# Machine Learning

**Summer Semester 2018, Homework 1**

**Prof. Dr. J. Peters, D. Tanneberg, B. Belousov**

TECHNISCHE UNIVERSITÄT DARMSTADT

**Total points: 85 + 15 bonus**
Due date: Wednesday, 02 May 2018 (before the lecture)

**Name, Surname, ID Number** **Fabian Otto, 2792549** **Felix Mannl, 2639576**

## Problem 1.1 Linear Algebra Refresher [20 Points]

a) **Matrix Properties [5 Points]**

A colleague of yours suggests matrix addition and multiplication are similar to scalars, thus commutative, distributive and associative properties can be applied. Is the statement correct? Prove it analytically or give counterexamples (for both operations) considering three matrices $A, B, C$ of size $n \times n$.

***distributive property:*** *For all* $i = 1, ..., n,\ j = 1, ..., n$

$$\left(A(B+C)\right)_{ij} = \sum_{k=1}^{n} A_{ik} \left(B+C\right)_{kj} = \sum_{k=1}^{n} A_{ik} \left(B_{kj} + C_{kj}\right) = \sum_{k=1}^{n} A_{ik} B_{kj} + \sum_{k=1}^{n} A_{ik} C_{kj} = (AB)_{ij} + (AC)_{ij}\,.$$

*Thus* $A(B+C) = AB + AC$.

***mulitiplication associative property:*** *For all* $i = 1, ..., n,\ j = 1, ..., n$

$$\left(A(BC)\right)_{ij} = \sum_{k=1}^{n} A_{ik} \left(BC\right)_{kj} = \sum_{k=1}^{n} \sum_{l=1}^{n} A_{ik} \left(B_{kl} C_{lj}\right)$$

$$\sum_{k=1}^{n} \sum_{l=1}^{n} A_{ik} \left(B_{kl} C_{lj}\right) = \sum_{l=1}^{n} \sum_{k=1}^{n} \left(A_{ik} B_{kl}\right) C_{lj} = \sum_{l=1}^{n} (AB)_{il}\, C_{lj} = \left((AB)\, C\right)_{ij}\,.$$

*Thus* $A(BC) = (AB)\,C$.

***addition associative property:*** *For all* $i = 1, ..., n,\ j = 1, ..., n$

$$\left(A + (B+C)\right)_{ij} = \sum_{k=1}^{n} A_{ik} + (B+C)_{kj} = \sum_{k=1}^{n} \sum_{l=1}^{n} A_{ik} + \left(B_{kl} + C_{lj}\right) = \sum_{l=1}^{n} \sum_{k=1}^{n} \left(A_{ik} + B_{kl}\right) + C_{lj} = \sum_{l=1}^{n} (A+B)_{il} + C_{lj} = \left((A+B) + C\right)_{ij}\,.$$

*Thus* $A + (B+C) = (A+B) + C$.

***multiplication commutativ property:*** *Let*

$$A = \begin{pmatrix} 1 & 2 \\ 1 & 2 \end{pmatrix},\ B = \begin{pmatrix} 1 & 1 \\ 2 & 2 \end{pmatrix}.$$

*Then* $AB = \begin{pmatrix} 5 & 5 \\ 5 & 5 \end{pmatrix}$ *and* $BA = \begin{pmatrix} 2 & 4 \\ 4 & 8 \end{pmatrix}$. *Therefore in general* $AB \neq BA$.

***addition commutativ property:*** *For all* $i = 1, ..., n,\ j = 1, ..., n$

$$(A+B)_{ij} = A_{ij} + B_{ij} = B_{ij} + A_{ij} = (B+A)_{ij}\,.$$

*Thus $A+B=B+A$.*

b) **Matrix Inversion [6 Points]**

Given the following matrix

$$A=\begin{pmatrix}1&2&3\\1&2&4\\1&4&5\end{pmatrix}$$

analytically compute its inverse $A^{-1}$ and illustrate the steps.
If we change the matrix in

$$A=\begin{pmatrix}1&2&3\\1&2&4\\1&2&5\end{pmatrix}$$

is it still invertible? Why?

*Calculate the inverse $A^{-1}$ with the Gaussian Elimination algorithm*

$$A_0=\begin{pmatrix}1&2&3\\1&2&4\\1&4&5\end{pmatrix}\quad B_0=\begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix}$$
$$A_1=\begin{pmatrix}1&2&3\\0&0&1\\0&2&2\end{pmatrix}\quad B_1=\begin{pmatrix}1&0&0\\-1&1&0\\-1&0&1\end{pmatrix}$$
$$A_2=\begin{pmatrix}1&2&3\\0&1&1\\0&0&1\end{pmatrix}\quad B_2=\begin{pmatrix}1&0&0\\-1/2&0&1/2\\-1&1&0\end{pmatrix}$$
$$A_3=\begin{pmatrix}1&2&0\\0&1&0\\0&0&1\end{pmatrix}\quad B_3=\begin{pmatrix}4&-3&0\\1/2&-1&1/2\\-1&1&0\end{pmatrix}$$
$$A_4=\begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix}\quad B_4=\begin{pmatrix}3&-1&-1\\1/2&-1&1/2\\-1&1&0\end{pmatrix}.$$

*Now we have $A^{-1}=B_4$.*

*The first two columns of the new matrix A are linearly dependent. So the determinate is 0 and thus A is not invertible.*

c) **Matrix Pseudoinverse [3 Points]**

Write the definition of the right and left Moore-Penrose pseudoinverse of a generic matrix $A \in \mathbb{R}^{n \times m}$.
Given $A \in \mathbb{R}^{2 \times 3}$, which one does exist? Write down the equation for computing it, specifying the dimensionality of the matrices in the intermediate steps.

*$A^+ \in \mathbb{R}^{n \times m}$ is a Moore-Penrose pseudoinverse of A if it statisfys the following four conditions:*
*1.* $AA^+A = A$
*2.* $A^+AA^+ = A^+$
*3.* $\left(AA^+\right)^T = AA^+$
*4.* $\left(A^+A\right)^T = A^+A$
*If additionally to the four conditions A has linearly independent columns the left Moore-Penrose pseudoinverse $A^+$ can be calculated by $A^+ = \left(A^TA\right)^{-1}A^T$. In this case $A^+A = I$.*
*Similary if A has linearly independent rows the right Moore-Penrose pseudoinverse $A^+$ can be calculated by $A^+ = A^T\left(AA^T\right)^{-1}$. In this case $AA^+ = I$.*

*If $A \in \mathbb{R}^{2 \times 3}$ the maximum rank is 2, so only the rows of A can be linearly independent. Thus only the right pseudoinverse can exist.*

*The right pseudoinverse is calculated by $A^+ = A^T\left(AA^T\right)^{-1}$ with $A^T \in \mathbb{R}^{3 \times 2}$, $AA^T$ and $\left(AA^T\right)^{-1} \in \mathbb{R}^{2 \times 2}$, $A^T\left(AA^T\right)^{-1} \in \mathbb{R}^{3 \times 2}$ and finally $A^+ \in \mathbb{R}^{3 \times 2}$*

d) **Eigenvectors & Eigenvalues [6 Points]**

What are eigenvectors and eigenvalues of a matrix *A*? Briefly explain why they are important in Machine Learning.

*Let $A \in \mathbb{R}^{n \times n}$. A vector $v \in \mathbb{R}^n$ is called an eigenvector of A with eigenvalue $\lambda \in \mathbb{R}$ if $Av = \lambda v$. As we can see eigenvectors only change in length when multiplied with a matrix and eigenvalues are the scaling factor for the previous mentioned length change.*

*Eigenvectors/-values can be used in order to identify features, which are important or have a high variance and represent a large amount of data points. One could say feature that have a higher variance provide more information about the data. As a consequence dimensions/features can be ranked and or be removed. E.g. in PCA Eigenvectors and -values play an important rule in order to determine the principal components, which are a linear combination of dimensions.*

## Problem 1.2 Statistics Refresher [25 Points]

a) **Expectation and Variance [8 Points]**

Let $\Omega$ be a finite set and $P : \Omega \to \mathbb{R}$ a probability measure that (by definition) satisfies $P(\omega) \geq 0$ for all $\omega \in \Omega$ and $\sum_{\omega\in\Omega} P(\omega) = 1$. Let $f : \Omega \to \mathbb{R}$ be an arbitrary function on $\Omega$.

**1)** Write the definition of expectation and variance of $f$ and discuss if they are linear operators.

**2)** You are given a set of three dices $\{A, B, C\}$. The following table describes the outcome of six rollouts for these dices, where each column shows the outcome of the respective dice. (Note: assume the dices are standard six-sided dices with values between 1-6)

| | | | | | | |
|---|---|---|---|---|---|---|
| $A$ | 4 | 4 | 2 | 4 | 1 | 1 |
| $B$ | 3 | 6 | 3 | 3 | 4 | 3 |
| $C$ | 5 | 5 | 2 | 1 | 1 | 1 |

Estimate the expectation and the variance for each dice using unbiased estimators. (Show your computations).

**3)** According to the data, which of them is the "most rigged"? Why?

***1)*** *We can define the expectation by* $\mathbb{E}[f] = \sum_{\omega\in\Omega} P(\omega) f(\omega)$.
*Using this definition for the variance we get:* $\text{var}[f] = \mathbb{E}[f^2] - \mathbb{E}[f]^2$.
*Let* $X, Y$ *be two random variables and* $Z = X + Y$. *The expectation is a linear function, because for any two points on the function the following holds:* $\mathbb{E}[Z] = \sum_{\omega\in\Omega} Z(\omega) P(\omega) = \sum_{\omega\in\Omega} \big(X(\omega) + Y(\omega)\big) P(\omega) = \mathbb{E}[X] + \mathbb{E}[Y]$.
*However the variance of the sum of two random variables is*

$$\text{var}[Z] = \mathbb{E}[(Z)^2] - \mathbb{E}[Z]^2 = \text{var}[X] + \text{var}[Y] + 2\mathbb{E}[XY] - 2\mathbb{E}[X]\mathbb{E}[Y].$$

*In general* $\mathbb{E}[XY] \neq \mathbb{E}[X]\mathbb{E}[Y]$ *so the variance is not a linear operator.*

***2)*** *The unbiased estimator for the expected value is given by*

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i.$$

*Calculating the estimated expected value for dices A, B and C:*

$$\bar{x}_A = \frac{1}{6}(4+4+2+4+1+1) = \frac{16}{6} = 2.\bar{6},$$
$$\bar{x}_B = \frac{1}{6}(3+6+3+3+4+3) = \frac{22}{6} = 3.\bar{6},$$
$$\bar{x}_C = \frac{1}{6}(5+5+2+1+1+1) = \frac{15}{6} = 2.5.$$

*The unbiased estimator for the variance is given by*

$$\bar{\sigma} = \frac{1}{n-1}\sum_{i=1}^{n}\big(x_i - \bar{x}\big)^2.$$

*Calculating the estimated variance for dices A, B and C:*

$$\sigma_A = \frac{1}{5}\left(\left(4-\frac{16}{6}\right)^2+\left(4-\frac{16}{6}\right)^2+\left(2-\frac{16}{6}\right)^2+\left(4-\frac{16}{6}\right)^2+\left(1-\frac{16}{6}\right)^2+\left(1-\frac{16}{6}\right)^2\right) = 2.2\bar{6},$$
$$\sigma_B = \frac{1}{5}\left(\left(3-\frac{22}{6}\right)^2+\left(6-\frac{22}{6}\right)^2+\left(3-\frac{22}{6}\right)^2+\left(3-\frac{22}{6}\right)^2+\left(4-\frac{22}{6}\right)^2+\left(3-\frac{22}{6}\right)^2\right) = 1.4\bar{6},$$
$$\sigma_C = \frac{1}{5}\left(\left(5-\frac{15}{6}\right)^2+\left(5-\frac{15}{6}\right)^2+\left(2-\frac{15}{6}\right)^2+\left(1-\frac{15}{6}\right)^2+\left(1-\frac{15}{6}\right)^2+\left(1-\frac{16}{6}\right)^2\right) = 3.9.$$

***3)*** *We use the Kullback-Leibler divergence to find the most rigged die with $P_i$ the probability distribution of tested die $i$ and $Q$ the probability distribution of a fair die $\sum_{x\in X} P(x)\log\frac{P(x)}{Q(x)}$. We choose to use the natural logarithm, this is a arbitrary choice since the logarithms are constant multiples of each other.*

$$D\left(P_A \,||\, Q\right) = \frac{2}{6}\ln\left(\frac{\frac{2}{6}}{\frac{1}{6}}\right)+\frac{1}{6}\ln\left(\frac{\frac{1}{6}}{\frac{1}{6}}\right)+\frac{3}{6}\ln\left(\frac{\frac{3}{6}}{\frac{1}{6}}\right) = 0.7804,$$
$$D\left(P_B \,||\, Q\right) = \frac{1}{6}\ln\left(\frac{\frac{1}{6}}{\frac{1}{6}}\right)+\frac{4}{6}\ln\left(\frac{\frac{4}{6}}{\frac{1}{6}}\right)+\frac{1}{6}\ln\left(\frac{\frac{1}{6}}{\frac{1}{6}}\right) = 0.9241,$$
$$D\left(P_C \,||\, Q\right) = \frac{3}{6}\ln\left(\frac{\frac{3}{6}}{\frac{1}{6}}\right)+\frac{1}{6}\ln\left(\frac{\frac{1}{6}}{\frac{1}{6}}\right)+\frac{2}{6}\ln\left(\frac{\frac{2}{6}}{\frac{1}{6}}\right) = 0.7804.$$

*As we can see die B has the biggest Kullback-Leibler divergence, so it is the 'most rigged' die*

b) **It is a Cold World [7 Points]**

Consider the following three statements:
a) A person with a cold has backpain 25% of the time.
b) 4% of the world population has a cold.
c) 15% of those who do not have a cold, still have backpain.

**1)** Identify random variables from the statements above and define a unique symbol for each of them.
**2)** Define the domain of each random variable.
**3)** Represent the three statements above with your random variables.
**4)** If you suffer from backpain, what are the chances that you suffer from a cold? (Show all the intermediate steps.)

***1/2)*** *Let* $b \in \{0,1\}$ *indicate if a person has backpain* $b = 1$ *or not* $b = 0$ *and* $c \in \{0,1\}$ *indicacte if a person has a cold* $c = 1$ *or not* $c = 0$.

***3)***
*a)* $P\left(b = 1 \mid c = 1\right) = 0.25$
*b)* $P\left(c = 1\right) = 0.04$
*c)* $P\left(b = 1 \mid c = 0\right) = 0.15$

***4)***
*With the rule of Bayes we get*

$$P\left(c = 1 \mid b = 1\right) = \frac{P\left(b = 1 \mid c = 1\right) P\left(c = 1\right)}{P\left(b = 1\right)}. \tag{4}$$

*Then we use the law of total probability and get*

$$\frac{P\left(b = 1 \mid c = 1\right) P\left(c = 1\right)}{P\left(b = 1 \mid c = 1\right) P\left(c = 1\right) + P\left(b = 1 \mid c = 0\right) P\left(c = 0\right)} \tag{5}$$

*Finally we insert the values given by a), b) and c)*

$$\frac{0.25 * 0.04}{0.25 * 0.04 + 0.15 * (1 - 0.04)} = 0.649. \tag{6}$$

c) **Journey to THX1138 [10 Points]**

After the success of the Rosetta mission, ESA decided to send a spaceship to rendezvous with the comet THX1138. This spacecraft consists of four independent subsystems $A, B, C, D$. Each subsystem has a probability of failing during the journey equal to 1/3.
1) What is the probability of the spacecraft $S$ to be in working condition (i.e., all subsystems are operational at the same time) at the rendezvous?
2) Given that the spacecraft $S$ is not operating properly, compute analytically the probability that **only** subsystem $A$ has failed.
3) Instead of computing the probability analytically, do a simple simulation experiment and compare the result to the previous solution. Include a snippet of your code.
4) An improved spacecraft version has been designed. The new spacecraft fails if the critical subsystem $A$ fails, or any two subsystems of the remaining $B, C, D$ fail. What is the probability that **only** subsystem $A$ has failed, given that the spacecraft $S$ is failing?

***1)*** *The spacecraft S is in working condition if the 4 independent subsystems are not failing:*

$$P\left(S \text{ is in working condition}\right) = \left(1 - \frac{1}{3}\right)^4 = \frac{16}{81}. \tag{9}$$

***2)*** *If A fails also S fails, so* $P\left(\text{only } A \text{ fails}, S \text{ fails}\right) = P\left(\text{only } A \text{ fails}\right)$. *Further* $P\left(S \text{ fails}\right) = 1 - P\left(S \text{ is in working condition}\right)$. *So*

$$P\left(\text{only } A \text{ fails} \mid S \text{ fails}\right) = \frac{P\left(\text{only } A \text{ fails}, S \text{ fails}\right)}{P\left(S \text{ fails}\right)} = \frac{P\left(\text{only } A \text{ fails}\right)}{P\left(S \text{ fails}\right)} = \frac{\frac{1}{3}\left(\frac{2}{3}\right)^3}{1 - \left(\frac{2}{3}\right)^4} = 0.123 \tag{10}$$

