Homework 1

Homework 1 involves the development of a comprehensive time series analysis framework, spanning various components including dataset handling, data transformations, metric calculations, and forecasting models.

Usage examples

```
python main.py --data_path ./dataset/ETT/ETTh1.csv --dataset ETT --target OT --
model MeanForecast
```

```
python main.py --data_path ./dataset/ETT/ETTh1.csv --dataset ETT --target OT --
model TsfKNN --n_neighbors 1 --msas MIMO --distance euclidean
```

Part 1. Dataset (30 pts)

path: `src/dataset/dataset.py`, `src/dataset/data_visualizer.py`

All datasets can be found here.

Objective: In this part, you will implement a custom dataset class, `CustomDataset`, to handle various time series datasets. Your custom dataset class will inherit from a base class, `DatasetBase`, and provide functionality to read and preprocess the data. This assignment aims to enhance your understanding of dataset handling in machine learning and time series analysis.

Instructions:

1. CustomDataset Class:

- Your task is to implement the `CustomDataset` class, a subclass of `DatasetBase`, specifically tailored to handle various time series datasets.
- Implement the `read_data` method in the `CustomDataset` class:
  - It should handle different dataset formats (CSV files) by providing a general mechanism for reading and preprocessing the data.
  - Note that the format and structure of the dataset may vary, so ensure your implementation is flexible.
- Implement the `split_data` method in the `CustomDataset` class:
  - Split the data into training, validation, and test sets based on the ratios provided in the `args` object. You can use the `ratio_train`, `ratio_val`, and `ratio_test` attributes from the base class.

2. `data_visualize` Function:

- Implement the `data_visualize` method in `data_visualizer`
- This function should take two parameters:
  - `dataset`: Dataset to visualize. Dataset.data might have shape (n_samples, timesteps, channels) or (n_samples, timesteps), depending on whether the dataset has multiple channels.
  - `t`: An integer representing the number of continuous time points to visualize.

- Ensure that your implementation is flexible enough to handle both single-channel and multi-channel datasets.

3. Testing:

- Test your implementation by creating an instance of the `CustomDataset` class and loading a custom dataset using the `get_dataset` function.
- Verify that the data is read and split correctly by accessing the relevant attributes of the dataset object (e.g., `dataset.train_data`, `dataset.val_data`, `dataset.test_data`).
- After loading the dataset, call the `data_visualize` function on the dataset object with a specific value of `t` to visualize the data.

In your report, plot one of the provided datasets and analyze the dataset's properties briefly according to the images. You can design the way you plot your dataset freely to make the images more beautiful or easier to analyze.

Part 2. Transform (20 pts)

path: `src/utils/transforms.py`

Objective: In this part, you will implement various data transformation techniques for preprocessing time series data. These transformations will help prepare the data for machine learning models. You will create custom transformation classes that inherit from the `Transform` base class and implement the necessary methods.

Instructions:

1. Transform Base Class:

You are provided with a base class named `Transform`, which defines the structure for data transformations. This class has two abstract methods: `transform` and `inverse_transform`. You will create custom transformation classes that inherit from this base class and implement these methods.

2. Custom Transformation Classes:

You need to implement the following custom transformation classes, each inheriting from the `Transform` base class:

a. Normalization

b. Standardization

c. MeanNormalization

d. BoxCox (input might less than 0)

In your report, write down the mathematical formula for each transformation.

Part 3 Metrics (20 pts)

path: `src/utils/metrics.py`

Objective: In this programming assignment, you will implement several metrics commonly used to evaluate the performance of predictive models. These metrics will help you assess the accuracy and quality of predictions made by models. Your task is to implement the Mean Absolute Error (MAE), Mean Absolute Percentage Error (MAPE), Symmetric Mean Absolute Percentage Error

(SMAPE), and Mean Absolute Scaled Error (MASE) metrics.

In your report, write down the mathematical formula for each metric.

Part 4 Models (30 pts)

path: `src/models/baselines.py`

Objective: In this part, you will design and implement two forecasting models: Linear Regression and Exponential Smoothing. These models will take historical time series data of length `seq_len` as input and generate predictions of length `pred_len`.

