Comparing different regression models to predict the price of electricity in Helsinki.

1. Introduction

Electricity is part of our everyday life and as technology advances, we become more and more dependent on it. We use it to make food, to warm ourselves, to keep us cool, to communicate etc. Electricity is produced with different energy production methods, which are some more and some less environment friendly. In addition to this I think everyone is familiar with electricity bills, the fee, we must pay for the electricity we have consumed. Change in access to raw materials for energy production and energy distribution might have an influence on the price of the electricity overall, which is very hard to predict since it has many factors.
Instead, we could predict the price with some other, simpler features.

2. Problem formulation

In this report I'm attempting to predict the electricity price with two different features: time and temperature. Regardless of the methods of energy production, I'm studying if the overall electricity price will alter by the change of temperature and daytime. These features are chosen because, for instance hydroelectric, could be more susceptible to temperature changes. In addition to this the price of electricity could be lower at night when usage, and thus demand, is lower.
As data points I'm going to have thousands of days from past years and I'm going to use certain times and the temperatures of these timepoints to predict the price of electricity (EUR/MWh) at that moment, which we are interested in.
In this area of interest, machine learning could be used for instance to predict at which specific time of the day it would be most profitable to use electricity to save money. We can also choose, which kind of electricity contract would be the most sensible one through different periods (cold, warm, night, day), because they also vary a lot.

2.1 Summary of the problem

As data points we will have days (from 1.1.2018 to 21.2.2022). Features will be certain time points, which will be presented as integers (09:00 = 9, 15:00 = 15, 21:00 = 21 and 03:00 = 3) and temperatures at these time points, which will be stated as decimal numbers. Label, y, will be the price of the electricity (EUR/MWh) in two decimal places (e.g., 10.55€) at different times.
The data will be found from [1] and [2].

Example of a possible data point including all the features and the label:

(Note that €/MWh is indeed 41.73 €, temperature is 10.4 Celsius and time is 11:00)

Date	Time	€/MWh	Temperature
19.4.2018	11	41.73	10.4

3. Methods

Data was collected for this project from two different platforms [1] and [2] for electricity prices from [1] and the data for temperature of each hour for the same period from [2].
The data used is from beginning of the year of 2018 to 2022 February. It's been collected from this time interval hourly. However, we are interested in only four different timestamps in a day six hours apart (03:00, 09:00, 15:00 and 21:00); This is because we are looking for possible varying on different times of set of days to get the change in price which occurs when people are at work, at home or sleeping for instance. In addition to this we don't want any unnecessary time points, which would have only a small impact, if all, on the problem. Therefore, we don't need to use every single time point of the day. In addition to this, electricity price dataset contains prices of different countries but since we are also interested in one (Finland, Helsinki), all the other countries' data apart from Finland will be pruned.
Overall, after filtering, we have approximately 6000 data points which we are using to get the best model to estimate our quantity of interest, electricity price in euros.

3.1 Splitting the data

Data points are divided into training (75%) and validation (25%) randomly.
We try to use as much data to train the model properly, so it would correspond to a real-world situation. Also, because in this situation we are investigating if the season of the year influences the price (since the temperature variation) we need to get plenty of data from every season of the year. Validation must have at least 25% to get a good approximate of the different situations, including the randomness, which may have an influence on the training set.

3.2 Models

For this ML problem I'm comparing two different models, polynomial regression (with degrees 4,5,6 and 7) and linear regression.
First approach and discussion over the problem gives intuitively a possibility to use regression model. Think about how electricity price, similarly to other products, changes due to supply and demand. During winter, the weather cools down and the need for various heating mechanisms increase. This is produced by energy which has an impact on electricity price. Therefore, there might be a chance that prices decrease linearly when temperature increases.

However, in the summer, during warm days electricity is consumed on cooling devices which might show as an increasement in electricity prices.
So, because of the statements mentioned above, I will train the model with both polynomial and linear regression to find out which correlates to the real-life situation. With polynomial regression, degrees 4,5,6 and 7 are compared since my discussion above concluded that in winter and in summer the prices might be higher and otherwise lower. This would form a parabola-like model with respectively low degree.
In addition to this, possible chance for overfitting and underfitting is decreased if the degree is correctly chosen.

