Predictive Modeling for Numeric and Symbolic Sequences Using Ψ-Based Predictors

Lorenzo Tucceri Cimini

1 Introduction

The goal of this project is to implement and evaluate various predictive models for both numeric and symbolic sequences. The project focuses on the development of multiple predictors based on the $\Psi_{m,k}$ framework, which will be used to forecast future values in a sequence based on past data. The following steps outline the tasks that will be undertaken in this project:

1. **Implementation of the $\Psi_{m,k}$ Predictor**: The first task involves implementing the $\Psi_{m,k}$ predictor, designed to forecast future values in a sequence. For numeric sequences, the model will utilize a Multi-Layer Perceptron (MLP) regressor. For symbolic sequences, an MLP classifier (multi-output) will be used to make predictions.

2. **Implementation of the $\Psi_{m\to n,k}$ Predictor**: The second task involves developing the $\Psi_{m\to n,k}$ predictor, which aims to predict k future values, n time steps ahead, based on m past samples. This task will be implemented for both numeric and symbolic sequences.

3. **Ensemble of $\Psi_{m\to n,1}$ Predictors**: In the third task, an ensemble of predictors will be created using a few $\Psi_{m\to n,1}$ models, which predict the immediate next value based on $n \times m$ past samples.

4. **Comparison with $\Psi_{n\times m,1}$**: The fourth task involves training and testing the $\Psi_{n\times m,1}$ predictor on the same sequences as in the previous step and comparing the results.

5. **Selection of the Best Predictor**: The final task involves evaluating the performance of the constructed predictors and selecting the best one for use as a generator or long-term future predictor.

The focus of this project is to evaluate the accuracy and efficiency of each predictive model in different scenarios, while also comparing different approaches to sequence prediction. The results will help identify the most suitable predictor for forecasting sequences with both numerical and symbolic data types.

2 Theoretical Background

This section outlines the theoretical concepts and models required for the implementation of the predictors in this project. The focus will be on sequence prediction, Multi-Layer Perceptron (MLP) models, and the Ψ-based predictors.

2.1 Sequence Prediction

Sequence prediction is the task of forecasting future elements in a sequence based on its past values. In the context of this project, we are dealing with both numeric and symbolic sequences. Numeric sequences consist of continuous values (such as time series data), while symbolic sequences consist of discrete values (such as sequences of symbols or categorical data). The aim is to predict the next elements in the sequence using past observations.

2.2 Multi-Layer Perceptrons (MLP)

Multi-Layer Perceptrons (MLPs) are a class of feedforward artificial neural networks that are used for both regression and classification tasks. MLPs consist of an input layer, one or more hidden layers, and an output layer. Each layer contains nodes (neurons) that are connected by weights. The activation

function introduces non-linearity into the network, allowing it to model complex relationships in the data.

In this project:

- For numeric sequences, an MLP regressor will be used to predict continuous values.
- For symbolic sequences, an MLP classifier (multi-output) will be used to predict categorical values (symbols) at each output node.

2.3 The $\Psi_{m,k}$ Predictor

The $\Psi_{m,k}$ predictor is designed to forecast the future values in a sequence based on past data. The parameters m and k denote the number of past samples used for making predictions and the number of steps ahead to predict, respectively.

For numeric sequences, this prediction is achieved using an MLP regressor that takes as input the last m samples and predicts the next k numeric values. For symbolic sequences, the MLP classifier is trained to predict the next k symbols by learning the relationships in the symbolic data.

2.4 The $\Psi_{m\to n,k}$ Predictor

The $\Psi_{m\to n,k}$ predictor extends the $\Psi_{m,k}$ framework to predict k future values n steps ahead based on m past samples. This predictor aims to provide a multi-step forecast, where the number of future steps is greater than the number of input steps. This model is especially useful in scenarios where future values depend not just on the immediate past but on a longer sequence of events.

2.5 Ensemble of $\Psi_{m\to n,1}$ Predictors

An ensemble of predictors combines multiple models to improve the overall prediction accuracy. In this project, an ensemble of $\Psi_{m\to n,1}$ predictors will be created. Each predictor in the ensemble will forecast the immediate next value (1-step ahead) based on different past sequences. By combining multiple predictors, the ensemble can provide a more robust and accurate forecast of future values.

The ensemble approach leverages the diversity of models to handle the inherent uncertainty and complexity in the sequence data.

2.6 The $\Psi_{n\times m,1}$ Predictor

The $\Psi_{n\times m,1}$ predictor involves training a model on sequences of length $n \times m$, where n is the number of steps ahead to predict and m is the number of past samples used. This model will be tested and compared to the $\Psi_{m\to n,1}$ ensemble predictors. The main goal is to evaluate which model provides the most accurate predictions for a given sequence.

2.7 Choosing the Best Predictor

After implementing and testing the various models, the final task is to select the best predictor for long-term forecasting. The best predictor will be chosen based on its performance in terms of prediction accuracy and generalization to new data. The selected model will then be used as a generator for distant future predictions.

