Applied Artificial Intelligence 2022/23
Lab 1 - Making Predictions with Regression

1. Background of the data and learning task

In this week's lab session, you will be working on the task of power prediction from an onshore wind farm in the north of Germany. The data records the power generated by the wind farm at regular intervals, alongside meteorological measurements such as the wind speed at different heights of a turbine's monopile. The original data and website can be found here:

https://bbdc.csl.uni-bremen.de/index.php/2018h/23-aufgabenstellung-2018

I have also uploaded the data folder to Canvas (files ending in "-en" have the English feature names, so you may want to use these as the original files are in German). There are train files for training, evaluation files for validation and challenge files for testing. The latter are less useful for us during this lab as they don't contain labels.

The training data provides wind measurements at 15-minute intervals from between January 2016 to June 2017 alongside the power output generated from the wind farm during that time. The evaluation data provides wind measurements for the second half of 2017 and requires prediction.

All data is available in the form of *.csv files with comma-separated values. For example:

```
Datum,Windgeschwindigkeit48M,Windgeschwindigkeit100M,Windgeschwindigkeit152M,Windrichtung48M,Windrichtung100M,Windrichtung152M,Windgeschwind
2016-01-01 00:00:00,8.49,10.77,12.69,188.0,190.0,194.0,9.07,9.63,10.06,10.43,10.78,11.05,11.54,11.94,12.42,0,122400.0,79168
2016-01-01 00:15:00,8.395,10.6175,12.475,190.75,193.0,196.75,9.0425,9.555,9.955,10.309999999999999,10.629999999999999,10.927500000000002,11.3
2016-01-01 00:30:00,8.3,10.465,12.26,193.5,196.0,199.5,9.015,9.48,9.850000000000001,10.19,10.48,10.805,11.235,11.69,12.205,1,122400.0,76072
```

The features are as follows:

- Date (Datum): Beginning of a 15-minute interval measurement (YYYY-MM-DD hh:mm:ss) in local time ME(S)Z.
- Wind_speed48M, Wind_speed100M, Wind_speed152M (Windgeschwindigkeit48M, Windgeschwindigkeit100M, Windgeschwindigkeit152M): wind speed at 48m height, 100m and 152 meters (measured in m/s)
- Wind_direction48M, Wind_direction100M, Wind_direction152M (Windrichtung48M, Windrichtung100M, Windrichtung152M): wind directions at 48m height, 100m and 152 (measured in degrees).
- Wind_speed_100MP10 - wind_speed_100MP90 (Windgeschwindigkeit100MP10 - Windgeschwindigkeit100MP90): Probabilistic wind speeds at heights 100m (10-90 percentile) in the current 15-minute interval (measured in m/s).

- Interpolated (Interpoliert): the wind measurements were originally taken at hourly intervals and were then interpolated to fit the 15-minute intervals. This field contains the interpolated values.

- available_capacity (Verfügbare_Kapazität): The maximum available power capacity of the wind farm currently, measured in kW. This can vary depending on maintenance or faults.

- Output: the produced wind energy (measured in kWh/h)

Some columns in output can be marked X, this means that the value is not available.

The full capacity of the entire wind farm is 122400kW — an hour of full capacity corresponds roughly to the energy needed to provide electricity for a 2-person household for 45 years.

2. Data Exploration

Using e.g. `pandas`, read in the train-en.csv data file and have a look at what you can see / observe. The goal of this first step is to learn more about the data, so that we can then model it better.

If you're running your programs in Google Colab today (this is where I tested it), don't forget to mount your drive first and upload the data.

```
from google.colab import drive
drive.mount('/content/gdrive')

import pandas as pd
import numpy as np

# Training data...
x_train=pd.read_csv('gdrive/My Drive/train-en.csv', usecols=["wind_speed48M"])
y_train=pd.read_csv('gdrive/My Drive/train-en.csv', usecols=["Output"])
```

As you see, I tested this code in Colab, where I now know it runs. If you're not using Colab, you can ignore the first two lines (but may need to deal with package compatibility issues).

Please make sure going through all of the code that you don't just copy-paste bits over but take the time this saves you to really understand what it all does.

I'm importing `numpy` here, not because I need it for the code above, but because we'll need it later.

Exercise — To learn more about the data, try to compute some basic statistics. You could e.g. compute correlations, means, variances, standard deviations etc. and inspect the value range in the Output variable. Ultimately, we are interested in predicting the output from the available capacity and (some of) the other features. So it will help us to understand what values we consider "normal" for the different columns/features.

We're not after "correctness" at the moment, we are trying to generate ideas and hypotheses.

Visualisation: Finally, plot the data and see what you can observe.

```
import matplotlib.pyplot as plt
plt.scatter(x_train["wind_speed48M"], y_train["Output"], color="cyan")
plt.show()
```

3. Heuristics for identifying faulty wind turbines.

Looking at your scatter plot above, try to find out:

- Are there any specific data points that look odd or unexpected?
- How could we - visually using the plot - create some rules that would allow us to identify faulty wind turbines in our wind farm?

Exercise — Write a Python program that - given a set of features, e.g. wind speed and output - decides whether or not the turbine is functioning correctly.

Once you've come up with your own solution, you may want to chat with others to see what ideas they used to address the task.

Just for comparison, the best that a student came up with last year is the below:

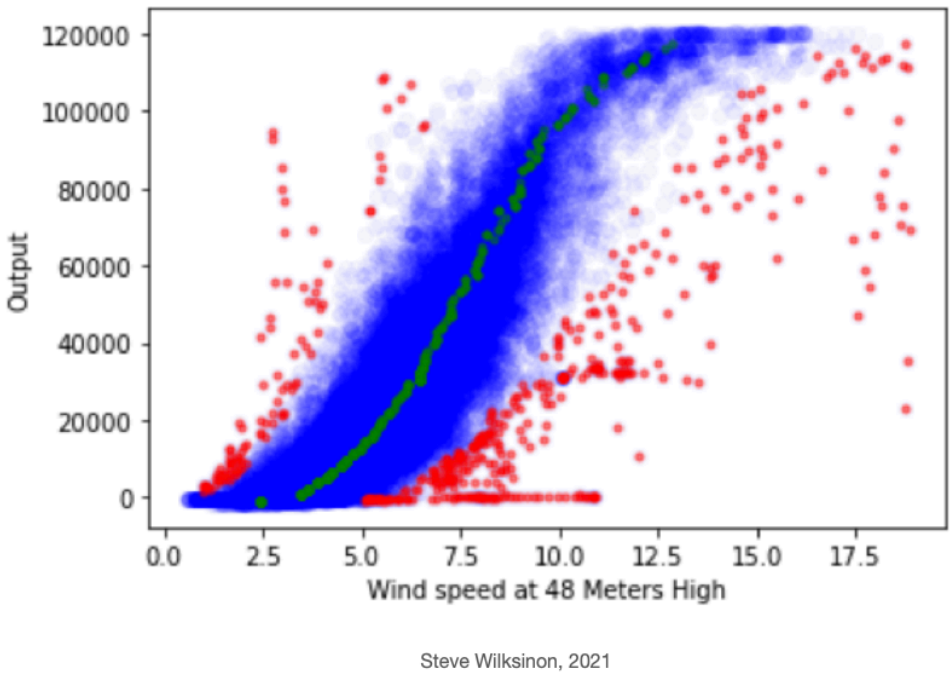

Figure 1: Heuristic solution for differentiating normal and faulty operation in a wind turbine.

4. Regression

Now that we've explored the data a bit, let's move on to a better solution — regression! We can use regression to predict the output of a continuous numeric variable, so let's apply it to our dataset and see how it can help us generate power output predictions for our wind farm given wind speed and direction. This could help us in the real world to address an operations and maintenance task for a wind turbine operator. If we can automatically detect deteriorated performance in the wind turbines, then we can issue early warnings, send out a maintenance / repair team, and avert the worst. This saves costs to the operator and makes renewable energy a more reliable alternative to fossil fuels.

To train a regressor, import the necessary libraries from Python's `sk_learn` library. See documentation here. You can also see a different example here.

```
from sklearn.linear_model import LinearRegression
reg = LinearRegression().fit(x_train, y_train)
scores = reg.score(x_train, y_train)

predictions = reg.predict(x_test)
print(predictions)
```

Then train the regressor using the training data we imported above, make some predictions and print them off to the console for some inspection. Note that I am using `x_test` here for my predictions. This is because I want to test my regressor on a new unseen dataset - to see how well it generalises. This is important — testing on the train set only tells me how well my model can recall/ remember the data points it has already seen. It doesn't tell me how well it generalises to new unseen tasks (which is the whole point of machine learning).

To get the `x_test` (input features) and `y_test` (labels), use eval-en.csv and read it in in the same as we did for the training data above.

We can plot the predictions against `x_test` to get a nice regression line.

```
plt.scatter(x_test["wind_speed48M"], y_test["Output"], color="cyan")
plt.plot(x_test, predictions, color="red", linewidth=3)
plt.show()
```

This line shows us where any labels are expected based on the inputs of x.

Exercise — Stop here to try and modify what x is. I have used "wind_speed48M" above, but that's just a guess - does any other feature give you a better regression line?

Evaluating our regression — We still want to generate a quantitative measure of how good our regressor is doing. We need to pick some evaluation metrics and apply them.

Read through the documentation here: https://scikit-learn.org/stable/modules/model_evaluation.html#regression-metrics and compare a few metrics for evaluating your regressor. The lecture slides should help you choose one if you're finding it tricky.

That's it - you've trained and evaluated your first regressor!

Predicting individual data points — while we evaluated our regressor quantitatively, we could try to inspect the performance on individual data points.

For example, I may want to know the expected output for a wind speed of 8. Can you modify your evaluation programme above to do this?

5. Generating labels

This section is now going beyond regression and offers an additional exercise if you've gone through the first part of the lab and still have time.

Let's return to our original motivation from above and think about how can we use the regression line above to help us create some labels faulty / non-faulty to inform our wind farm operators?

For each data point we can compare its predicted value against the actual value and work out the difference / the error between the two. This is also called residual and you compute it by subtracting the prediction value from the actual value, i.e. actual - prediction.

Error = Actual - Predicted.

All we need now is to decide a threshold value that marks the boundary between normal operation and error. I tweaked this a little bit and wrote a program that could colour in data points based on whether they were considered erroneous or not.

To get you started I did something very simple and wrote a for loop (very inefficient!) to do this per data point.

```
for i in range(0, len(y_test)):
  error = y_test["Output"][i] - predictions[i]


  if error > 8000 or error < -8000:
      discrete_error.append(1)
      plt.scatter(x_test["wind_speed48M"][i], y_test["Output"][i],
color="black")
  else:
    discrete_error.append(0)
    plt.scatter(x_test["wind_speed48M"][i], y_test["Output"][i], color="green")
```

As you can see, I have decided to also create a list `discrete_error` in which I want to save my discretised output labels. I'm using 0 for "normal operation" and 1 for "probably faulty".

The error can be positive or negative (either side of the regression line), so I'm trying to capture this in my code above.

Remember that the 8,000 is a hyperparameter that I made up for the sake of this. You can adjust this number and catch out different data points. Feel free to experiment with this to tweak what a good error threshold is.

The code also added differently coloured data points to our plot from above, you can visualise this plot again by calling `plot.show()`.

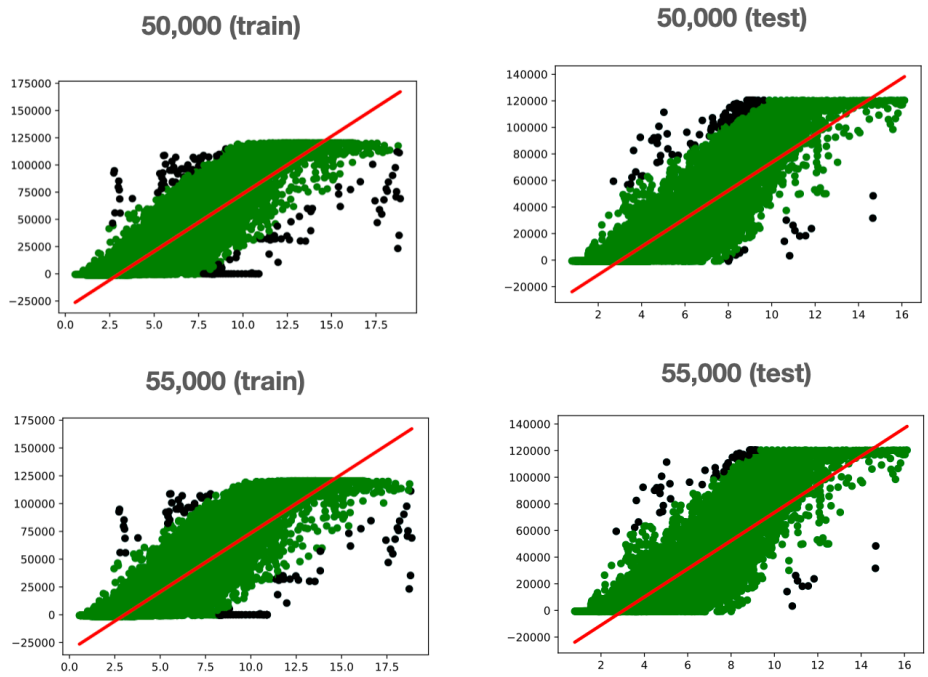

Figure 2: Scatter plots with tweaked error thresholds, where bold-face grey numbers mark the error threshold. Green data points are considered normal operations, black points are errors. This is for the train set (on the left) and for the test set (on the right).

I then want to save my new discretised outputs into a new DataFrame:

```
error = pd.DataFrame(discrete_error)
pd.DataFrame.to_csv(error, 'gdrive/My Drive/discrete_test.csv', sep=',', index=False)
```

At this point, we have generated a further layer of explanation of the data, we have created a model that can forecast the expected power output for a wind farm given e.g. the wind speed, and we have generated data to train a new model that can alert us to potential faults indicated through a shortfall of power output. That's a good outcome for our first lab!