparison. We applied the same hyperparameter tuning and performance evaluation metrics to assess the models' predictive accuracy and generalization capabilities.

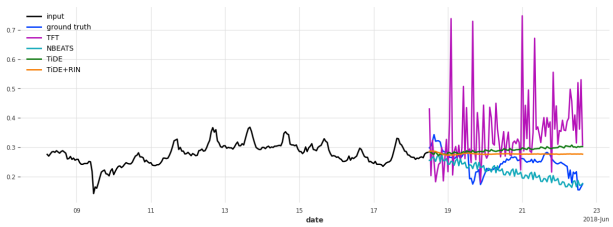

Figure 4. Comparison of the models

Following model training and evaluation, we assessed the Mean Absolute Error (MAE) metric and the Mean Squared Error (MSE) metric to determine the model with the most accurate predictions for the 100 last values available on the given dataset 'ETTh1'. Notably, our analysis revealed that the TiDE+RINModel exhibited the lowest MAE and MSE (see Fig 5), indicating superior predictive performance compared to the other models evaluated.

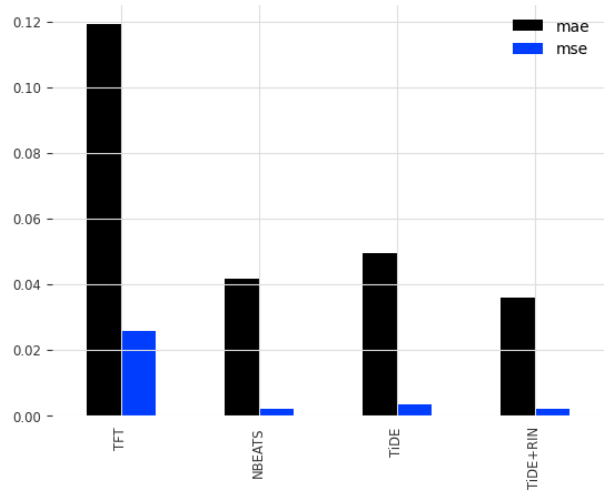

Figure 5. Comparison of the mae and mse of the models

This preliminary testing phase served as a valuable precursor to the final model selection, providing valuable insights into the performance of each model under consideration.

Then, we proceeded with the TiDE+RINModel with specific hyper-parameters (see Table 2) as our selected model for predicting the forthcoming 100 'OT' values, based on its demonstrated efficacy and accuracy in capturing the underlying patterns in the oil temperature time series data.

Table 2. Hyper-parameters for the TiDE+RINModel for Part 1

Parameters	Value
inputChunkLength	512
outputChunkLength	100
hiddenSize	512
numEncoderLayers	2
numDecoderLayers	2
decoderOutputDim	32
temporalDecoderHidden	16
useLayerNorm	True
dropout	0.5

Below, we present a pseudo code outlining the implementation of the TiDE+RINModel. This approach builds upon the foundational concepts of the TiDE model while incorporating Recurrent Integration Network (RIN) components to further enhance its predictive capabilities.

Algorithm 1 Pseudo-code for TiDE+RIN model implementation

Input: Time series data
Output: Predicted values for the next 100 time steps
Split the time series into training, validation, and test sets
Scale the data to normalize the values using a scaler
Initialize the Tide-Rin model with specific parameters
Train the model on the scaled training data
Predict the next 100 values using the trained model
Invert the scaling to obtain the final predictions
Return: Predicted values for the next 100 time steps

The Tide-Rin model implementation is coded in Python, leveraging the TiDEModel class from the tide library for the model architecture. Additionally, the Scaler class from the sklearn.preprocessing module is employed for data normalization. Below, you will find a detailed Python implementation of the Tide-Rin model algorithm:

```
# Definition of a function tide_rin_model taking a time series as input
def tide_rin_model(series):
    # Splitting the time series into training, validation, and test sets
    train, temp = series.split_after(0.8)
    val, test = temp.split_after(0.9)

    # Scaling the data using Scaler() to normalize the values
    scaler = Scaler()  # By default, uses MinMaxScaler from sklearn
    series_scaled = scaler.fit_transform(series)
    train = scaler.fit_transform(train)
    val = scaler.transform(val)
    test = scaler.transform(test)

    # Initializing the TiDE+RIN model (TiDE with reversible instance normalization)
    tideRIN = TiDEModel(
        input_chunk_length=512,
        output_chunk_length=100,
        num_encoder_layers=2,
        num_decoder_layers=2,
        decoder_output_dim=32,
        hidden_size=512,
        temporal_decoder_hidden=16,
        use_layer_norm=True,
        use_reversible_instance_norm=True,
```

```
        dropout=0.5,
    )

    # Training the model on the training data
    tideRIN.fit(
        series=series_scaled,
        epochs=5,
        verbose=True,
    )

    # Predicting the next 100 values using the trained model
    scaled_pred_tideRIN = tideRIN.predict(n=100)
    # Inverting the scaling to obtain the final predictions
    pred_tideRIN = scaler.inverse_transform(scaled_pred_tideRIN)

    # Returning the model predictions
    return pred_tideRIN
```

Code 1. TiDE+RIN model implementation.