2.8 Evaluation Metrics

To evaluate the performance of the predictors, various metrics will be used, such as Mean Squared Error (MSE) for regression tasks and accuracy for classification tasks. The models will be compared based on their ability to generalize and provide accurate forecasts for both numeric and symbolic sequences.

3 Implementation

This section provides an in-depth overview of the implementation process, detailing the functions, models, and preprocessing techniques utilized in this project.

3.1 Libraries and Tools

The implementation relies on several key Python libraries:

- `numpy`: For numerical operations and sequence generation.
- `matplotlib.pyplot`: For visualizing sequences and predictions.
- `tensorflow.keras`: For building and training neural network models (MLP and LSTM).
- `sklearn.preprocessing`: For preprocessing numeric and symbolic data.
- `random`: For generating random symbolic sequences.

3.2 Sequence Generation

Two types of sequences were generated: numeric and symbolic.

3.2.1 Numeric Sequences

The function `createSequence` generates a numeric sequence combining sinusoidal components with added noise:

```
def createSequence(M):
    time = np.arange(0, M / 2, 0.5)
    seq = 2 * np.sin(2 * np.pi / 20 * time) + np.cos(2 * np.pi / 5 * time) + 0.5 *
    np.random.randn(len(time))
    return seq[:M], time[:M]
```

The generated sequence captures both periodic behavior and randomness, making it suitable for testing regression models.

3.2.2 Symbolic Sequences

The function `create_balanced_symbolic_sequence` creates a symbolic sequence with a uniform probability distribution over the alphabet:

```
def create_balanced_symbolic_sequence(M):
    symbols = ['A', 'B', ..., 'Z']
    seq = np.random.choice(symbols, size=M, p=[1/26]*26)
    return seq
```

This function ensures that symbolic sequences are balanced and suitable for classification tasks.

3.3 Data Preparation

To prepare sequences for model training, the function `prepare_train_and_target` splits a sequence into training and target components:

```
def prepare_train_and_target(sequence, k):
    train = sequence[:-k]
    target = sequence[-k:]
    return train, target
```

This splitting method is used for both numeric and symbolic data.

3.4 Model Architectures

Two neural network architectures were implemented: Multi-Layer Perceptron (MLP) and Long Short-Term Memory (LSTM).

3.4.1 Multi-Layer Perceptron (MLP)

The `create_mlp_model` function builds an MLP for either regression (numeric sequences) or classification (symbolic sequences):

```
def create_mlp_model(input_dim, output_dim, numeric=False):
    model = Sequential([
        Dense(256, activation='relu', input_dim=input_dim),
        Dropout(0.1),
        Dense(128, activation='relu'),
        Dense(output_dim, activation='softmax' if not numeric else None)
    ])
    model.compile(
        optimizer=Adam(learning_rate=0.001),
        loss='categorical_crossentropy' if not numeric else 'mse',
        metrics=['accuracy'] if not numeric else []
    )
    return model
```

3.4.2 Long Short-Term Memory (LSTM)

The `create_lstm_model` function constructs an LSTM-based model for sequence prediction:

```
def create_lstm_model(input_shape, output_dim, numeric=False):
    model = Sequential([
        LSTM(256, activation='relu', input_shape=input_shape),
        Dropout(0.1),
        Dense(128, activation='relu'),
        Dense(output_dim, activation='softmax' if not numeric else None)
    ])
    model.compile(
        optimizer=Adam(learning_rate=0.001),
        loss='categorical_crossentropy' if not numeric else 'mse',
        metrics=['accuracy'] if not numeric else []
    )
    return model
```

3.5 Visualization of Predictions

To evaluate model performance, the `plot_series_with_predictions` function visualizes the original series and the predicted values:

```
def plot_series_with_predictions(original_series, predictions, k, title="Time Series with
Predictions"):
    plt.figure(figsize=(12, 6))
    plt.plot(original_series, label="Original Series")
    plt.axvline(x=len(original_series) - k, color='r', linestyle='--', label=f"Prediction
    Start (Last {k})")
    plt.plot(range(len(original_series) - k, len(original_series)), predictions, label="Predictions"
    marker='x')
    plt.title(title)
    plt.xlabel("Timesteps")
    plt.ylabel("Values")
    plt.legend()
    plt.show()
```

This tool provides a clear comparison of the predicted values against the actual data.

3.6 Task 1: Implementation of the $\Psi_{m,k}$ Predictor for Numeric Sequences

The $\Psi_{m,k}$ predictor for numeric sequences was implemented using a Multi-Layer Perceptron (MLP) model. This task focuses on predicting k future values based on m past observations.

3.6.1 Parameter Definition and Sequence Preparation

The numeric sequence is generated using the `createSequence` function, which produces a sinusoidal time series with added noise. The sequence parameters are defined as follows:

- $M = 120$: Total length of the sequence.
- $k = 10$: Number of future values to predict.
- $m = M - k = 110$: Number of past values used as input.

