AML assignment 2 report

Roel Duijsings s1086375

# Data analysis

## inbound\_loads.csv:

* all weight\_uom values are in pounds (or NaN)
* 17756 different trailers in use. Some are used >267 times, others only once.
* During loading, the dock door opens, so the indoor temperature gets mixed with the outdoor temperature.

weather.csv

* Contain datetime, temperature and humidity.
* Weather is being measured only once every five minutes!!
* Connect datetime of demand\_kW with weather.

## demand\_kWtrain\_val.csv

* The train data ranges from 2018-12-31 21:15:00 to 2021-10-11 06:05:00.
* The validation data ranges from 2021-10-11 06:08:00 to 2021-12-13 17:59:00.
  + Therefore the validation data is only winter period, so expect lower temperatures! and therefore lower demand? Is the data representative?
* I found that the datetime objects in column ‘datetime\_local’ are not sorted in order. For some reason time goes back 1 hour at 2021-11-07 01:59:00 (see table below). This influence the use of pd.merge\_asof() which I use to merge the weather with the demand, since not all demand datetimes are represented in the weather file and therefore, I pick the nearest weather.

|  |  |
| --- | --- |
|  | datetime\_local |
| 312568 | 2021-11-07 01:58:00 |
| 312569 | 2021-11-07 01:59:00 |
| 312570 | 2021-11-07 01:00:00 |
| 312571 | 2021-11-07 01:01:00 |
| 312572 | 2021-11-07 01:02:00 |

## Pallet\_history\_Gold\_Spike.csv



After we connected df\_demand with df\_weather: df\_demand\_train.describe() (see table below).

* We see that demand\_kW ranges from +- 300 to 3300 (a quite wide range), but most points range between 1500-2500. Pretty close to the mean, so you could say data is clustered around the mean.
* Relative humidity ranges from 9% to 100% (is 100% even possible?), with 95% of points between 28% and 100%. I expect there might be some errors in measuring relative humidity, because there are 6991 rows with 100% humidity and the second highest Relative Humidity is 97.05%, which is a big gap compared to the gaps between other measurements! Should I remove these outliers? Perhaps this is due the fact that it was raining and the sensor got wet. I see that the Relative Humidity stays 100% for a couple of measurements in a row, on non-consecutive days. Which implies no technical error.
* Temperature also clusters around the mean pretty much.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | demand\_kW | datetime | Relative Humidity | Temperature |
| count | 117028.000000 | 117028 | 116241.000000 | 116877.000000 |
| mean | 2187.339130 | 2020-08-23 12:07:18 | 66.930375 | 70.062610 |
| min | 294.851640 | 2018-12-31 21:15:00 | 9.430000 | -2.200000 |
| 25% | 1806.044864 | 2019-11-05 10:41:15 | 52.030000 | 60.800000 |
| 50% | 2252.156000 | 2020-09-20 16:37:30 | 68.340000 | 73.400000 |
| 75% | 2572.752635 | 2021-06-30 23:26:15 | 83.130000 | 82.400000 |
| max | 3294.331000 | 2021-10-11 06:05:00 | 100.000000 | 107.600000 |
| std | 483.618044 | NaN | 20.114604 | 16.745439 |

# Thinking process

First of all, I read through the assignment on Bright Space. Then, I scanned the data files that were provided.

After, scanning the demand\_kWtrain\_val.csv file, I realized that the only input is the datetime and the output is a float of the demand in energy in kW.

This datetime input can be connected to multiple pieces of information, such as:

* the weather at that datetime (temperature, humidity)
* the outbound/inbound loads

# Problems

* What if a datetime in demand\_valition is not in weather.csv? Should we take the temperature of nearby datetime? Or perhaps last years temperature? Or the average temperature for that hour? I would say, pick the nearest temperature, since that is more representative for the real temperature than last year’s.
* Do we drop the

# Raw validation results

This sections describes the intermediate results of the XGBoost classifier for a given set of data. The training dataset is split into 80% training and 20% validation data.

## Temperature and relative humidity

Validation statistics:

|  |  |
| --- | --- |
| MSE | 85151.657 |
| RootMSE | 291.807 |
| Mean value y\_test | 2405.976 |
| RelRootMSE | 0.1212 == 12.1% (good) |

12.1% Relative Root Mean Squared Error is a decent score for such a small set of features, but could be improved.