3.3. Results

In our training phase of various models, as depicted in Figure 5, the one showing the most promise was Tide+RIN. However, due to potential overfitting, we couldn't be certain of its actual performance until testing it on Kaggle. We found that even on Kaggle, Tide+RIN remained the best among the tested methods.

In this Figure 6 below, we showcase our best Kaggle score achieved thanks to the TIDE+RIN Model for the Part 1.

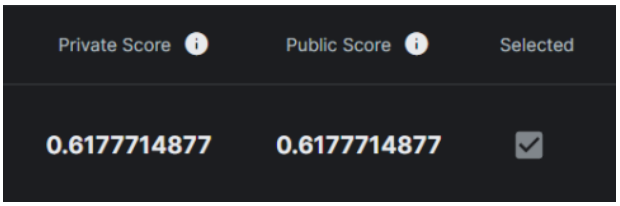

Figure 6. result of Kaggle MAE = 0.6177

The results obtained by this method are rather satisfactory, even though they are slightly distant from our MAE in Figure 5 (due to potential slight overfitting).

Before obtaining these results and using TFT, Tide, etc., we had tested several different methods; here are their various results on Kaggle :

Table 3. Differents methods and their results in Kaggle

Method	Mae in Kaggle
Linear + Polynomial Regression	1.12
random forest	1.44
TensorFlow (using LSTM)	0.80
NeuralProphet	0.89

4. Part 2 : Weather indicators prediction on time series data handling the missing values

In the second part of our project, we delved into the realm of multivariate time series forecasting, focusing on predicting weather indicators while effectively handling missing values. The training dataset comprises a single multivariate time series encapsulating eight distinct features. Our objective centers on forecasting the sixth feature within this dataset, denoted as '6'.

4.1. Data description

Each feature within the dataset holds significance, with features indexed from 0 to 6 representing various currency exchange rates relative to a base currency or the exchange rates of different countries. It's essential to note that within this context, the feature labeled as 'OT' carries a distinct meaning; it does not denote "Oil Temperature" as observed in the ETTh1 dataset. Instead, 'OT' represents an auxiliary variable or feature that influences exchange rates indirectly, potentially encapsulating external factors or market dynamics impacting currency fluctuations.

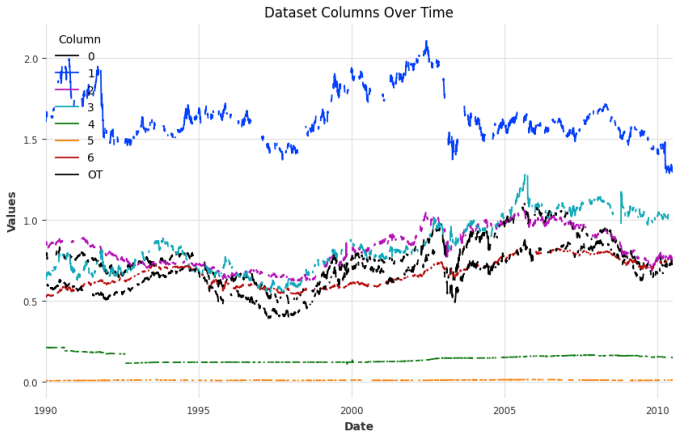

Figure 7. Plot of the 8 features with missing values

In our approach to handling missing values within the dataset, we employed backward filling. This technique facilitates the replacement of missing values by copying the next available observation in the dataset. This strategy ensures that missing values are effectively imputed based on subsequent data points, thereby maintaining the temporal coherence and integrity of the time series.

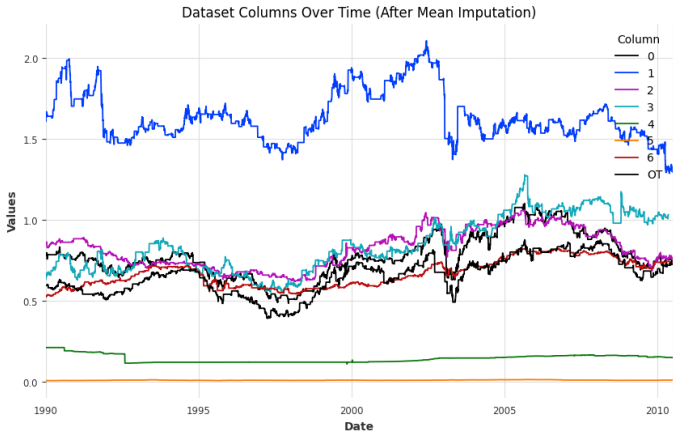

Figure 8. Plot of the 8 features without missing values

In our feature selection process, we have identified the features 0, 2, 6, and 'OT' as the most relevant and closely related

to the target feature ’6’ that we aim to predict. These features have been chosen based on their potential correlations and perceived significance in influencing the dynamics of the target variable, as well as their proximity and apparent relevance within the context of the dataset.