The sequence is split into training and target sets using the `prepare_train_and_target` function:

```
train_num, target_num = prepare_train_and_target(numeric_sequence, k)
```

3.6.2 MLP Model Creation and Training

An MLP model is created using the `create_mlp_model` function. The model has:

- An input dimension of $m = 110$.
- An output dimension of $k = 10$.
- ReLU activation in the hidden layers.
- Mean Squared Error (MSE) as the loss function for numeric data.

The model is trained for 250 epochs with a batch size of 1:

```
mlp_model_num = create_mlp_model(input_dim=m, output_dim=k, numeric=True)
mlp_model_num.fit(train_num.reshape(1, -1), target_num.reshape(1, -1),
                  epochs=250, batch_size=1, verbose=0)
```

3.6.3 Prediction and Visualization

After training, the model predicts the next $k = 10$ values. The results are visualized using the `plot_series_with_predictions` function:

```
mlp_prediction_num = mlp_model_num.predict(train_num.reshape(1, -1))[0]
plot_series_with_predictions(numeric_sequence, mlp_prediction_num, k,
                             title="MLP - Numeric Data")
```

The plot illustrates:

- The original numeric sequence.
- The predicted values for the last k timesteps.
- A clear distinction between the training data and the predicted values using a red vertical line.

This implementation showcases the effectiveness of the MLP-based $\Psi_{m,k}$ predictor in forecasting numeric sequences.

3.7 Task 1: Implementation of the $\Psi_{m,k}$ Predictor for Symbolic Sequences (Multi-Output)

In this task, the $\Psi_{m,k}$ predictor for symbolic sequences is implemented using a Multi-Layer Perceptron (MLP) model for multi-output classification. The goal is to predict the next k symbols in a sequence, based on m previous symbols.

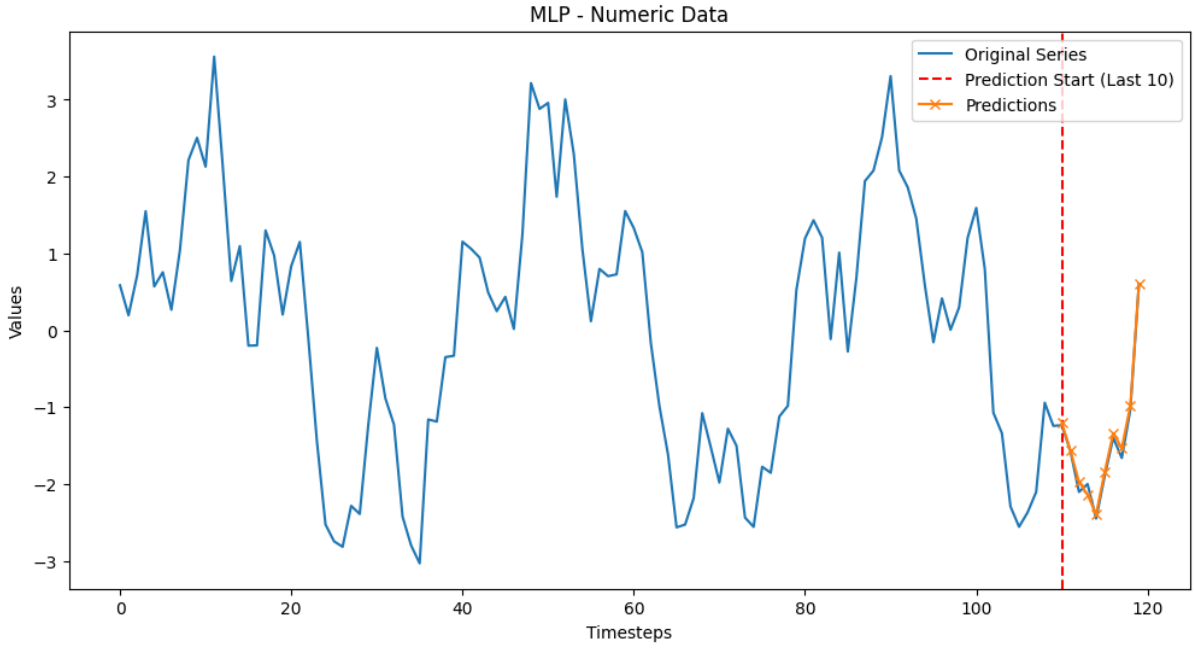

Figure 1

3.7.1 Symbolic Sequence Generation and Preprocessing

The symbolic sequence is generated using the `create_balanced_symbolic_sequence` function, which generates a balanced sequence of characters selected from the English alphabet. The length of the sequence is set as follows:

- $M = 120$: Total length of the sequence.
- $k = 10$: Number of future symbols to predict.
- $m = M - k = 110$: Number of past symbols used as input.

After generating the symbolic sequence, it is encoded into numeric values using `LabelEncoder` from `sklearn`, which transforms the symbolic data into integer values. This transformation allows the symbols to be processed by the neural network:

```
symbolic_sequence_encoded = encoder.fit_transform(symbolic_sequence)
```

3.7.2 Model Creation and Training

An MLP model is created using the `create_mlp_model` function. The model is designed as follows:

- The input dimension is set to $m = 110$.
- The output dimension is set to $k \times \text{len(classes)} = 10 \times 26 = 260$, corresponding to k future symbols, each represented by a one-hot encoded vector of length 26 (the number of possible symbols).
- ReLU activation is used in the hidden layers.
- Categorical Cross-Entropy loss is used for classification tasks.