3.3 Loss function

To minimize the empirical risk, I'm using mean absolute and mean squared error loss to choose the best model from the hypothesis space for both regression models. Mean squared error loss (MSE) is chosen since being a commonly used default loss function with linear and polynomial regressions and it's simple to use with Python with sklearn (scikit-learn).
Also, with MSE possible deviations will not be totally pruned which might be important for our estimation.

The second lost function to minimize the error is mean absolute loss (MAE). The main reason to choose this loss function as the other option is because of its features. Unlike MSE, it's slightly more robust towards outliers. Consequently, we will use one with more impact on outliers and one with less.

4. Results

After comparing both polynomial and regression models with two different loss functions, the difference in errors were significant. As discussed before, polynomial outcome worked genuinely better since the temperature extremes for one year increase the demand for electricity nonlinearly.
In both regression models, the preferable loss function was indeed MAE (mean absolute error), since the number of outliers was explicitly high. Because MSE squares the differences between label and feature, MSE was significantly larger than MAE.

Table 1 (Polynomial)

Mean Absolute Error

poly degree	polynomial_train_errors	polynomial_val_errors
4	9.458921	9.589877
5	9.500086	9.591257
6	9.489580	9.581673
7	9.489861	9.691516

Mean Squared Error

poly degree	polynomial_train_errors	polynomial_val_errors
4	218.498416	323.560268
5	217.816798	316.027392
6	217.795694	314.950921
7	215.336715	323.219467

Table 2 (Linear)

Mean Absolute Error		Mean Squared Error	
linear_train_errors	linear_val_errors	linear_train_errors	linear_val_errors
9.706409	9.79869	231.959827	339.493584

From the tables above, we can clearly tell that MAE is the correct loss function out of these two for this problem. Even though the MAE doesn't differ a lot between polynomial and linear regression models, it's still a clear choice having a lower error in polynomial regression. Consequently, with a lower error (lowest at degree 6) and ability to fit the data better than linear, since being an indirect line, we will choose polynomial regression with degree 6 to have the most efficient outcome.

Since we decided to use polynomial regression with mean absolute error as loss function and we stated that degree is most efficient to be 6, we train the model again with 50% of the data. Rest is divided 30% to validation and 20% to testing. Training must be the largest because it's the data we are choosing the model in the first place. By splitting the data in three parts, we can ensure that the same data won't be used twice, and we get the most realistic outcome. Now we train the model with MAE and get the result for test error in the table 3.

Table 3 (Polynomial test error)

poly degree	Polynomial_test_errors
6	10.625083

5. Conclusion

Prediction of electricity price requires a lot of research to choose the features correctly since it's dependent on various incidents. My approach in this problem with time and temperature was an intuitive point of view since thinking how electricity is consumed.
We chose MAE to be the right loss function for choosing the model and the results were quite positive (test error about 10) since electricity could be predicted with many other possible features.
This way we had some reasonable results with polynomial regression but also astonishingly high errors (with MSE) which indicates that there is room for plenty of improvements.
For instance, totally different kind of approach in features or adding more variables might be the next step to improve on this subject. Also, the pruning of the data could be done differently, perhaps with less time points for instance, since there were a high number of outliers which were influencing on training with MSE.
Consequently, a loss function even more robust to outliers could have a better outcome.