## Temperature, relative humidity and 5 hours load\_duration

Validation statistics:

|  |  |
| --- | --- |
| MSE | 64101.71946819282 |
| RootMSE | 253.1831737461888 |
| Mean value y\_test | 2403.664279856949 |
| RelRootMSE | 0.10533216966608026 == 10.5% (good) |

## Temperature, relative humidity and 2 hours load\_duration

Validation statistics:

|  |  |
| --- | --- |
| MSE | 66095.18651509732 |
| RootMSE | 257.0898413300248 |
| Mean value y\_test | 2405.495297978833 |
| RelRootMSE | 0.10687605232321142 == 10.7% (good) |

# Model parameters

To further improve the raw model (without any parameters), I analyzed the hyperparameters. This was done with this setup

model = XGBRegressor(

    n\_estimators=1000,

    learning\_rate=0.05,

    max\_depth=10,

    min\_child\_weight=1,

    early\_stopping\_rounds=5,

    random\_state=42

)

With the next ranges:

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Min value** | **Max value** |
| n\_estimators |  |  |
| Learning\_rate |  |  |
| Max\_depth |  |  |
| Min\_child\_weights |  |  |
| Early\_stopping\_round |  |  |
| Random\_state |  |  |

# Model with parameters results

## Result 1

model = XGBRegressor(

    n\_estimators=1000,

    learning\_rate=0.05,

    max\_depth=10,

    min\_child\_weight=1,

    early\_stopping\_rounds=5,

    random\_state=42

)

|  |  |
| --- | --- |
| Training score | 0.7999563254922486 |
| MSE | 48971.96376207384 |
| RootMSE | 221.29609974437832 |
| Mean value y\_test | 2405.6100317815653 |
| RelRootMSE | 0.09199167646490447 == 9.2% (very good) |

## Result 2

model = XGBRegressor(

    n\_estimators=1000,

    learning\_rate=0.1,

    max\_depth=10,

    min\_child\_weight=1,

    early\_stopping\_rounds=5,

    random\_state=42

)

|  |  |
| --- | --- |
| Training score | 0.80349568413666966 |
| MSE | 46112.66519529044 |
| RootMSE | 214.73859735802142 |
| Mean value y\_test | 2407.6619983237088 |
| RelRootMSE | 0.08918967758245522 == 8.9% (very good) |

# Cross-validation

To get a better analysis of what are the best parameters to use, I use GridSearchCV and RandomizedSearchCV for cross-validation.

param\_grid = {

    'n\_estimators': [500, 1000, 1500],

    'learning\_rate': [0.1, 0.3],

    'max\_depth': [10],

    'min\_child\_weight': [1]

}

Which returns the parameters with the highest training score:

Best parameters: {'learning\_rate': 0.1, 'max\_depth': 10, 'min\_child\_weight': 1, 'n\_estimators': 1000}

Train score: 0.8293708324328936

# Feature importance

After fitting the model, we can get the importance of the features of the training data.

Afbeelding met tekst, schermopname, Rechthoek, Lettertype

Automatisch gegenereerde beschrijving

In this graph we can see that load duration and relative humidity are way more important than the outdoor temperature. This is something that I did not expect.

# Outliers

I can see that there are some outliers. I performed Outlier Detection using Z-score measure:

“*Z-score: The Z-score is a measure of how many standard deviations an element is from the mean. A commonly used rule says that a data point is an outlier if its z-score is above 3 or below -3.”*

I used a Z-score threshold of 3 on the column ‘demand\_kW’ on the training dataframe. This showed 1265 outliers. I wanted to know why they are picked as outliers, so I ran outliers.describe:

|  |  |
| --- | --- |
| count | 1265.000000 |
| mean | 778.283098 |
| std | 232.004757 |
| min | 294.851640 |
| 25% | 629.369354 |
| 50% | 848.160242 |
| 75% | 996.477000 |
| max | 1055.636464 |

What is interesting about this is that they all have low demand\_kW values. Maybe a bit too low? If that’s the case it might be due to bad measurements.

# Extra feature: Average demand of previous measurements

Now, one feature I just came up with is the average demand of the previous 10 measurements. As demand would not fluctuate too much in a short amount of time, I expect the current demand be close to the previous 10 demands.

Afbeelding met tekst, schermopname, lijn, diagram

Automatisch gegenereerde beschrijving

Afbeelding met tekst, schermopname, lijn, Perceel

Automatisch gegenereerde beschrijving

Figure 1: In this plot, the black dashed line represents the line of perfect predictions (i.e., where the true value equals the predicted value). The points represent the validation data: the x-coordinate of each point is the true value for that data point, and the y-coordinate is the predicted value. If the model would be perfect, all points would lie on this line. The more closely the points cluster around the line, the better the model is.