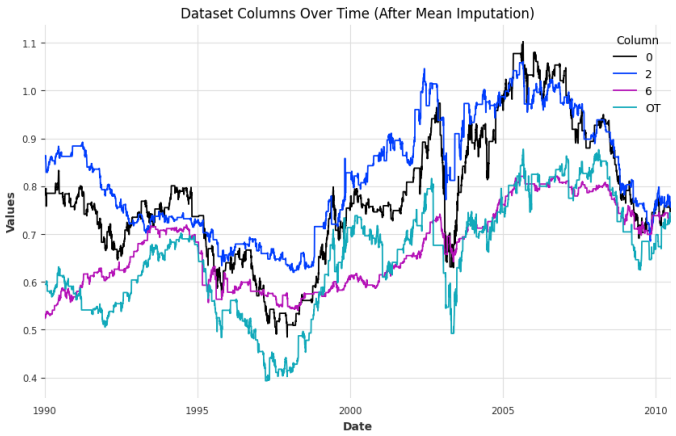

Figure 9. Plot the 4 important features

Before resorting to backward filling, we explored leveraging the correlation between different features in the dataset. We aimed to identify the most correlated feature with the target feature ’6’ and utilize linear regression to fill in the missing values.

We iteratively searched for features with the highest correlation to ’6’, restarting the process as long as a strong correlation was observed. This was visually assessed by examining the linear dispersion of points between the target feature ’6’ on the ordinate axis and the most correlated feature on the abscissa axis.

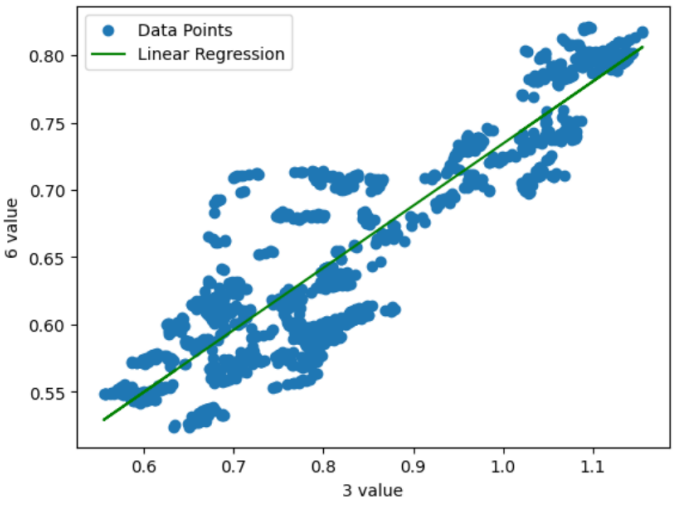

Figure 10. Plot the most correlated value with the 6th value

Initially, we filled the missing values of ’6’ using linear regression with feature ’3’. However, some gaps persisted, indicating moments when both features ’3’ and ’6’ were missing. Consequently, we repeated the process with the second most correlated feature after applying the initial transformation.

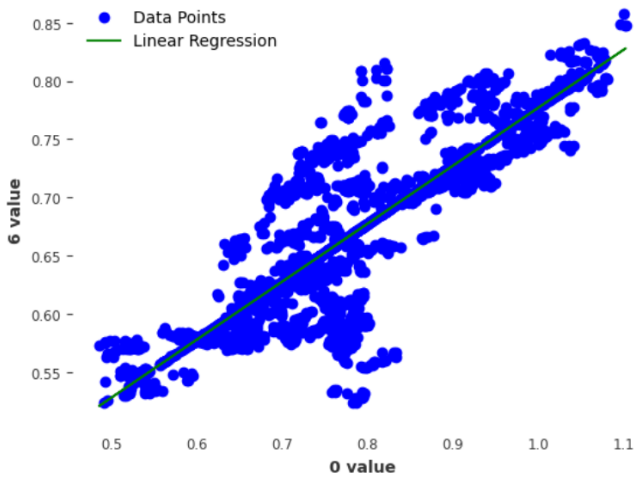

Figure 11. Plot the second most correlated with value the 6th value

Then, we employed the interpolate method (method=nearest) from Pandas to fill in the remaining missing values based on the nearest available data points. However, the linear regression method did not yield as satisfactory results as backward filling, as observed in Kaggle’s evaluation. This could be attributed to the inherent limitations of linear regression as an approximation method, particularly when the correlation between features is not strong. The visual analysis of the data schemas did not reveal a strong correlation between features, which may have affected the effectiveness of linear regression in imputing missing values.