References

[1] www.nordpoolgroup.com/elspot-price-curves/

[2] www.ilmatieteenlaitos.fi/havaintojen-lataus

Machine learning: The Basics, Alexander Jung

www.wikipedia.org/wiki/Polynomial_regression

www.wikipedia.org/wiki/Mean_squared_error

www.scikit-learn.org

Polynomial

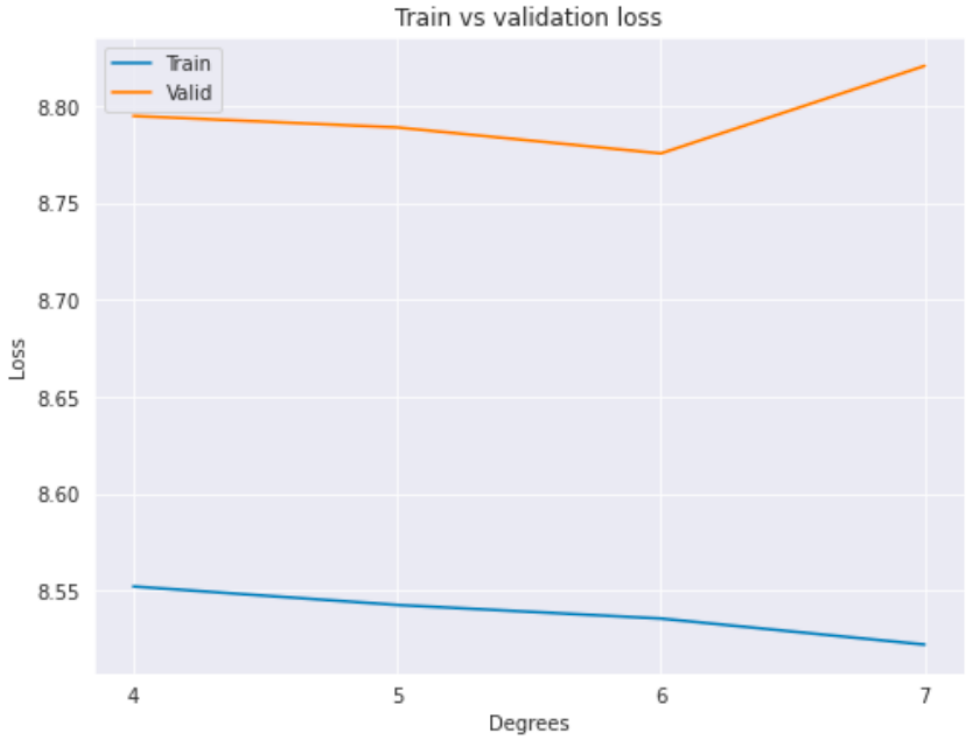
March 31, 2022

[10]:
```
%config Completer.use_jedi = False  # enable code auto-completion
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.metrics import mean_absolute_error
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split
import numpy as np #import numpy to work with arrays
import pandas as pd #import pandas to manipulate the dataset
from matplotlib import pyplot as plt #import the module matplotlib.pyplot to do␣
↪visulization
from sklearn.preprocessing import PolynomialFeatures    # function to generate␣
↪polynomial and interaction features
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, accuracy_score

import random
sns.set_style("darkgrid")

Price = pd.read_csv('Data.csv') # read the csv file which contains the data.

Price = Price.drop(['SYS','SE1','SE2','SE3','SE4','DK1','DK2','Oslo'], axis=1)␣
↪# drop unnecessary columns

Price.columns = ['Date','Time','€/MWh', 'Temperature'] # new names for columns

Price = Price.loc[((Price['Time'] == 3) |
                   (Price['Time'] == 9) |        # Choose only the specific time␣
↪points from each day
                   (Price['Time'] == 15) |
                   (Price['Time'] == 21))]
Price = Price.dropna()
Price
```

```
X = Price.drop(['Date', '€/MWh'], axis=1).to_numpy()    # features are␣
↪temperatures and time points
y = Price["€/MWh"].to_numpy()     # label is electricity price

print(X.shape)
print(y.shape)  # print out the shape of features and label.

# Split the dataset into a training set and a validation set
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.25,␣
↪random_state=42)

degrees = [4,5,6,7]

# we will use this variables to store the resulting training and validation␣
↪errors for each polynomial degree
linear_tr_errors = []
linear_val_errors = []
for degree in degrees:    # use for-loop to fit polynomial regression models␣
↪with different degrees
    lin_regr = LinearRegression(fit_intercept=False) # NOTE:␣
↪"fit_intercept=False" as we already have a constant iterm in the new feature␣
↪X_poly
    poly = PolynomialFeatures(degree=degree)    # generate polynomial features
    X_train_poly = poly.fit_transform(X_train)    # fit the raw features
    lin_regr.fit(X_train_poly, y_train)    # apply linear regression to these␣
↪new features and labels

    y_pred_train = lin_regr.predict(X_train_poly)    # predict using the linear␣
↪model
    tr_error = mean_absolute_error(y_train, y_pred_train)    # calculate the␣
↪training error