***3)***

```
import numpy as np
import pandas as pd

np.random.seed(1)

iter = 1000000

# simulate n times
fail_random = np.random.binomial(1, 1 / 3, size=(iter, 4))

only_a = 0
s_broken = 0

for elem in fail_random:
    # count only A not working
    if elem[0] == 1 and elem[1] + elem[2] + elem[3] == 0:
        only_a += 1
    # count ship broken
    if elem[0] + elem[1] + elem[2] + elem[3] >= 1:
        s_broken += 1

print("P(only A broken | S broken) = {}"
      .format(only_a / s_broken))
```

*The Simulation (repeated 1 Mio. times) returned a result of 0.122968, which differs only slightly from the calculated value and seems therefore plausible.*

***4)*** *The probability is calculated the same way as in 2), but P(S is in working condition) changed. Now S is additionally working if only one of the subsystems $B, C, D$ fails.*

$$P\left(S \text{ is in working condition}\right) = \left(\frac{2}{3}\right)^4 + 3\left(\frac{2}{3}\right)^3 \frac{1}{3}$$

*So now we have*

$$P\left(\text{only A fails} \mid S \text{ fails}\right) = \frac{P\left(\text{only A fails}\right)}{P\left(S \text{ fails}\right)} = \frac{\frac{1}{3}\left(\frac{2}{3}\right)^3}{1-\left(\left(\frac{2}{3}\right)^4 + 3\left(\frac{2}{3}\right)^3 \frac{1}{3}\right)} = \frac{8}{41} = 0.195$$

## Problem 1.3 Optimization and Information Theory [40 Points + 15 Bonus ]

a) **Entropy [5 Points]**

You work for a telecommunication company that uses a system to transmit four different symbols $S_1, S_2, S_3, S_4$ through time. In the current system, each symbol has a probability to occur according to the following table

| | $S_1$ | $S_2$ | $S_3$ | $S_4$ |
|---|---|---|---|---|
| $p_i$ | 0.03 | 0.62 | 0.26 | 0.09 |

Compute the entropy of the system and write the minimum number of bits requires for transmission.

*We compute the entropy of the system by*

$$H\left(p\right) = -0.03\log_2\left(0.03\right) - 0.62\log_2\left(0.62\right) - 0.26\log_2\left(0.26\right) - 0.09\log_2\left(0.09\right) = 1.397$$

*Entropy is a lower bound on the minimum amount of information required to transmit the state of a random variable. So we round up 1.397 and see that a minimum of 2 bits are required for transmission.*

b) **Constrained Optimization [25 Points]**

After an upgrade of the system, your boss asks you to change the probabilities of transmission in order to maximize the entropy. However, the new system has the following constraint

$$4 = \sum_{i=1}^{4} 2p_i i.$$

**1)** Formulate it as a constrained optimization problem. Do you need to include additional constrains beside the one above?
**2)** Write down the Lagrangian of the problem. Use one Lagrangian multiplier per constraint.
**3)** Compute the partial derivatives of the Lagrangian above for each multiplier and the objective variable. Is it easy to solve it analytically?
**4)** Formulate the dual function of this constrained optimization problem. Solve it analytically.
**5)** Name one technique for numerically solve these problems and briefly describe it.

***1)*** *We have to optimize the function* $f\left(p\right) = -\sum_{i=1}^{4} p_i \log\left(p_i\right)$ *with the folowwing constraints:* $\sum_{i=1}^{4} 2p_i i = 4$ *and* $\sum_{i=1}^{4} p_i = 1$. *It is easier to use the natural logarithm instead of the binary logarithm. We can choose this because both functions are constant multiples of each other, therefore both problems have the same optimal values. Now we have the constrainted optimization problem for*

$$\begin{aligned} \max_{p \geq 0} f\left(p\right) &= -\sum_{i=1}^{4} p_i \log\left(p_i\right), \\ s.t.\ g_1\left(p\right) &= \sum_{i=1}^{4} 2p_i i - 4 = 0, \\ g_2\left(p\right) &= \sum_{i=1}^{4} p_i - 1 = 0. \end{aligned}$$

***2)*** *The Lagrangian is as follows*

$$L\left(p, \lambda\right) = f\left(p\right) + \lambda_1 g_1\left(p\right) + \lambda_2 g_2\left(p\right) = \sum_{i=1}^{4} p_i \left(-\log\left(p_i\right) + 2\lambda_1 i + \lambda_2\right) - 4\lambda_1 - \lambda_2.$$