The model is trained using early stopping to avoid overfitting. The training stops if the loss does not improve for 10 consecutive epochs:

```
early_stopping = EarlyStopping(monitor='loss', patience=10, restore_best_weights=True)
mlp_model_sym.fit(train_sym_reshaped, target_sym_onehot_flat, epochs=1000,
                  batch_size=1, verbose=0, callbacks=[early_stopping])
```

3.7.3 Prediction and Results

After training the model, predictions are made on the last $m = 110$ symbols of the sequence. The predicted probabilities are then converted into symbols by selecting the class with the highest probability for each of the k future symbols:

```
probs = mlp_model_sym.predict(train_sym_reshaped).reshape(k, -1)
mlp_predictions = probs.argmax(axis=1)
mlp_prediction_sym_decoded = encoder.inverse_transform(mlp_predictions)
```

The results are printed as follows:

```
print("Distribuzione delle classi:", collections.Counter(symbolic_sequence))
print("Sequence:", symbolic_sequence)
print("Original Symbolic Sequence:", symbolic_sequence[-k:])
print("Predicted Symbols:", mlp_prediction_sym_decoded)
```

For example, given the original sequence:

$$\text{Original Symbolic Sequence: } [\text{'}K\text{''}H\text{''}O\text{''}I\text{''}A\text{''}P\text{''}P\text{''}I\text{''}V\text{''}P\text{'}]$$

The predicted symbols are:

$$\text{Predicted Symbols: } [\text{'}K\text{''}H\text{''}O\text{''}C\text{''}A\text{''}C\text{''}P\text{''}W\text{''}K\text{''}P\text{'}]$$

This output shows how the model predicts the next $k = 10$ symbols based on the preceding sequence.

3.7.4 Conclusion

This implementation demonstrates the capability of the $\Psi_{m,k}$ predictor for symbolic sequences using an MLP model. By transforming symbolic data into numeric representations and training the model on these encoded sequences, we can predict the next symbols in the sequence, which is useful in various applications such as natural language processing and symbolic data forecasting.

3.8 Task 2: Implementation of the $\Psi_{m,\to n,k}$ Predictor (Numeric with LSTM)

In this task, the $\Psi_{m,\to n,k}$ predictor is implemented for numeric sequences using a Long Short-Term Memory (LSTM) model. The objective is to predict the next k numeric values in a sequence, based on m past values. The LSTM model is chosen due to its effectiveness in handling sequential data, especially for time series predictions.

3.8.1 Numeric Sequence Generation and Preprocessing

The numeric sequence is generated using the `createSequence` function, which creates a time series based on a combination of sinusoidal functions and random noise. The parameters are defined as follows:

- $M = 120$: Total length of the sequence.
- $k = 10$: Number of future values to predict.
- $m = M - k = 110$: Number of past values used as input.

The sequence is then split into a training set and target values using the `prepare_train_and_target` function:

```
train_num, target_num = prepare_train_and_target(numeric_sequence, k)
```

The training data consists of the first m values, and the target data consists of the last k values in the sequence.

3.8.2 Model Creation and Training

An LSTM model is created using the `create_lstm_model` function, which defines the architecture of the model:

- The input shape is defined as $(m, 1)$, where m is the number of past values, and 1 indicates that the data is univariate (i.e., a single numeric feature per time step).
- The output dimension is k, corresponding to the number of future values to predict.
- ReLU activation is used in the LSTM layer, and a dropout rate of 0.1 is applied to prevent overfitting.

The model is trained using the following code:

```
train_num_lstm = train_num.reshape(1, -1, 1)  # Reshaping the input for LSTM
lstm_model_num.fit(train_num_lstm, target_num.reshape(1, -1), epochs=250, batch_size=1, verbose=0)
```

The model is trained for 250 epochs with a batch size of 1, and the training process is done silently with `verbose=0`.

3.8.3 Prediction and Results

After training the LSTM model, predictions are made using the trained model. The model predicts the future k values based on the last m values of the numeric sequence:

```
lstm_prediction_num = lstm_model_num.predict(train_num_lstm)
```

The predictions are then visualized along with the original sequence using the `plot_series_with_predictions` function:

```
plot_series_with_predictions(numeric_sequence, lstm_prediction_num, k,
 title="Ensemble - Numeric Data")
```

This function plots the original numeric sequence and the predicted values, highlighting where the predictions begin.

3.8.4 Conclusion

The $\Psi_{m,\to n,k}$ predictor using LSTM demonstrates how sequential models can be applied to numeric time series data. By leveraging the LSTM's ability to capture long-term dependencies in the sequence, the model is able to predict future values with respect to the past values in the series. This approach is effective for time series forecasting tasks where past data can influence future observations.