Problem

In the test dataset, there are no demand\_kW values. We can only use the train dataset demand\_kW values.. One solution: train the model without these avg\_demand values and predict demand\_kW for testdata. Next, with these predicted demand\_kW calculate the average and add this column to train data. Now train a second model and make new predictions.

## CHatGPT answer to the problem:

*Creating a feature such as the average demand of previous measurements can indeed be a good idea, especially in time-series problems like yours, where the assumption is that the current data point is often influenced by the past data points.*

*However, when it comes to using this feature with your test set, you're correct that you cannot directly calculate this feature for your test set because you don't have the `demand\_kW` values. What you're suggesting (creating the feature after an initial round of predictions) is a form of model stacking or ensemble learning, and it can potentially improve your results.*

*Here is a simplified version of your idea:*

*1.* ***First round of predictions****: Train a model (let's call it Model 1) on your training data without the `avg\_demand` feature. Use Model 1 to predict `demand\_kW` for your test data.*

*2.* ***Calculate the `avg\_demand` feature for your test data****: Using the predicted `demand\_kW` values from step 1, calculate the `avg\_demand` feature for your test data.*

*3.* ***Second round of predictions****: Add the calculated `avg\_demand` feature to your training data and train a new model (Model 2) on this expanded training data. Use Model 2 to predict `demand\_kW` for your test data.*

*This process is a little more complex than simply training a single model, but it can potentially lead to improved predictions because Model 2 has access to more (albeit predicted) information than Model 1. However, you should be aware that because the `avg\_demand` feature in your test data is calculated from predicted `demand\_kW` values (which may have errors), there is a risk of propagating these errors.*

*It's also important to cross-validate and assess your models properly. You can measure how much the second model improves over the first model, which will tell you if the extra complexity is worth it. You can also experiment with other techniques like different models, more features, and different ways of handling your data.*

# ChatGPT prompts

Given this dataset, what models should I consider and why?

Based on the given dataset, it appears that you are working with a time series data, where you want to predict the demand in kW based on the datetime, relative humidity and temperature.

Here are some models that you can consider for this problem:

1. ARIMA (AutoRegressive Integrated Moving Average): ARIMA is a popular model for time series forecasting. It takes into account the past values of the dependent variable, and predicts the future values based on that. It can handle trends and seasonality in the data.

2. SARIMAX (Seasonal AutoRegressive Integrated Moving-Average with eXogenous regressors): SARIMAX is an extension of ARIMA that can also handle seasonal patterns in the data. In addition to the past values of the dependent variable, it also takes into account the past values of the exogenous variables (in this case, relative humidity and temperature).

3. Prophet: Prophet is a time series forecasting model developed by Facebook. It is based on a decomposable time series model with three main components: trend, seasonality, and holidays. It can handle missing data and outliers in the data, and also incorporates the effects of external regressors.

4. LSTM (Long Short-Term Memory): LSTM is a type of neural network that can handle sequential data like time series. It is particularly good at capturing long-term dependencies in the data. LSTM can be used for both regression and classification tasks.

5. XGBoost (Extreme Gradient Boosting): XGBoost is a popular machine learning model that can handle both regression and classification problems. It is based on gradient boosting and uses decision trees as base learners. It can handle missing data and is known for its high accuracy and speed.

The choice of the model will depend on various factors such as the size of the dataset, the number of features, and the nature of the data. You may try multiple models and compare their performance based on some evaluation metrics such as mean squared error (MSE), root mean squared error (RMSE), mean absolute error (MAE), and R-squared score.

Why should I use XGBoost?

You should consider using XGBoost because it is a powerful and highly flexible machine learning algorithm that is well-suited for a wide range of regression and classification problems, including time series forecasting. Here are some reasons why you might want to use XGBoost for your dataset:

1. High accuracy: XGBoost is known for its high accuracy and has been used to win many machine learning competitions. It uses an ensemble of decision trees and gradient boosting to iteratively improve the model's performance.

2. Handles missing data: XGBoost can handle missing data by automatically learning how to best impute missing values during training.

3. Feature importance: XGBoost provides an estimate of feature importance, which can help you identify which variables are most important in predicting the target variable.

4. Regularization: XGBoost includes regularization techniques to prevent overfitting, which can improve the model's generalization performance.

5. Speed: XGBoost is designed to be computationally efficient, and can train models on large datasets in a relatively short amount of time.

Overall, XGBoost is a versatile algorithm that can handle a wide range of problems and is known for its high accuracy and speed. If you are working with a regression problem and want to try a powerful machine learning algorithm, then XGBoost could be a good choice for your dataset.

# XGboost

Uses Decision Tree Ensembles. The tree ensemble model consists of a set of classification and regression trees (CART)