Stage 2 ML Project

Thomas DANIEL Théo CHICHERY Jean MARCHEGAY Julien DE VOS

November 29, 2024
DIA 3

Contents

1. Previous Methods and Limitations

In the first stage of the project, two primary regression models were explored: Linear Regression and k-Nearest Neighbors (KNN). These models were applied to the dataset, which consists of variables like `load`,`Gb(i)`,`Gd(i)`, `P`, `T2m`, `H_sun`, `WS10m`, and `residual_load`.

1.1. Linear Regression

Linear Regression was employed to model the relationship between the features and the target variable. The model optimizes the equation:

$$Y = B_0 + B_1X_1 + B_2X_2 + \cdots + B_nX_n$$

to minimize the difference between actual and predicted values. While straightforward and interpretable, Linear Regression has the following limitations:

- **Linearity**: It assumes linear relationships between features and the target, which limits its ability to model non-linear patterns.
- **Feature Selection**: Poorly chosen features significantly impact prediction quality.
- **Sensitivity to Outliers**: Outliers can skew the model, leading to suboptimal predictions.

1.2. k-Nearest Neighbors (KNN)

KNN regression predicts a target value based on the values of its nearest neighbors. It relies on calculating distances between data points and choosing a predefined number of neighbors (k). KNN's limitations include:

- **Computational Complexity**: Predictions require recalculating distances for each query, which becomes inefficient for large datasets.
- **Hyperparameter Sensitivity**: The choice of k significantly affects model performance, with small values prone to overfitting and large values prone to underfitting.
- **Model Reusability**: The lack of a fixed model requires recalculating predictions for every query.

Despite their simplicity, these methods provided a strong baseline for the project.

2. Improvement Assumptions

Based on the limitations of previous methods and the nature of the dataset, the following improvements are proposed:

2.1. Residual Load Calculation

Instead of predicting `residual_load` directly, it can be computed as:

$$\texttt{residual\_load} = \texttt{load} - \texttt{P}$$

This approach allows the model to focus on predicting `load` and `P` separately, leveraging tailored models for each. This decomposition is expected to enhance the overall accuracy of predictions by addressing the distinct patterns in these variables.

2.2. Advanced Modeling Techniques

To better capture the temporal and multivariate characteristics of the dataset, the following models will be explored:

- **Tree-Based Models**: Techniques like XGBoost or LightGBM can handle non-linear relationships and interactions among features effectively.
- **Deep Learning**: Neural Networks, including LSTM, will be considered for their ability to capture temporal dependencies in time series data.

By addressing the weaknesses of previous methods and leveraging these improvements, the accuracy and robustness of predictions can be significantly enhanced.

3. Problem Formalisation and Methods

3.1. New Algorithm Description

3.1.1 XGBoost

XGBoost (Extreme Gradient Boosting) is a powerful and efficient tree-based algorithm that is well-suited for tabular data like this dataset. Its relevance lies in the following:

- **Handling Non-Linear Relationships**: XGBoost can capture complex non-linear relationships between variables such as `load`, `residual_load`, and weather features (`T2m`, `WS10m`, `H_sun`).
- **Feature Importance**: It provides feature importance metrics, which help identify the most influential variables for predicting `residual_load`.
- **Efficiency and Scalability**: XGBoost is computationally efficient and handles large datasets with ease, making it suitable for datasets with multiple time steps like this one.

However, XGBoost treats data as independent instances and does not inherently account for the temporal relationships in the data, which limits its ability to model sequential patterns effectively.

3.1.2 Neural Network

Artificial Neural Networks (ANNs) are flexible models capable of approximating complex functions. For this dataset, their relevance includes:

- **Non-Linearity Modeling**: Neural Networks excel at modeling non-linear relationships, making them effective for predicting `load` based on multiple features (`T2m`, `H_sun`, etc.).
- **Custom Architectures**: A neural network can be designed to handle both temporal and non-temporal features, such as combining weather data and time.
- **Scalability**: With sufficient data and computational power, neural networks can outperform traditional machine learning models.

While ANNs are powerful, they require careful hyperparameter tuning and are prone to overfitting, especially on small datasets.

3.1.3 LSTM

Long Short-Term Memory (LSTM) networks are a type of Recurrent Neural Network (RNN) specifically designed to handle sequential data, making them highly relevant for this dataset:

- **Temporal Dependencies**: LSTMs excel at capturing temporal patterns in sequential data, such as the dependency of `residual_load` on previous time steps and trends.
- **Multivariate Time Series**: They can process multiple features (`P`, `Gb(i)`, `Gd(i)`, `H_sun`, etc.) simultaneously to predict `load`, `residual_load` or P.
- **Robustness to Long-Term Dependencies**: Unlike traditional RNNs, LSTMs can model long-term dependencies, which are crucial for forecasting tasks.

LSTMs are more computationally intensive than XGBoost and require a larger dataset to achieve optimal performance, but they are ideal for capturing both temporal and multivariate aspects of this dataset.

3.1.4 Relevance of Models for This Dataset

The dataset contains:

- **Time Feature (`time`)**: Indicates temporal ordering, critical for models like LSTM.
- **Weather Variables (`T2m`, `WS10m`, `H_sun`)**: These provide environmental context for predicting `load`, `residual_load` and P.
- **Energy Variables (`P`, `Gb(i)`, `Gd(i)`)**: These are predictors of energy consumption or generation.

Each model has unique strengths:

- **XGBoost** is effective for quick and interpretable results on tabular data.

- **Neural Networks** offer flexible architectures for multivariate modeling.
- **LSTMs** are essential for capturing temporal dependencies and trends in the dataset.

3.2. Limitations

XGBoost

- Can't extrapolate as it's based on a tree principle.
- The model may be prone to overfitting if not properly parameterized.
- Time-consuming training on large datasets.

Neural Network and LSTM

- The model may tend to overfitting if the dataset is too small.
- Training LSTMs requires a lot of resources, especially in terms of memory and time.
- The model is difficult to interpret.

4. Methodology

4.1. New Algorithm Implementation and Hyperparameters

P and Load predictions

We start by implementing a function to test different parameters on different models, using the predictions of P and load to find the residual load.

To do this we use a train_test_split which takes as parameters the column of variables to be used to make predictions as well as the 2 columns to be predicted in our case P and load. We then run a loop that tests all our models and hyperparameters.

For the random forest we test the parameters :

- n estimators with 50, 100, 200.
- max depth with 5, 10, 20
- min samples split with 2, 5, 10

For LinearRegression we test the model with fitintercept on or off.

Finally for the XGBoost we test the parameters :

- n estimators with 50, 100.
- learning rate with 0.01, 0.1, 0.2
- max depth with 3, 6, 8, 10

To make these results accessible, they are saved in a dictionary with the model name as the key.

P and load prediction improvements

We then integrated a function (model repeat) to improve our dataframe in order to get better results, especially on the predictions of P. To do this, the function takes as parameters the number of times the algorithm will rotate the column to be predicted, the variables, the model, the dataframe and an error variable chosen arbitrarily to remove only undesirable data.

The function model repeat(model, rounds, X cols, y col, df, error multiplier): will use a model to predict y col with X cols, then calculate the RMSE accuracy of the model for each row, finally it will check that the prediction of X cols does not exceed the average RMSE multiplied by our error multiplier to leave it in the dataframe. This is why the value must not be too low, otherwise it will remove too many columns, or if it's too high, it will keep all the values, even the undesirable ones.