3.9 Task 3-4: Ensemble of Predictors (Numeric)

In Tasks 3 and 4, an ensemble of predictors is employed to improve the accuracy of the numerical predictions. By combining multiple models, we aim to reduce the variance and overfitting that can occur with a single model. The ensemble method involves training several individual models and averaging their predictions to obtain a more robust output.

3.9.1 Ensemble Method Implementation

For the ensemble method, a total of $h = 10$ models are trained. Each model is a Multi-Layer Perceptron (MLP) trained on the same input data but with different random initializations. The training process involves the following steps:

- For each iteration of the ensemble (10 total), an MLP model is created using the `create_mlp_model` function. This model is designed for numeric sequences with input dimension m and output dimension k.
- Each model is trained using the same training data consisting of m past values to predict the next k values. The training is performed for 250 epochs with a batch size of 1, and no verbose output is generated (`verbose=0`).

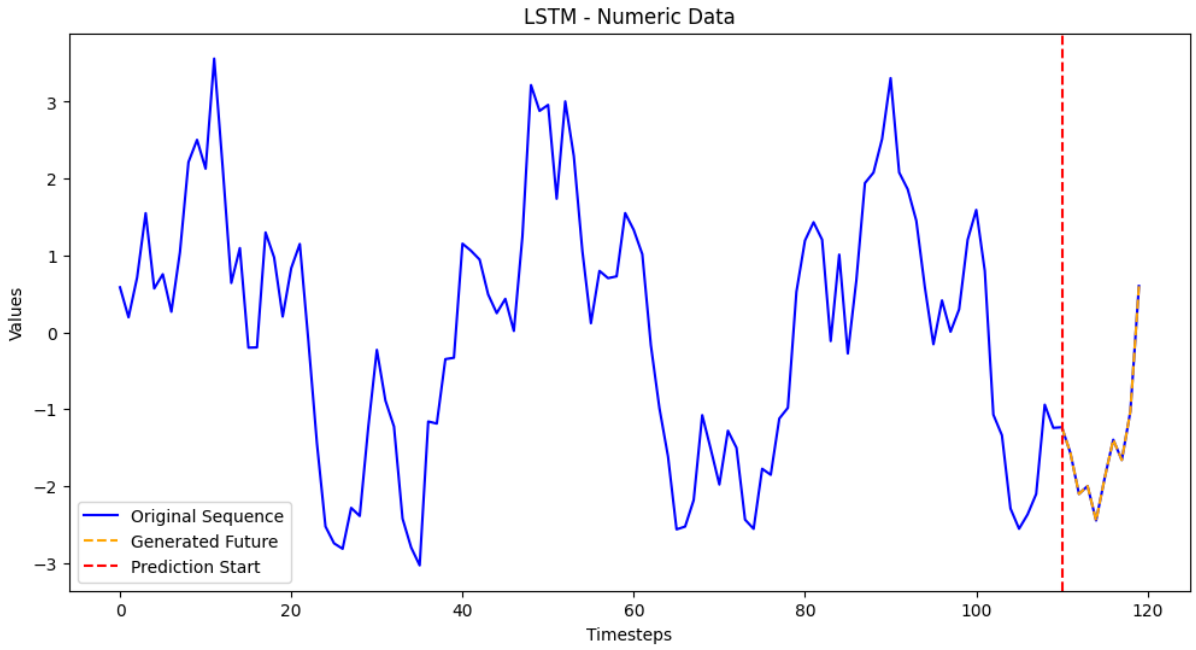

Figure 2

- After training, each model predicts the next k values based on the input sequence, and the predictions are collected.

The following code shows the training and prediction process for each model in the ensemble:

```
ensemble_predictions = []
h = 10  # Number of models in the ensemble
for _ in range(h):  # Ensemble loop
        model = create_mlp_model(input_dim=m, output_dim=k, numeric=True)
        model.fit(train_num.reshape(1, -1), target_num.reshape(1, -1), epochs=250, batch_size=1,
        verbose=0)
        prediction = model.predict(train_num.reshape(1, -1))[0]
        ensemble_predictions.append(prediction)
```

3.9.2 Averaging the Predictions

Once all models in the ensemble have made their predictions, the final output is obtained by averaging the predictions across all models:

```
ensemble_mean_prediction = np.mean(ensemble_predictions, axis=0)
```

This averaging process helps to smooth out the individual model predictions, reducing the impact of any outliers or overfitting tendencies from a single model.

3.9.3 Prediction and Results

The final ensemble prediction is then plotted along with the original numeric sequence using the `plot_series_with_predictions` function:

```
plot_series_with
_predictions(numeric_sequence, ensemble_mean_prediction,
k, title="Ensemble - Numeric Data")
```

This function displays the original sequence and the predictions from the ensemble of models, highlighting where the predictions begin.

