Denise Löfflad (4017947),
Ingeborg Wenger (4029524)

Data Literacy Exercise 6

EXAMple

a)

$$
\begin{aligned}
\frac{\delta E(\omega)}{\delta\omega} &= \sum r_n 2(t_n - \omega^T x_n)(-x_n^T) \\
\Leftrightarrow \sum r_n t_n x_n^T &= \sum \omega^T x_n r_n x_n^T \\
\Leftrightarrow \sum r_n t_n x_n^T &= \omega^T \sum x_n r_n x_n^T \\
\Leftrightarrow \omega &= \sum r_n t_n x_n^T (\sum x_n r_n x_n^T)^{-1}
\end{aligned}
$$

b)

If some measurements are repeated, we would divide each r_n by the number of the repetitions of measurement n. This leads to the loss for the repeated measurement having the same size/influence as if the measurement was only once within the features.

c)

In case of different but known values σ^2, we would use $\frac{r_n}{\sigma^2}$. This would increase the influence of the more accurate measurements.

Theory Question

$$
\begin{aligned}
\log p(\omega|D,\alpha,\beta) &= C + \log p(D|\omega,\beta) + \log p(\omega|\alpha) \\
&= C + \log \prod_{i=0}^{N} p(t_i|\omega,\beta) + \log p(\omega|\alpha) \\
&= C + \sum_{i=0}^{N} \log p(t_i|\omega,\beta) + \log p(\omega|\alpha) \\
&= C - \frac{\beta}{2}\sum_{i=0}^{N}(t_i - \omega^T x_i)^2 - \frac{1}{2}\omega^T \alpha I \omega \\
&= C - \frac{1}{2}\Big[\beta\sum_{i=0}^{N} t_i^2 - 2\omega^T\beta\sum_{i=0}^{N} x_i t_i + \beta\sum_{i=0}^{N}\omega^T x_i \cdot \omega^T x_i + \omega^T \alpha I \omega\Big] \\
&= C - \frac{1}{2}\Big[-2\omega^T\beta\sum_{i=0}^{N} x_i t_i + \omega^T\beta\sum_{i=0}^{N} x_i x_i^T \omega + \omega^T \alpha I \omega\Big] \\
&= C - \frac{1}{2}\Big[-2\omega^T\beta\sum_{i=0}^{N} x_i t_i + \omega^T\big[\beta\sum_{i=0}^{N} x_i x_i^T + \alpha I\big]\omega\Big] \\
\Sigma^{-1} &= \beta\sum_{i=0}^{N} x_i x_i^T + \alpha I \\
\omega^T\beta\sum_{i=0}^{N} x_i t_i &= \omega^T \Sigma^{-1}\mu \\
\Leftrightarrow \mu &= \Sigma\beta\sum_{i=0}^{N} x_i t_i
\end{aligned}
$$

Data Literacy

University of Tübingen, Winter Term 2021/22

Exercise Sheet 6

© 2021 Prof. Dr. Jakob Macke & Marius Hobbhahn

This sheet is **due on Monday, December 6, 2021 at 10am sharp (i.e. before the start of the lecture).**

Regression

Much of Machine Learning that is currently done in the real world uses simple regression approaches such as linear or logistic regression. It is therefore important that we understand how these methods works, and what can go wrong when trying them out. This week we will focus on implementing linear regression and nonlinear regresssion on our own in simple settings, next week we will focus on logistic regression and regularization.

Part I: One-dimensional linear regression

In this part we will do linear (Gaussian) regression on a simple one-dimensional toy problem. We use simulated weight and height data (in reality, the relationship between the two is much more messy than in our simulated data).

Tasks:

1. Define a function that computes the linear regression weights given data X and labels y (don't use a pre-made package such as scikit-learn)
2. Import the weight-height.csv data
3. Apply linear regression with weight as X and height as y (Hint: if your function goes through the origin, you might still be missing something)
4. Plot the result (label the axis with correct units)
5. Finally, now use a linear regression package (e.g. scikit-learn) and fit it to your data, and check that you get the same result.

In [ ]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

def compute_regression_weights_v1(x, y):

    x_offset = np.ones(shape=(x.shape[0], 2))
    x_offset[:, 1:] = x.reshape(-1, 1)

    cov_x = np.dot(x_offset.T, x_offset)
    cov_yx = np.sum(y.reshape(-1, 1) * x_offset, axis=0)
    omega = np.dot(np.linalg.inv(cov_x), cov_yx.T)

    return omega

def compute_regression_weights_v2(x, y):

    x_offset = np.ones(shape=(x.shape[0], 2))
    x_offset[:, 1:] = x.reshape(-1, 1)

    omega = np.dot(np.linalg.pinv(x_offset), y.reshape(-1, 1)).squeeze()

    return omega
```

In [ ]:

```
### load data
data = pd.read_csv('weight-height.csv')
data.head()
```

Out[ ]:

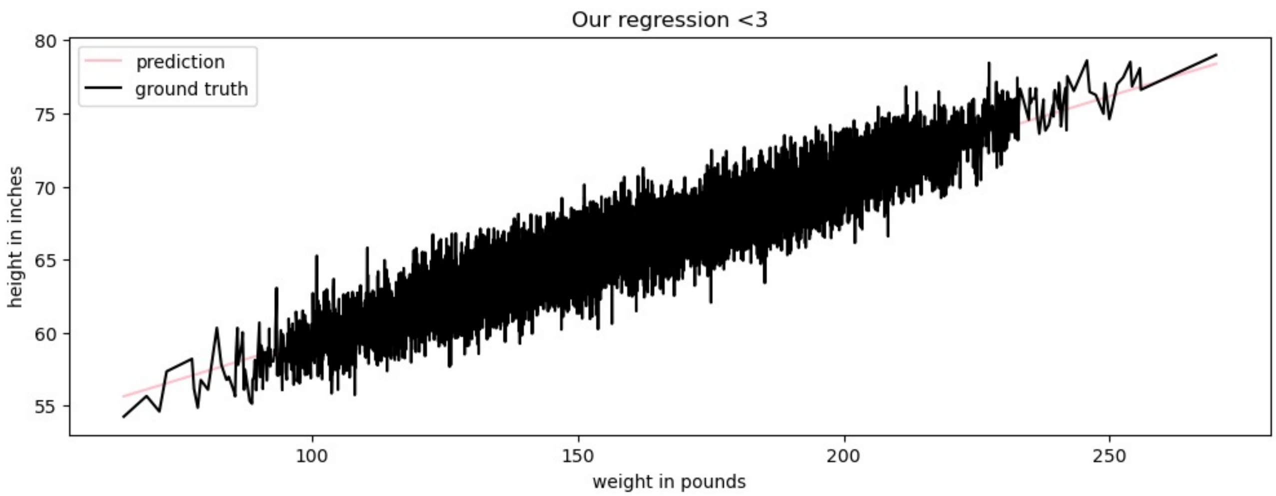
	Gender	Height	Weight
0	Male	73.847017	241.893563
1	Male	68.781904	162.310473
2	Male	74.110105	212.740856
3	Male	71.730978	220.042470
4	Male	69.881796	206.349801

In [ ]:

```
assert ((compute_regression_weights_v1(data['Weight'].values, data['Height'].values) - compute_regression_weights
```

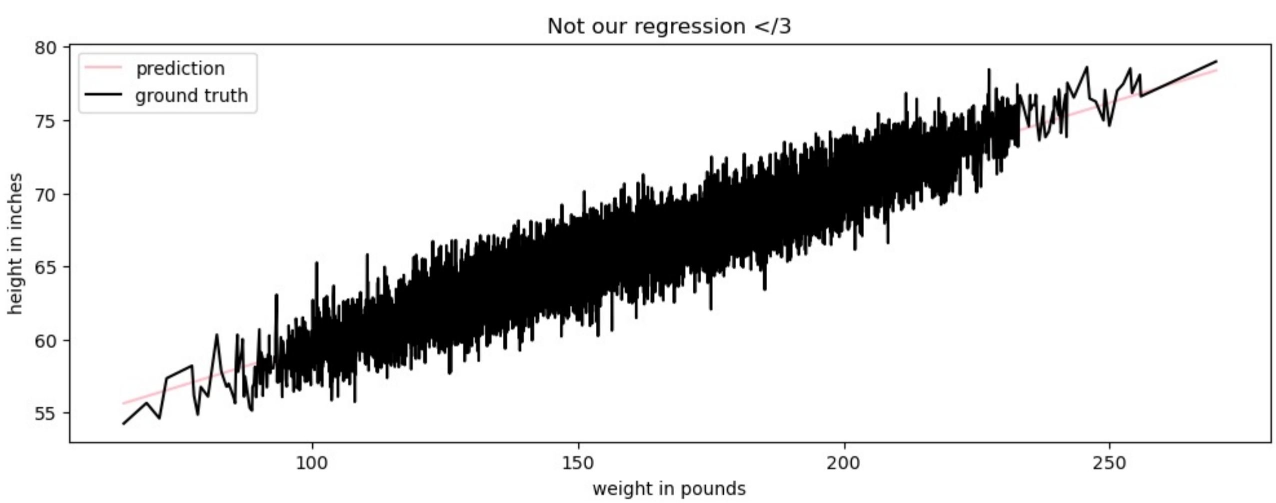
In [ ]:

```
### predict and plot
omega = compute_regression_weights_v1(data['Weight'].values, data['Height'].values)
sorted_data = data.sort_values(by='Weight')
prediction = omega[0] + sorted_data['Weight'].values * omega[1]

plt.figure(figsize=(12, 4))
plt.plot(sorted_data['Weight'], prediction, color='pink', label='prediction')
plt.plot(sorted_data['Weight'], sorted_data['Height'].values, color='black', label='ground truth')
plt.xlabel('weight in pounds')
plt.ylabel('height in inches')
plt.legend()
plt.title('Our regression <3')
plt.show();
```



In [ ]:

```
### compare to sklearn
from sklearn.linear_model import LinearRegression

reg = LinearRegression().fit(sorted_data['Weight'].values.reshape(-1, 1), sorted_data['Height'].values.reshape(-1
sklearn_predictions = reg.predict(sorted_data['Weight'].values.reshape(-1, 1))

plt.figure(figsize=(12, 4))
plt.plot(sorted_data['Weight'], sklearn_predictions, color='pink', label='prediction')
plt.plot(sorted_data['Weight'], sorted_data['Height'].values, color='black', label='ground truth')
plt.xlabel('weight in pounds')
plt.ylabel('height in inches')
plt.legend()
plt.title('Not our regression </3')
plt.show();
```

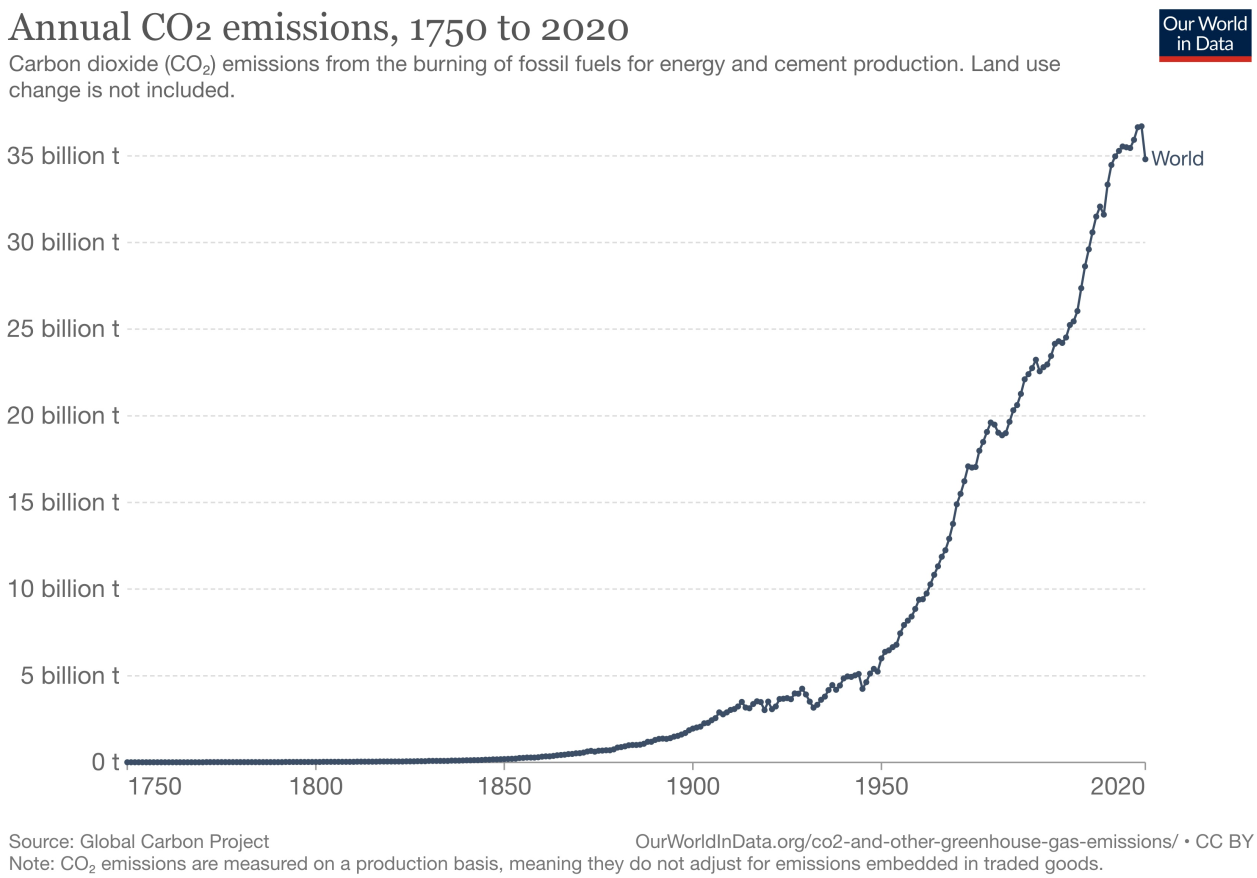


Part II: One-dimensional nonlinear regression

We are now looking at a trend that is clearly not linear. However, we can use a transformation of the data to look at it in the log space and do a linear regression there. See https://ourworldindata.org/co2-emissions for the data.

In [ ]:
```
from IPython.display import Image
Image(filename='co2_world.png')
```

Out[ ]:



Tasks:

1. Import the co2.csv data
2. Apply the log transformation
3. Do the same as in Part I
4. Transform the predicted function back and plot the results

In [ ]:
```
### load dataset
co2_data = pd.read_csv('co2_world.csv', index_col=0)
co2_data.head()
```

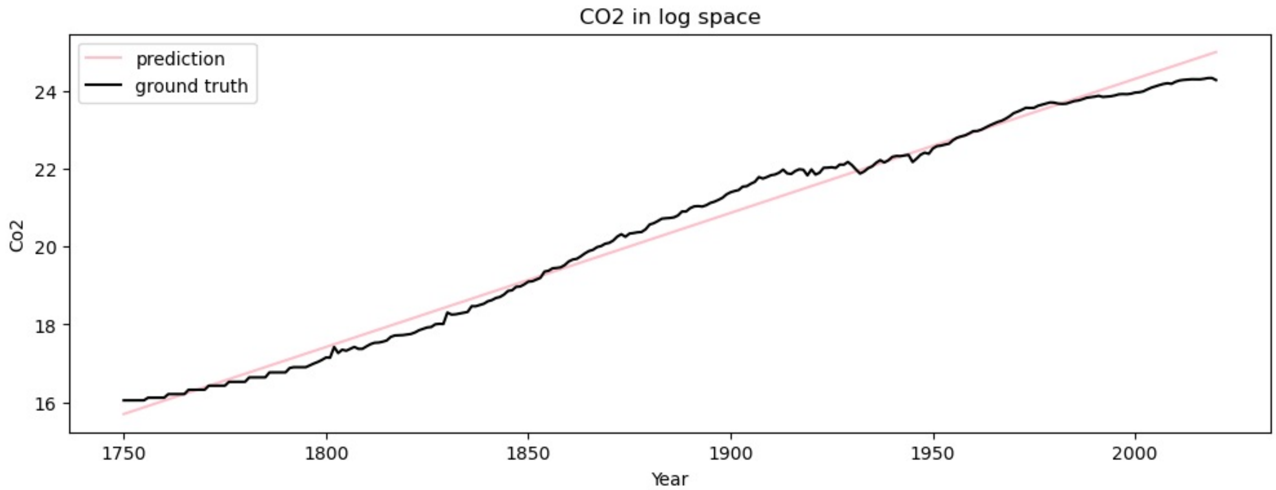
Out[ ]:

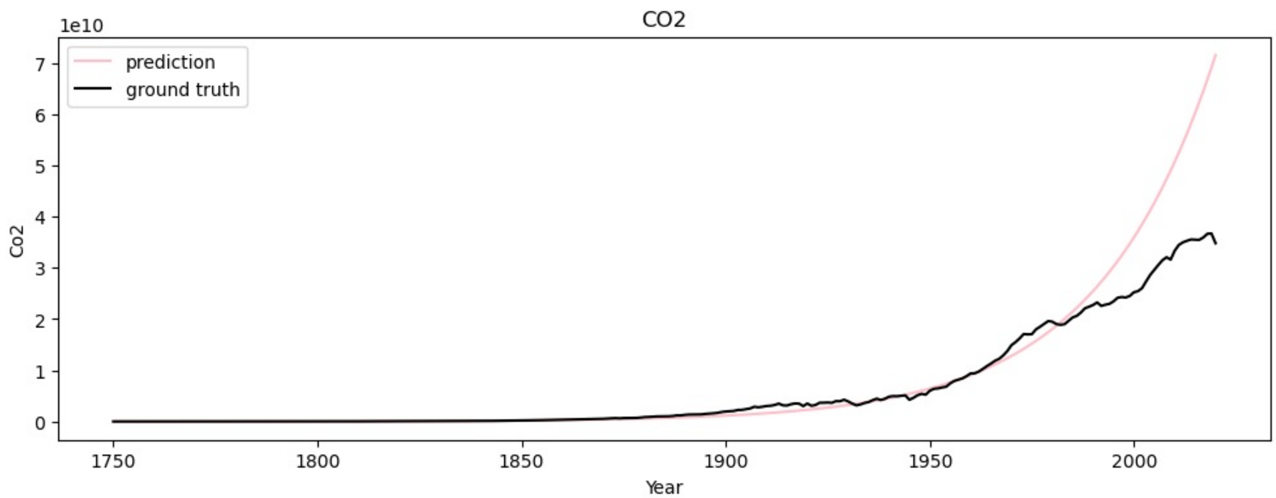
	Year	Annual CO2 emissions
23418	1750	9350528
23419	1751	9350528
23420	1752	9354192
23421	1753	9354192
23422	1754	9357856

In [ ]:
```
### make linear regression in log space
years = co2_data['Year'].values
emissions = co2_data['Annual CO2 emissions'].values
log_emissions = np.log(emissions)

omega = compute_regression_weights_v1(years, log_emissions)
```

In [ ]:
```
### predict and plot
log_prediction = omega[0] + years * omega[1]
```

```python
plt.figure(figsize=(12, 4))
plt.plot(years, log_prediction, color='pink', label='prediction')
plt.plot(years, log_emissions, color='black', label='ground truth')
plt.xlabel('Year')
plt.ylabel('Co2')
plt.legend()
plt.title('CO2 in log space')
plt.show();

plt.figure(figsize=(12, 4))
plt.plot(years, np.exp(log_prediction), color='pink', label='prediction')
plt.plot(years, emissions, color='black', label='ground truth')
plt.xlabel('Year')
plt.ylabel('Co2')
plt.legend()
plt.title('CO2')
plt.show();
```





Part III: a different log regression

The simple logarithmic transform that we used above assumes a linear function in log-space. However, there are other ways in which we can modify a linear function that uses an implicit logarithmic function.

In this exercise we use the model $f(x) = a * \exp(b * x) + c$ to fit our data.

You are given the code for the new model and a loss function.

Tasks:

1. Use scipy.optimize to fit the parameters a, b, and c to the data. We found that the "Nelder-Mead" method of optimization works well in this case.
2. Try different initializations for the parameters. We found that the resulting functions look very different depending on the initialization.
3. Plot your fit and compare it with the original data and the fit from the previous exercise.

In [ ]:

```python
def exp_regression(x, a, b, c):
```

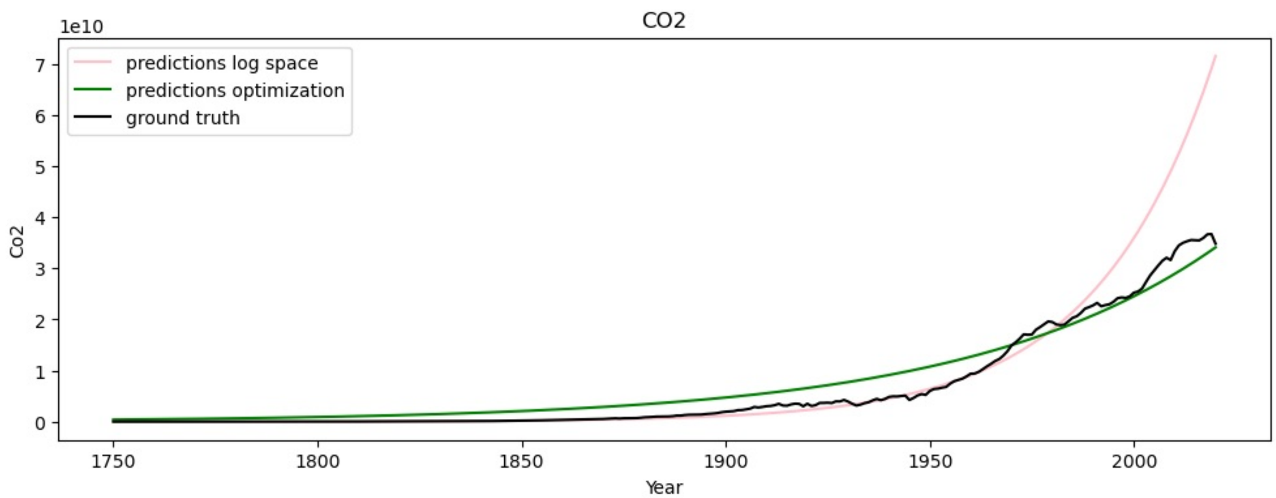
```
    return(a * np.exp(b * x) + c)

def l2_norm(y, y_hat):
    return(np.sqrt(np.sum((y - y_hat)**2)))
```

In [ ]:
```
import scipy.optimize
```

In [ ]:
```
def minimize_l2(x, data, obs):
    prediction = exp_regression(data, x[0], x[1], x[2])
    return l2_norm(obs, prediction)
```

In [ ]:
```
### use scipy.optimize to get parameter values for a,b and c
initial_state = [10, 1e-2, 5]
opt = scipy.optimize.minimize(minimize_l2, x0=initial_state, args=(years, emissions), method='Nelder-Mead')
a, b, c = opt.x
```

In [ ]:
```
### predict and plot
opti_predictions = exp_regression(years, a, b, c)

plt.figure(figsize=(12, 4))
plt.plot(years, np.exp(log_prediction), color='pink', label='predictions log space')
plt.plot(years, opti_predictions, color='green', label='predictions optimization')
plt.plot(years, emissions, color='black', label='ground truth')
plt.xlabel('Year')
plt.ylabel('Co2')
plt.legend()
plt.title('CO2')
plt.show();
```



High-level questions:

We have now fitted two different functions with implicit logarithmic transformations. Please answer the following questions:

1. What are the different assumptions these two models make, i.e. in which ways and why do the induced functions differ?
2. Which of them make more sense in your opinion?
3. Do you think either of the models yields a good prediction for the next 10, 20 or 50 years of CO2 development? Why or why not?

1. The first model assumes a linear function in log space. However, when transforming the function back, small deviations from log space linearity possibly lead to large deviations when taking the exponential. The second model already includes some non-linearity and therefore adapts more easily to the data.
2. Non-linear functions are generally more powerful and can fit better to data. Thus, since we see that the data increases exponentially, it seems more sensible to directly fit it with an exponential function.
3. We assume that the next 10 years might be fitted reasonably well with the second model. However, since there is no causality betweeen years and CO2, but the CO2 depends on socio-economic factors and politics, the model would have to be far more complicated to make correct predictions about the future.

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js