Python Function: `model_repeat`

Here is the Python implementation of the `model_repeat` function:

```
def model_repeat(model, rounds, X_cols, y_col, df, error_multiplier):

    scores = []
    length = []

    for round in range(rounds):

        # Number of rows for iteration
        length.append(len(df.index))

        # Set X and y columns and df for iteration and calculate mean
            rmse
        X = df.loc[:, X_cols]
        y = df.loc[:, y_col]
        scores.append(np.mean(np.absolute(
            cross_val_score(model, X, y,
                scoring='neg_root_mean_squared_error', cv=3))))

        # Fit model and generate predictions, squared error for each
            prediction
        model.fit(X, y)
        df.loc[:, 'y_pred'] = model.predict(X)

        df.loc[:, 'sq_error'] = np.square(df.loc[:, 'y_pred'] - y)
        sq_error_threshold = error_multiplier * df.loc[:,
            'sq_error'].mean()

        df = df.loc[df.sq_error < sq_error_threshold]

    return df, scores, length
```

Once this method is implemented, we create a function to test with different prediction models in the repeat to find the best one to remove our useless data.

Now that we have a method for improving our dataframe, we choose to modify the columns used to predict P and load.

Since P and load are predicted separately, and P represents the amount of energy generated by the photovoltaic system, we can use only the data related to the sun :

- H sun : Sun height (elevation).
- Gd(i) : It is the fraction of solar radiation that reaches the ground after being reflected or scattered by the atmosphere.
- Gb(i) : It is the fraction of the solar radiation that directly reaches the ground.

Since Gb(i) and Gd(i) are related, we add them together to predict P

As load represents the energy used to operate the solar panel, it does not depend on the sun but on climatic conditions and time, mainly air temperature and wind. To predict load we mainly use :

- T2M : Air temperature at 2m
- WS10m : Wind speed at 10m

In addition, as load depends on the time of day, we use date to retrieve the hour, day, and year that we create in each column.

Now that we've processed our dataframe, we'll make various predictions about P and load

LSTM

To complete Stage 2, we implemented an LSTM to manage time series. Before using the model, we need to prepare the data, creating sequences for x and y.
To do this, we create a create sequence function that takes into account the number of sequences required and the dataframe, and then loops through to create the sequences of X and y associated with them.

Now that the data is ready, we create the LSTM model, which is configured as follows :

Model Summary

Layer (type)	**Output Shape**	**Param #**
lstm_13 (LSTM)	(None, 6, 64)	17,152
dropout_13 (Dropout)	(None, 6, 64)	0
lstm_14 (LSTM)	(None, 32)	12,416
dropout_14 (Dropout)	(None, 32)	0
dense_6 (Dense)	(None, 1)	33

Total params: 29,601 (115.63 KB)
Trainable params: 29,601 (115.63 KB)

We choose dense (1) because we have a regression project, dropout(0,2) disable 20% of neurons during training to avoid overfitting, we always put activation in relay because this filter then allows the model to focus only on certain characteristics of the data, while eliminating others. Finally 64 and 32 are the number of neurons used.

We then look at the model's accuracy

5. Results

5.1. Metrics

To measure the effectiveness of our models, we'll use two metrics: R^2 and negative root mean squared error.

The R^2 formula is:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

Where:

- y_i = Actual values
- $\hat{y}_i$ = Predicted values
- $\bar{y}$ = Mean of actual values
- n = Number of data points

The closer the R^2 score is to 1, the better the model; for 0, the model is equivalent to the mean value; for less than 0, the model is worse than the mean.

The RMSE (Root Mean Squared Error) formula is:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Where:

- y_i = Actual values
- $\hat{y}_i$ = Predicted values
- n = Number of data points

This gives us a number that represents the difference between our predicted values and the actual values. The lower this number is, the more accurate our model is, and it gives us an idea of how far our predicted values deviate from the actual ones.

These two metrics give a different view of our results and the accuracy of our models. In addition, the RMSE is the standard metric used in the Kaggle competition associated with our data. In the next stages, we will use this metric to measure our models and to submit solutions on the Kaggle leaderboard.