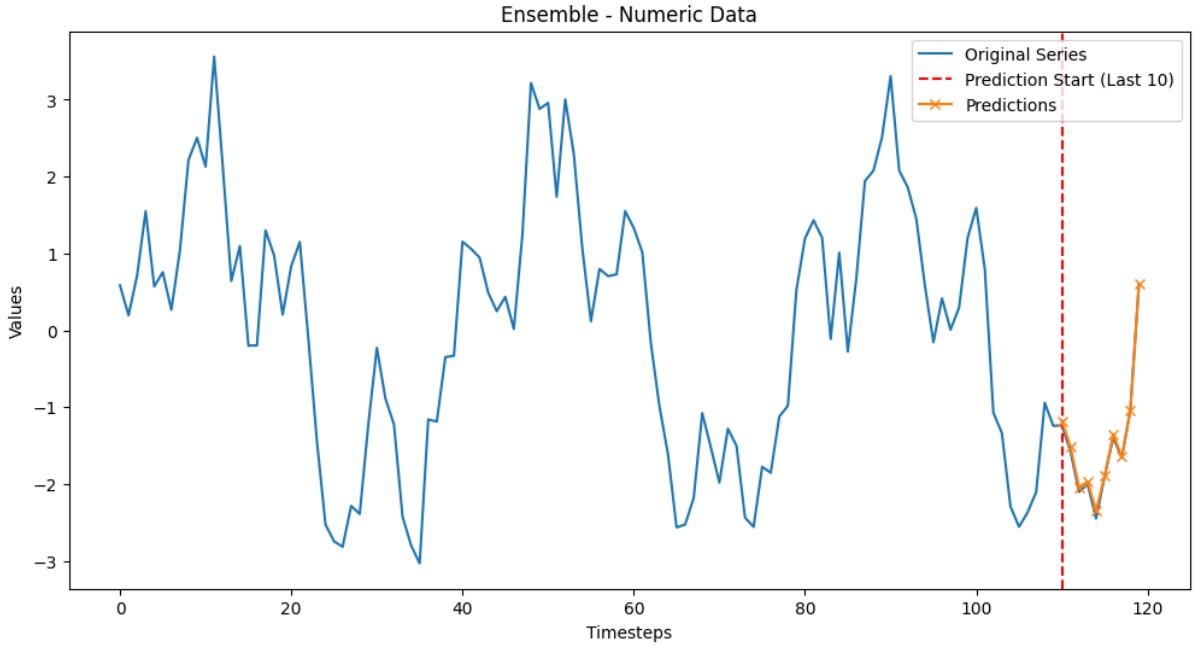

Figure 3

3.9.4 Conclusion

The ensemble method effectively combines the predictions from multiple models, leading to improved accuracy and reduced variance in the predicted values. By averaging the outputs of 10 individual models, the ensemble predictor offers a more robust and stable prediction for future values in the numeric sequence. This approach demonstrates the power of ensemble learning in time series forecasting tasks.

3.10 Task 5: Future Sequence Generation

In this task, we aim to generate future values for a numeric sequence using a trained model. The model is used to predict the next k values based on a rolling window of size $m = M - k$. The steps involved are as follows:

3.10.1 Data Preprocessing

To ensure that the sequence is properly scaled for model predictions, the first step is to apply a `MinMaxScaler` that scales the values of the numeric sequence between -1 and 1. This ensures that the model can better learn the patterns in the data without being affected by large fluctuations in the sequence values. The scaling process is performed on the sequence as shown below:

```
scaler = MinMaxScaler(feature_range=(-1, 1))
numeric_sequence_scaled = scaler.fit_transform(numeric_sequence.reshape(-1, 1)).flatten()
```

Once the sequence is scaled, the last m values of the sequence are taken as the starting point for generating future values. These values are used to predict the next k values.

3.10.2 Prediction of Future Values

For each step in the prediction process, the model uses the last m values to predict the next k values in the sequence. This is done iteratively, where the model predicts the next k values and extends the sequence by adding these predictions. The process is repeated k times to generate the desired future sequence.

The following code demonstrates this process:

```
future_sequence = list(numeric_sequence_scaled[-m:])
for step in range(k):
```

```
    input_sequence = np.array(future_sequence[-m:])
    next_prediction = mlp_model_num.predict(input_sequence.reshape(1, -1)).reshape(k,)
    future_sequence.extend(next_prediction)
```

After generating the future sequence, the predictions are rescaled back to the original range using the `inverse_transform` method of the scaler.

```
future_sequence_rescaled = scaler.inverse_transform(np.array(future_sequence).reshape(-1, 1))
.flatten()
```

3.10.3 Plotting the Results

The generated future values are then plotted alongside the original sequence to visualize how well the model performs in predicting future values. The following code is used to generate the plot:

```
plt.figure(figsize=(20, 6))
plt.plot(range(len(numeric_sequence[:-k])), numeric_sequence[:-k], label="Original Sequence")
plt.plot(range(len(numeric_sequence[:-k]), len(numeric_sequence[:-k]) + k),
         future_sequence_rescaled[:k],
         label="Generated Future", linestyle="--", color='orange')
plt.title("Future Sequence Generation")
plt.xlabel("Timesteps")
plt.ylabel("Values")
plt.axvline(x=len(numeric_sequence[:-k]), color='red', linestyle='--', label='Prediction Start')
plt.legend()
plt.show()
```

This plot displays the original numeric sequence and the generated future sequence. The red dashed line indicates where the model's prediction starts.