4.2. Models and implementation

For the implementation of our predictive model, we followed the same methodology as in Part 1. This involved model selection, hyperparameter tuning (see Table 1), and performance evaluation using the same techniques and procedures.

Initially, our intention was to use this approach to gain insights into the performance of our models before training a model with multiple features. However, during this preliminary testing phase, we achieved exceptional prediction results using the same method as in Part 1 but with different hyperparameters for our model (see Table 4).

Table 4. Hyper-parameters for the TiDE+RINModel for Part 2

Parameters	Value
inputChunkLength	512
outputChunkLength	100
hiddenSize	256
numEncoderLayers	2
numDecoderLayers	2
decoderOutputDim	32
temporalDecoderHidden	16
useLayerNorm	True
dropout	0.5

Given the outstanding predictive performance observed with this approach, we made the decision to forego training a model with multiple features. Instead, we proceeded to train our final model solely on the ’6’ feature, as it provided excellent forecasting accuracy and met our project objectives effectively.

4.3. Results

In this Figure 12 below, we showcase our best Kaggle score achieved thanks to the TIDE+RIN Model for the Part 2. The lower MAE obtained in Part 2 in contrast to Part 1 can be attributed to the relatively homogeneous nature of the predicted values.

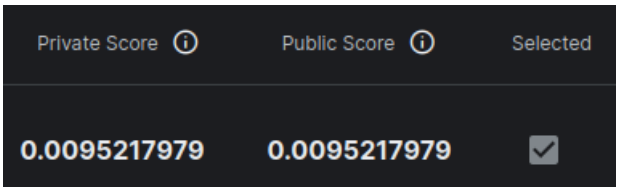

Figure 12. result of Kaggle MAE = 0.0095

Here are other results we obtained with different preprocessing methods on Kaggle :

Table 5. Differents preprocessing and their results in Kaggle

Preprocessing	Mae in Kaggle
linear regression	0.012
juste interpolate method	0.02
backward filling	0.0098

5. Conclusion

In conlusion, our project delved into the realm of time series classification, focusing on two distinct tasks: univariate time series forecasting of oil transformer temperature and multivariate time series forecasting of exchange rates while handling missing values.

For the univariate time series task, we successfully developed a predictive model, with the TiDE+RINModel demonstrating superior performance in forecasting oil temperature. Hyperparameter tuning and model selection were pivotal in achieving these results, leading us to select the TiDE+RINModel for further analysis.

In the multivariate time series forecasting task, we initially intended to explore predictive models with multiple features. However, during preliminary testing, we observed exceptional performance using the same method employed in the univariate task but with different hyperparameters. Consequently, we decided to focus solely on training and forecasting the '6' feature.

Overall, our project highlights the effectiveness of advanced modeling techniques in time series forecasting. By leveraging sophisticated models and careful hyperparameter tuning, we were able to achieve accurate predictions in both univariate and multivariate contexts. These findings underscore the importance of methodical experimentation and validation in developing robust predictive models for time series data. Moving forward, further research could explore additional datasets and modeling approaches to expand the scope of time series classification tasks.

References

[1] H. Lütkepohl, *New Introduction to Multiple Time Series Analysis*. Springer, 2005.

[2] G. E. P. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time Series Analysis: Forecasting and Control*. John Wiley & Sons, 2015.

[3] Y. Zhang, G. J. Qi, and C. D. Manning, "Time series individualized deep embeddings for forecasting", in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ACM, 2019, pp. 1302–1311.

[4] B. N. Oreshkin, D. Carpov, N. Chapados, and Y. Bengio, "N-beats: Neural basis expansion analysis for time series forecasting", in *International Conference on Learning Representations*, 2020.

[5] B. Lim, S. Ö. Arık, N. Loeff, and T. Pfister, "Temporal fusion transformers for interpretable multi-horizon time series forecasting", *International Journal of Forecasting*, vol. 37, no. 4, pp. 1748–1764, 2021, ISSN: 0169-2070.

[6] *Forecasting*, https://en.wikipedia.org/wiki/Forecasting, [Online; accessed 18-April-2024].

[7] *Time series*, https://en.wikipedia.org/wiki/Time_series, [Online; accessed 18-April-2024].

[8] Unit8co, *Darts - time series library: Tide model*, https://unit8co.github.io/darts/generated_api/darts.models.forecasting.tide_model.html.