5.2. Overfitting

5.2.1 XGBoost

In XGBoost, overfitting can occur when the model is excessively complex, often due to too many trees, high tree depth, or inadequate regularization. This results in the model memorizing the training data, including noise, rather than capturing the underlying patterns. Overfitting in XGBoost can lead to poor performance on unseen data despite excellent results on the training set.

5.2.2 Random Forest

In Random Forest, overfitting can happen if the trees are grown too deep, allowing the model to capture noise and anomalies in the training data. While Random Forests inherently reduce overfitting compared to a single decision tree by averaging multiple trees, they can still overfit if the number of trees is too large or the individual trees are overly complex.

5.2.3 LightGBM

In LightGBM, overfitting can arise when the model is trained with excessive iterations, high leaf complexity, or without sufficient regularization. This results in the model focusing too much on specific patterns or noise in the training data. LightGBM's speed and flexibility make it particularly prone to overfitting if hyperparameters like max depth or learning rate are not carefully tuned.

5.3. Mitigating Overfitting

5.3.1 Train and Test Sets

The principle of using train and test sets is fundamental to evaluating model performance. The training set is used to fit the model, allowing it to learn from the data. The test set is kept separate and used to assess the model's ability to generalize to new, unseen data. This separation ensures that the model is not just memorizing the training data but instead is learning patterns that can be applied to other datasets.

5.3.2 Cross-Validation

Cross-validation is a technique used to evaluate a model's performance on different subsets of the data. The dataset is divided into several smaller subsets, or "folds." The model is trained on some folds and tested on the remaining ones. This process is repeated multiple times to ensure that the model performs well across different portions of the data, helping to prevent overfitting.

5.3.3 Grid Search

Grid search is used to find the optimal hyperparameters for a model by testing a range of different values systematically. For example, in linear regression, grid search might test different regularization strengths, while in KNN, it could test different values for k. By finding the best combination of parameters, grid search helps to improve model performance and reduce the risk of overfitting.

5.4. Evaluation and Comparison with Previous Solutions

Model	**5-Folds RMSE**	**Kaggle Score**
XGBoost (residual directly)	54.11	62.78
Simple NN (residual directly)	60.79	67.23
XGBoost/LightGBM (Load-P approach)	17.01 (P), 23.77 (Load)	47.61 (Top 3 leaderboard)
LSTM (residual directly)	67.76	Not Submitted

Table 1: Summary of Model Results and Performance

The results highlight notable performance improvements achieved through careful model selection and optimization. The XGBoost/LightGBM (Load-P approach) demonstrated the best performance, achieving a remarkably low RMSE of 17.01 for predicting P and 23.77 for predicting the load across 5-fold cross-validation. This approach also secured a strong Kaggle score of 47.61, ranking among the top 3 on the leaderboard, indicating its robustness in real-world scenarios.

In comparison, the XGBoost model directly predicting residuals achieved an RMSE of 54.11 across 5-folds, with a Kaggle score of 62.78, showcasing decent performance but not as competitive. Similarly, the Simple Neural Network (NN) showed slightly lower performance, with a 5-fold RMSE of 60.79 and a Kaggle score of 67.23.

The LSTM model, while tested, yielded a high RMSE of 67.76, making it unsuitable for submission. Overall, the Load-P approach with XGBoost/LightGBM effectively leveraged domain-specific insights to outperform other methods by a significant margin.

6. Discussion and Conclusion

While we have found a very satisfying result using XGBoost/LightGBM models and load-P approach, a few ideas can be explored to further improve the results:

- Time-Series Analysis: We can continue to explore models that consider more the temporal dependencies of the data. In addition to long short-term memory networks (LSTM), we can explore techniques like autoregressive models (such as ARIMA) or time-series cross-validation to improve predictions.
- Deep Learning Approach: Some deep learning techniques like recurrent neural networks (RNNs) and convolutional neural networks (CNNs) are becoming more popular in time-series analysis and could be useful for our project. These models

are good for understanding relationships and patterns in the data over time, which can lead to more precise predictions.