Instructions:

1. MLForecastModel Base Class:

You are provided with a base class named `MLForecastModel`, which defines the structure for forecasting models. This class includes methods for training (`fit`) and forecasting (`forecast`). You will create custom forecasting model classes that inherit from this base class and implement the `_fit` and `_forecast` methods.

2. Custom Models:

a. LR(MLForecastModel): Linear Regression

b. ES(MLForecastModel): Exponential Smoothing

3. Testing:

- Test both `LR` and `ES` on the dataset you plot in Part 1.
- Load the selected datasets, split them into training and testing sets, and apply each forecasting model to make predictions on the testing data.
- Calculate and record relevant metrics for each model and dataset pair, including Mean Absolute Error (MAE), Mean Absolute Percentage Error (MAPE), Symmetric Mean Absolute Percentage Error (SMAPE), and Mean Absolute Scaled Error (MASE).
- Present the evaluation metrics in a table format, showcasing the performance of each model on each dataset.

In your report, write down the mathematical formula for each algorithm, test the algorithm with different transformations on the dataset you plot in Part 1 and fill the table below.

Dataset	Model	Transform	MSE	MAE	MAPE	SMAPE	MASE
dataset	LR	None					
		Normalize					
		Box-Cox					
		...					
	ES	None					
		Normalize					
		Box-Cox					

Dataset	Model	Transform	MSE	MAE	MAPE	SMAPE	MASE
		...					

Part 5 TsfKNN* (bonus 20pts)

path: `src/models/TsfKNN.py`

objective: In this part, you will work with a basic version of the K-Nearest Neighbors (KNN) Time Series Forecasting Model (`TsfKNN`). Your goal is to enhance and improve the model's performance by implementing various improvements. The initial model takes historical time series data as input and generates predictions of length `pred_len` using the KNN algorithm. You will explore two main areas for improvement: enhancing distance calculation and reducing computational complexity. A simple introduction to this method can be found here.

Instructions:

1. Current Implementation:

You are provided with a basic implementation of the `TsfKNN` class, which uses the KNN algorithm for time series forecasting. The model currently supports two options: `MIMO` and `recursive` for multi-step ahead forecasting.

2. Distance Calculation Enhancement:

a. Explore Different Distance Metrics:

- Currently, the model uses the Euclidean distance (`euclidean`) for KNN. Enhance the model by allowing the use of different distance metrics such as Manhattan distance, Chebyshev distance, or others.
- Implement distance functions for these metrics and provide an option for users to choose the desired distance metric when creating an instance of the `TsfKNN` class.

b. Experiment with Custom Distance Metrics:

- Explore the use of custom distance metrics tailored to the characteristics of time series data. Custom distance metrics might consider factors such as trend, seasonality, or cyclic patterns.
- Implement one or more custom distance metrics and allow users to choose them as an option when creating an instance of the `TsfKNN` class.

3. Computational Complexity Reduction:

a. Approximate KNN Search:

- The current implementation uses an exact KNN search, which can be computationally expensive for large datasets. Investigate and implement approximate KNN search algorithms, such as locality-sensitive hashing (LSH), to reduce the computational complexity while maintaining reasonable accuracy.
- Provide options for users to enable or disable approximate KNN search when creating an instance of the `TsfKNN` class.

4. Testing:

- Test the enhanced `TsfKNN` model on 1-2 of the real-world time series datasets, such as ETT and Custom datasets, to evaluate its performance.
- Use evaluation metrics (e.g., MAE, MAPE, SMAPE, MASE) to measure the model's accuracy and compare it with the basic version of the model.
- Compare the computational time required for various operations between the basic and enhanced versions.

In your report, include the detailed implement of your algorithm and comparison of time and accuracy in 1-2 of the provided datasets.

Submission

1. Modified Code:

- Provide the modified code for all components of the task.
- Include a `README.md` file in Markdown format that covers the entire task. This file should contain:
  - how to install any necessary dependencies for the entire task.
  - how to run the code and scripts to reproduce the reported results.
  - datasets used for testing in all parts of the task.

2. PDF Report:

- Create a detailed PDF report that encompasses the entire task. The report should include sections for each component of the task.

3. Submission Format:

- Submit the entire task, including all code and scripts, along with the `README.md` file and the PDF report, in a compressed archive (.zip).

4. Submission Deadline:
2023-10-24 23:55