    X_val_poly = poly.transform(X_val) # transform the raw features for the␣
↪validation data
    y_pred_val = lin_regr.predict(X_val_poly) # predict values for the␣
↪validation data using the linear model
    val_error = mean_absolute_error(y_val, y_pred_val) # calculate the␣
↪validation error
    linear_tr_errors.append(tr_error)
    linear_val_errors.append(val_error)

errors = {"poly degree":degrees,
          "Polynomial_test_errors":linear_tr_errors,
          "Polynomial_val_errors":linear_val_errors,
         }
pd.DataFrame(errors)
```

```
pd.DataFrame({ key:pd.Series(value) for key, value in errors.items()})

plt.figure(figsize=(8, 6))
plt.plot(degrees, linear_tr_errors, label = 'Train')
plt.plot(degrees, linear_val_errors,label = 'Valid')
plt.xticks(degrees)
plt.legend(loc = 'upper left')

plt.xlabel('Degrees')
plt.ylabel('Loss')
plt.title('Train vs validation loss')
plt.show()

lin_degrees = [4,5]
print(X_train.shape)
print(y_train.shape)
regr = LinearRegression()
regr.fit(X,y)

## 2.Predict label values based on features and calculate the training error
y_pred_lin = regr.predict(X_train)
tr_error_lin = mean_absolute_error(y_pred_lin, y_train)

y_val_pred_lin = regr.predict(X_val)
tr_val_error_lin = mean_absolute_error(y_val_pred_lin, y_val)

errors_lin = {"poly degree":degrees,
          "linear_train_errors":tr_error_lin,
          "linear_val_errors":tr_val_error_lin,
         }
pd.DataFrame(errors_lin)
```

```
(1440, 2)
(1440,)
```




```
(1080, 2)
(1080,)
```

[10]:

	poly degree	linear_train_errors	linear_val_errors
0	4	9.484751	9.714967
1	5	9.484751	9.714967
2	6	9.484751	9.714967
3	7	9.484751	9.714967

[ ]:

[ ]:

```
print(X.shape)
print(y.shape)  # print out the shape of features and label.
degrees = 6
# Split the dataset into a training set and a validation set
X_train, X_rem, y_train, y_rem = train_test_split(X,y,test_size = 0.5,␣
 ↪random_state = 42)

X_val, X_test, y_val, y_test = train_test_split(X_rem, y_rem, test_size = 0.2,␣
 ↪random_state = 42)
lin_regr = LinearRegression(fit_intercept=False)
poly = PolynomialFeatures(degree=6)
X_train_poly = poly.fit_transform(X_train)     # fit the raw features
lin_regr.fit(X_train_poly, y_train)    # apply linear regression to these new␣
 ↪features and labels

## Polinomial transforming raw features of test set and evaluate model␣
 ↪performance on test data:
X_test_poly = poly.fit_transform(X_test) # transform the raw features for the␣
 ↪test data
y_pred_test = lin_regr.predict(X_test_poly) # predict values for the test data␣
 ↪using the linear model
test_error = mean_absolute_error(y_pred_test, y_test) # calculate the test error

errors = {"poly degree":degrees,
          "polynomial_test_error":test_error,
          "polynomial_val_error":test_error,
         }
print(test_error)
```

```
(1440, 2)
(1440,)
10.625083914961307
```

[ ]: