Principles of Machine Learning: Exercise 2

Alina Pollehn (3197257), Julian Litz (3362592), Manuel Hinz (3334548)
Felix Göhde (3336445), Felix Lehmann (3177181), Caspar Wiswesser (3221493)
Adrian Köring (3347785), Greta Günther (3326765), Linus Mallwitz (3327653)
Niklas Mueller-Goldingen (3363219), Jennifer Kroppen (2783393)

19.11.2023

Task 2.1.1-2 :: Loading

Instead of removing the outliers, its easier to keep the inliers:

```
inliers = w > 0
X = np.stack([h[inliers], w[inliers]])
# X.shape = [2, 37] = [F, N]
```

Maximum Likelihood Estimation of a Gaussian via empirical mean and covariance:

```
μ = np.mean(X, axis=1, keepdims=True)
# μ = [[175.729], [73.865]] with μ.shape=[2, 1]
S = np.cov(X - μ, ddof=1) # ddof = degree of freedom = scales with 1/(n-1)
# S == (1 / (N-1)) * (X - μ) @ (X - μ).T
# S == [[ 75.925,  64.546 ],
#       [ 64.546, 186.953]]
```

Task 2.1.3 :: Predictions

Conditional Probability of a Gaussian:

```
for h in np.arange(140, 220, step=10):
    μw = μ[1] + S[1, 0] * S[0, 0]**(-1) * (h - μ[0])
    Sw = S[1, 1] - S[1, 0] * S[0, 0]**(-1) * S[0, 1]
```

	Height	Weight	Covariance
0	140	43.490072	132.081358
1	150	51.991338	132.081358
2	160	60.492604	132.081358
3	170	68.993869	132.081358
4	180	77.495135	132.081358
5	190	85.996401	132.081358
6	200	94.497666	132.081358
7	210	102.998932	132.081358

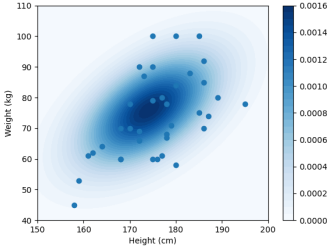


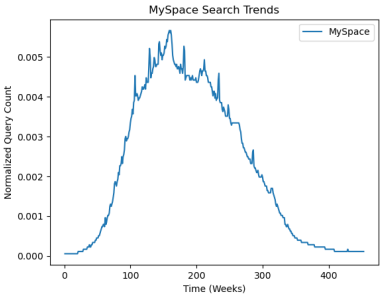
Figure: Estimated PDF and given data

Task 2.1.3 :: Pondering

+ Results seem plausible – can mostly judge around the mean
- Fixed variance doesn't match expectations
  - maybe taller people → more variance?
  - there are more average sized people → maybe more variance?
- Cube-Square-Law / BMI suggests a non-linear / quadratic relationship?
- Plausibility? Test-Set and more 'human-readable' metrics!

Task 2.2.1 :: Loading

```
import pandas as pd
# Loading data as pandas Dataframe
df = pd.read_csv("myspace.csv",
                 header=None,
                 names=["Week", "Accesses"])
# Filter leading zeros only with cumulative sum
df = df[np.cumsum(df.Accesses) > 0]
h = np.array(df.Accesses, dtype=float)
t = np.arange(1, len(h)+1, dtype=float)
```



Task 2.2.2 :: A more efficient way to calculate partial derivatives (1)

Goal: Fitting a Weibull distribution $f(t|\alpha,\beta) = \frac{\alpha}{\beta}\left(\frac{t}{\beta}^{\alpha-1} e^{-\left(\frac{t}{\beta}^{\alpha}\right)}\right)$ While our data is given as a histogram, our derivatives are expressed in terms of a set of numbers. We therefore calculate $N = \sum_{i=1}^{n} h_i$ and rewrite our sums in terms of h_i:

$$\frac{\partial L}{\partial \beta} = \frac{\alpha}{\beta}\left(\sum_{i=1}^{N}\left(\frac{d_i}{\beta}\right)^{\alpha} - N\right) = \frac{\alpha}{\beta}\left(\sum_{i=1}^{n}\sum_{j=1}^{h_i} \underbrace{\left(\frac{t_{i,j}}{\beta}\right)^{\alpha}}_{\text{constant for fixed } i} - N\right) = \frac{\alpha}{\beta}\left(\sum_{i=1}^{n} h_i \left(\frac{t_i}{\beta}\right)^{\alpha} - N\right)$$

Task 2.2.2 :: A more efficient way to calculate partial derivatives (2)

We can rewrite all of our derivatives in this way to get:

$$\frac{\partial L}{\partial \alpha} = \frac{N}{\alpha} - N \log \beta + \sum_{i=1}^{n} h_i \log t_i - \sum_{i=1}^{n} h_i \left(\frac{t_i}{\beta}\right)^{\alpha} \log \frac{t_i}{\beta}$$

$$\frac{\partial^2 L}{\partial \alpha^2} = -\frac{N}{\alpha^2} - \sum_{i=1}^{n} h_i \left(\frac{t_i}{\beta}\right)^{\alpha} \left(\log \frac{t_i}{\beta}\right)^2$$

$$\frac{\partial^2 L}{\partial \beta^2} = \frac{\alpha}{\beta^2} \left(N - (\alpha + 1) \sum_{i=1}^{n} h_i \left(\frac{t_i}{\beta}\right)^{\alpha}\right)$$

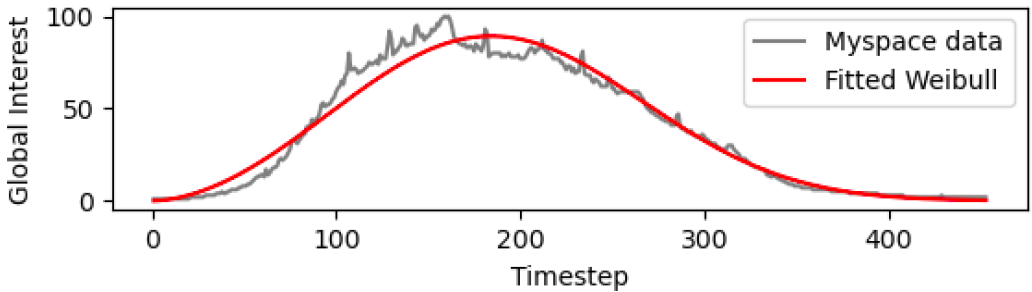
$$\frac{\partial^2 L}{\partial \alpha \partial \beta} = \frac{1}{\beta} \sum_{i=1}^{n} h_i \left(\frac{t_i}{\beta}\right)^{\alpha} + \frac{\alpha}{\beta} \sum_{i=1}^{n} h_i \left(\frac{t_i}{\beta}\right)^{\alpha} \log \frac{t_i}{\beta} - \frac{N}{\beta}$$

Task 2.2.2 :: Results

Our MLE implementation yields

$$\alpha \approx 2.809$$

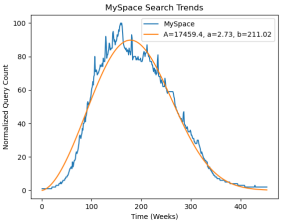
$$\beta \approx 215.429$$



Task 2.3 :: Scipy CurveFit

```
def weibull(t, A, alpha, beta):
    ab, tb = alpha / beta, t / beta
    return A * ab * tb**(alpha - 1) * np.exp(-tb**alpha)
(A, alpha, beta), _ = curve_fit(weibull, t, h, p0=[1000, 1.0, 1.0])
```

- mostly works just like that
- could have also scaled the data (see 2.5) instead of adding the amplitude parameter
- $A = 17459.4$, $\alpha = 2.731$, $\beta = 211.024$



Task 2.4 :: Theory

1. Idea: Fitting a multinomial distribution to our histogram:

$$h(x_1, \ldots, x_n) = N! \prod_i \frac{p_i^{x_i}}{x_i!}$$

$$p_i(\theta_1, \theta_2) = \begin{cases} F(t_i) - F(t_{i-1} & i > 0 \\ F(t_i) & i = 0 \end{cases})$$

2. Use a iterative weighted least squares scheme:

$$\sum_i w_i (x_i - N p_i(\theta_1, \theta_2))^2$$

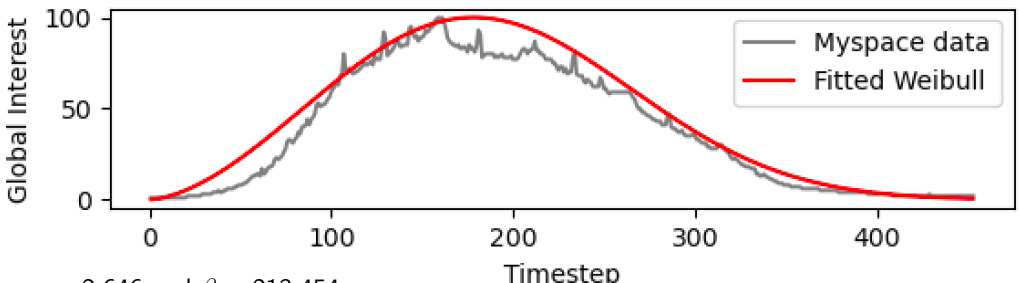
3. Scipy's least squares only wants a function which computes the residual of a single data point (h_i, t_i):

$$\sqrt{w_i}(x_i - N p_i(\theta_1, \theta_2))$$

Task 2.4 :: Results

```
theta_hat=least_squares(lambda theta:get_residuals(N,hs,ts,*theta),\
                        [1,100],bounds=([0,0],np.inf),ftol=1e-14).x
```

yields



- $\alpha = 2.646$ and $\beta = 213.454$
- Observation: Choosing a arbitrary ftol $\in [10^{-16}, 10^{-10}]$ yields the result above, while choosing a higher tolerance (10^{-4}) yields $2.646, 213.454$. In this case it does not make a significant difference (both in speed and accuracy) which (reasonable) tolerance we use.

Task 2.5 :: Scipy Minimize

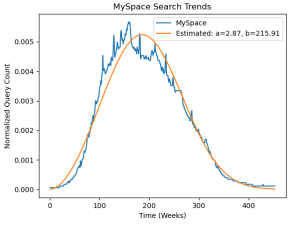
```
def KL(f, q): return np.sum(f * np.log(f / q))

def objective(x):
    return KL(weibull(t, alpha=x[0], beta=x[1]), q)
result = minimize(objective, x0=[1.0, 100.], bounds=[(0, 10), (0, 500)])
```

- $\alpha = 2.87$, $\beta = 215.91$
- Bounds are important, but also not: $[-\infty, \infty]$ works, too
- Feels odd anyway, because a gradient method optimizing $1 \rightarrow 2.8$ shouldn't come near 0 anyway

$\Rightarrow$ Adding bounds just change the algorithm underneath

- Without bounds 'BFGS' is choosen and doesnt quite work
- 'L-BFGS-B' or 'SLSQP' without bounds works equally well



- In Task 2.2, 2.3, 2.4 and 2.5 we fitted a Weibull distribution to the same dataset.

	2.2	2.3	2.4	2.5
α	2.809	2.731	2.646	2.87
β	215.429	211.024	213.454	215.91

- Each task had a different way to formalize "fitting a Weibull distribution", therefore slightly different result are not surprising
- No single fit is the best, each minimizes a different loss.
- Depending on the use case, it might be useful to add another random component (such as a gaussian with mean zero) to better explain small deviations from our (scaled) Weibull distribution