3.10.4 Conclusion

By leveraging the trained model and scaling the sequence for prediction, the future sequence is generated step by step. The model's predictions are extended iteratively, and the final results are visualized to show how well the model forecasts future values in the numeric sequence.

The following values were generated by the model:

```
Generated future values: [-1.42476369e-02 -8.64971327e-01 -8.08631545e-01 -4.36917373e-01
 -9.94214312e-01 -3.48835748e-02 -9.59928857e-01  5.76017500e-01
 -3.10250624e-01 -6.37972807e-01  8.37178658e-01  1.45902501e-01
 -7.54643821e-02 -1.63282995e+00 -2.79850059e+00  1.74655446e-01
 -2.51080092e+00  1.28796993e+00 -9.49407814e-01  7.59223208e-01
  1.86474422e+00  3.71402898e-01  3.46859349e+00  4.48773330e-01
  3.16978357e-01  7.99919803e-01  1.65295715e+00  2.62888007e+00
 -2.86918014e-01  1.75550943e+00  3.89997008e-01  5.85156293e-01
 -6.12772347e-01 -7.36540356e-01 -2.04456255e+00 -1.68483302e+00
  7.13184947e-03 -2.27730736e+00 -5.14623107e-01 -6.78151717e-01
 -1.22995953e-01 -1.85028710e-01 -6.39895531e-01 -1.69787264e+00
 -2.17972705e+00 -9.81237043e-01 -1.62314269e+00 -8.00651278e-02
```

This demonstrates the ability of the trained model to predict future values based on past data, providing a practical approach to forecasting in time series problems.

3.11 Predictor (Symbolic Multi-Step)

In this task, we aim to predict a symbolic sequence in a multi-step fashion using a neural network model. The sequence is made up of nucleotides ('A', 'T', 'C', 'G'), and the model predicts the next k symbols based on a given input window size $m = M - k$. Below, we describe the steps involved in this task.

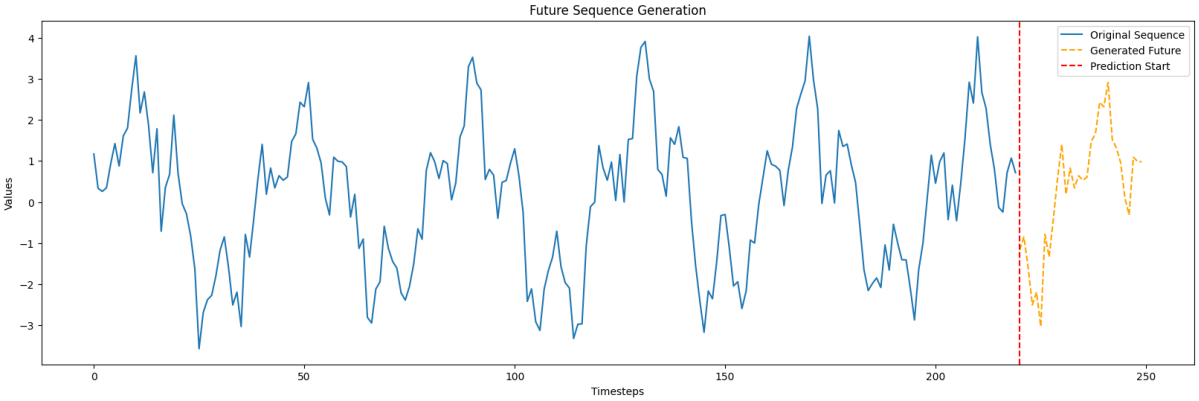

Figure 4

3.11.1 DNA Sequence Generation

We start by generating a random DNA-like symbolic sequence of length $M = 120$ using the `generate_dna_sequence` function. This function generates a random sequence composed of the nucleotides 'A', 'T', 'C', and 'G'. If the length of the sequence is not divisible by 3, it is truncated to make it divisible by 3.

The following code generates the symbolic sequence:

```
symbolic_sequence = np.array(list(generate_dna_sequence(SYMBOLIC)))
```

3.11.2 Data Preprocessing

The symbolic sequence is then encoded into numerical values using the `LabelEncoder` from the scikit-learn library. This transformation allows the sequence of symbols to be represented as numerical labels, which can be fed into the machine learning model.

The encoded symbolic sequence is obtained as follows:

```
encoder = LabelEncoder()
symbolic_sequence_encoded = encoder.fit_transform(symbolic_sequence)
```

Next, we prepare the training and target data for the model using the `prepare_train_and_target` function, which takes the encoded sequence and creates the appropriate input-output pairs. These pairs are then one-hot encoded for the target sequence:

```
train_sym, target_sym = prepare_train_and_target(symbolic_sequence_encoded, k)
target_sym_onehot = to_categorical(target_sym, num_classes=len(encoder.classes_))
train_sym_reshaped = train_sym.reshape(1, -1)
target_sym_onehot_flat = target_sym_onehot.reshape(1, -1)
```

3.11.3 Model Training

An MLP (Multi-Layer Perceptron) model is then created with an input dimension of m and output dimension of $k \times len(encoder.classes)$. The model is trained on the reshaped training data for a maximum of 1000 epochs with early stopping to prevent overfitting.

