DOCUMENTATION

Finding forecast metric scores of Real Data

-Purvi, Santhosh

Contents:

Section1: Approach

Preprocessing the data: distributing datasets for training and testing
Then train the data, use this trained model on the testing data to find the Probabilistic Estimate of the load (Probabilistic Forecast), we can then deploy this forecast information or statistics to calculate the performance metrics.

Preprocess the Data —> Train the Data —> Use Trained Model to find Forecasts —> Use this Forecast to find the metric scores.

Purvi:

These are the few takeaways from the modules provided in the git.

- GitHub Actions Integration:
- This integration ensures the Hugo-based documentation site is always up-to-date by automatically triggering updates whenever changes are pushed to the master branch, streamlining the development and documentation process.

- Project Focus:
- designed for probabilistic load forecasting, specifically targeting hourly predictions of German power consumption. This framework leverages advanced data handling and machine learning techniques to produce accurate and reliable forecasts.

- Data Handling:
- The data preprocessing step is crucial for ensuring the quality and reliability of the forecasting model. Data is imported from CSV files, missing values are interpolated, and timestamps are aligned, all of which prepare the dataset for effective model training.

- Training Process:
- The neural network model is trained over 50 epochs with early stopping to prevent overfitting, optimising the model's performance.

- Evaluation Metrics:
- The project uses Mean Absolute Error (MAE) for its simplicity and Mean Absolute Scaled Error (MASE) for scale-independent comparisons. It avoids percentage error metrics to prevent inaccuracies when forecast or true values are zero.

- Probabilistic metrics like Quantile Score, Interval Score, Continuous Ranked Probability Score (CRPS), and Ignorance Score provide a comprehensive evaluation of the predictive distributions, ensuring forecasts are accurate and reliable.

- Probabilistic Forecast Properties:
- Reliability, resolution, and sharpness are key properties assessed to ensure the forecasts are consistent, informative, and tailored to specific cases.

- Configuration Adjustments:

- Adjusting training parameters in configuration files allows for quick but less accurate model training, which is useful for technical runs where speed is prioritised over precision. This flexibility supports various stages of model development and testing.

- Environment and Deployment:

- The pipeline automates the entire process from environment setup to model training, evaluation, and deployment. This automation ensures a seamless workflow and consistent deployment of results to the documentation site, enhancing both developer efficiency and end-user experience.

Section 2: Finding metric scores of Real Data

To make **Preprocess.py** work add this in **datahandler.py**:

```
import ssl
ssl._create_default_https_context = ssl._create_unverified_context
```

Train.py:

Was taking too long than expected due to the size of dataset, slowness of epochs and my Macbook's GPU

Problem with finding a Module : ModuleNotFoundError: No module named 'proloaf.event_logging'

Section 3: Conclusion and Future Proceedings with Stage1.

The quality of the scores we got and how we can containerise them in Kubernetes Clusters using docker files.

The initial phase of our project has focused on the implementation and evaluation of probabilistic load forecasting using the ProLoaF framework. The quality of the scores obtained from our performance metrics indicates that our forecasting models are delivering satisfactory results. These metrics—Root Mean Square Error (RMSE), Sharpness, Prediction Interval Coverage Probability (PICP), and Mean Interval Score (MIS)—are crucial for evaluating the accuracy and reliability of our forecasts.

Performance Metrics:

- The performance evaluation will provide metrics scores such as RMSE, sharpness, PICP, and MIS to assess the accuracy and reliability of our forecasts.
- Example output:

  RMSE: 50.3

  Sharpness: 2.1

  PICP: 0.95

  MIS: 1.7

Containerization with Kubernetes and Docker:

To facilitate the deployment and scalability of our forecasting models, we plan to containerize our application using Docker and orchestrate it with Kubernetes. Containerization will ensure that our application runs consistently across different environments, while Kubernetes will provide the necessary infrastructure for managing these containers in a cluster.

Steps for Containerization:

1. **Create Dockerfiles**: We will create Dockerfiles for each component of our forecasting system, including data preprocessing, model training, and prediction services.
2. **Build Docker Images**: Build the Docker images using the Dockerfiles.
3. Push Docker Images to a Container Registry: Push the built images to a container registry such as Docker Hub or a private registry.
4. **Deploy with Kubernetes**: Create Kubernetes deployment and service YAML files to deploy these Docker containers within a Kubernetes cluster.
5. **Monitor and Scale**: Use Kubernetes' built-in monitoring and scaling capabilities to ensure the application can handle varying loads efficiently.

Future Proceedings

Moving forward, the next stages will involve:

- **Enhanced Model Development:** Continuously refining our forecasting models to improve the performance metrics.
- **Integration with EMS and Control Systems:** Ensuring proper use of the forecasting models with Energy Management Systems (EMS) and other control tasks.
- **Scalability and Robustness:** Further testing the scalability and efficiency of our containerized applications within Kubernetes clusters.
- **Real-time Data Integration:** Using real-time data streams for more accurate and timely forecasts.

Sources used so far:
https://data.rte-france.com/catalog/-/api/generation/Generation-Forecast/v2.1