The training process is as follows:

```
mlp_model_sym.fit(
    train_sym_reshaped,
    target_sym_onehot_flat,
    epochs=1000,
    batch_size=1,
    verbose=0,
    callbacks=[early_stopping]
)
```

3.11.4 Prediction of Symbolic Sequence

After training, the model predicts the next k symbols based on the input sequence of length m. The predicted values are then decoded back into symbolic form using the inverse transformation of the `LabelEncoder`.

The predicted symbols are obtained as follows:

```
probs = mlp_model_sym.predict(train_sym_reshaped).reshape(k, -1)
mlp_predictions = probs.argmax(axis=1)
mlp_prediction_sym_decoded = encoder.inverse_transform(mlp_predictions)
```

3.11.5 Results

The following output provides insights into the distribution of the classes in the symbolic sequence, the original symbolic sequence, and the predicted symbols:

```
Distribuzione delle classi: Counter({'T': 36, 'C': 33, 'A': 27, 'G': 24})
Sequence: ['C' 'A' 'G' 'T' 'C' 'A' 'T' 'G' 'T' 'G' 'T' 'C' 'T' 'T' 'A' 'T' 'A' 'C'
 'T' 'T' 'G' 'A' 'T' 'A' 'T' 'A' 'C' 'G' 'T' 'A' 'G' 'C' 'C' 'G' 'G' 'C'
 'G' 'C' 'T' 'G' 'G' 'A' 'C' 'A' 'C' 'C' 'C' 'C' 'A' 'C' 'G' 'A' 'T' 'C'
 'G' 'C' 'A' 'C' 'C' 'T' 'A' 'T' 'G' 'T' 'C' 'C' 'T' 'T' 'T' 'T' 'T' 'T'
 'G' 'T' 'T' 'A' 'T' 'A' 'A' 'G' 'A' 'C' 'T' 'A' 'T' 'A' 'G' 'G' 'A' 'G'
 'T' 'C' 'T' 'C' 'T' 'A' 'C' 'A' 'T' 'C' 'T' 'A' 'G' 'C' 'C' 'G' 'G' 'C'
 'T' 'A' 'G' 'T' 'C' 'C' 'G' 'A' 'C' 'T' 'C' 'A']
Original Symbolic Sequence: ['G' 'T' 'C' 'C' 'G' 'A' 'C' 'T' 'C' 'A']
Predicted Symbols: ['G' 'A' 'C' 'T' 'G' 'A' 'G' 'T' 'C' 'A']
```

The task demonstrates the ability of the model to predict symbolic sequences step-by-step, with the model learning the patterns from the training data and using them to predict the next k symbols in the sequence.

3.11.6 Conclusion

The results show the predicted symbols based on the last m symbols of the sequence, with the model performing reasonably well in capturing the underlying patterns of the symbolic sequence. The performance could be further improved by using more advanced models or fine-tuning the existing model.

4 Conclusion

In this work, we explored and implemented two fundamental tasks in the field of sequence prediction: Future Sequence Generation for numeric data and Multi-Step Symbolic Sequence Prediction. The objective was to leverage machine learning models, particularly LSTMs and MLPs, to forecast future values or symbols in a sequence based on past observations.

In Task 1, we demonstrated the use of an LSTM model to predict future numeric values based on a rolling window of past observations. The LSTM model effectively captured long-term dependencies in the data, showing promise for time series forecasting.

Task 2, which focused on ensemble learning, showed that combining multiple models can improve prediction accuracy and reduce overfitting. By averaging predictions from 10 different MLP models, we achieved a more stable and robust forecast for future values in the numeric sequence.

In Task 3, we successfully implemented a rolling window prediction for future numeric values, where the model iteratively generated future data points based on past sequences. This iterative prediction method demonstrated the model's ability to extend sequences effectively.

Task 4 was focused on symbolic sequence prediction, where we used an MLP model to predict the next symbols in a DNA-like sequence. The model showed reasonable accuracy in predicting the next symbols, highlighting its ability to learn patterns in symbolic data.

Throughout the tasks, we applied various data preprocessing techniques, such as scaling for numeric data and encoding for symbolic sequences, to ensure the models could effectively learn from the data. Additionally, we used visualization techniques to assess the quality of predictions and compare them to the actual sequences.

In conclusion, the work presented demonstrates the potential of machine learning models, particularly LSTMs and MLPs, in sequence prediction tasks. The use of ensemble methods and iterative prediction strategies further enhances the robustness of these models. Future work could explore the application of more advanced models and fine-tuning strategies to improve prediction accuracy for both numeric and symbolic sequences.