***3)*** *Compute the partial derivatives*

$$\begin{aligned} \nabla_{p_i} L\left(p, \lambda\right) &= -\log\left(p_i\right) + 2\lambda_1 i + \lambda_2 - 1, \\ \nabla_{\lambda_1} L\left(p, \lambda\right) &= \sum_{i=1}^{4} 2p_i i - 4, \\ \nabla_{\lambda_2} L\left(p, \lambda\right) &= \sum_{i=1}^{4} p_i - 1. \end{aligned}$$

*It isn't easy to solve it analytically, because there isn't a general algorithm to solve a system of non-linear equations analytically.*

***4)*** *The dual function is as follows*

$$G(\lambda) = \max_{p \geq 0} L(p, \lambda) = \max_{p \geq 0} \sum_{i=1}^{4} p_i \left(-\log(p_i) + 2\lambda_1 i + \lambda_2\right) - 4\lambda_1 - \lambda_2.$$

*We find the maximum by calculating the gradient and setting it to* 0

$$\nabla_{p_i} L(p, \lambda) = -\log(p_i) + 2\lambda_1 i + \lambda_2 - 1 = 0 \iff p_i = \exp\left(2\lambda_1 i + \lambda_2 - 1\right).$$

*Insert the maximum of* $p_i$ *into* $G(\lambda)$

$$G(\lambda) = \sum_{i=1}^{4} \exp\left(2\lambda_1 i + \lambda_2 - 1\right)\left(-\left(2\lambda_1 i + \lambda_2 - 1\right) + 2\lambda_1 i + \lambda_2\right) - 4\lambda_1 - \lambda_2 = \sum_{i=1}^{4} \exp\left(2\lambda_1 i + \lambda_2 - 1\right) - 4\lambda_1 - \lambda_2.$$

*Now we have to find the maximum by setting the gradient of* $G$ *to* 0*:*

$$\nabla_{\lambda_1} G(\lambda) = \sum_{i=1}^{4} 2i \exp\left(2\lambda_1 i + \lambda_2 - 1\right) - 4 = 0 \iff \sum_{i=1}^{4} i \exp\left(2\lambda_1\right)^i \exp\left(\lambda_2\right) = 2e$$

$$\nabla_{\lambda_2} G(\lambda) = \sum_{i=1}^{4} \exp\left(2\lambda_1 i + \lambda_2 - 1\right) - 1 = 0 \Leftrightarrow 2\sum_{i=1}^{4} \exp\left(2\lambda_1\right)^i \exp\left(\lambda_2\right) = 2e.$$

*Merge the two equations by subtracting the second equation from the first equation*

$$\exp\left(\lambda_2\right) \sum_{i=1}^{4} (i-2) \exp\left(2\lambda_1\right)^i = 0,$$

*since* $\exp\left(\lambda_2\right)$ *can't be 0 this can only be true if* $2X^4 + X^3 - X = 0$ *with* $X = \exp\left(2\lambda_1\right)$ *is true. Solving this equation (with wolfram alpha) provides* $X = 0.657 = \exp\left(2\lambda_1\right)$ *so* $\lambda_1^* = -0.21$ *and finally after rearranging* $2\sum_{i=1}^{4} \exp\left(2\lambda_1\right)^i \exp\left(\lambda_2\right) = 2e$ *we get*

$$\lambda_2^* = \log\left(\frac{e}{\sum_{i=1}^{4} \exp\left(2\lambda_1^*\right)^i}\right) = 0.555$$

***5)*** *We can use the constraints* $g_1$ *and* $g_2$ *to write* $p_3$ *and* $p_4$ *in terms of* $p_1$ *and* $p_2$ *and get an unconstrained optimization problem only depending on* $p_1$ *and* $p_2$*. This problem then can be solved gradient ascent.*
*Locally the gradient points into the direction of steepest ascent, so incrementally setting* $p_{n+1} = p_n + \gamma \nabla f\left(p_n\right)$ *and thus moving into the direction of steepest ascent should lead to the maximum of* $f$ *as long as we set the learning* $\gamma$ *sufficiently small, so we do not move past it.*
*When choosing* $\gamma$ *we also have to consider choosing it small enough, so* $\gamma \nabla f\left(p_n\right)$ *does not lead us away from the domain* $p \geq 0$ *where the optimization problem is defined.*

c) **Numerical Optimization [10 Points]**

Rosenbrock's function (to be minimized) is defined as

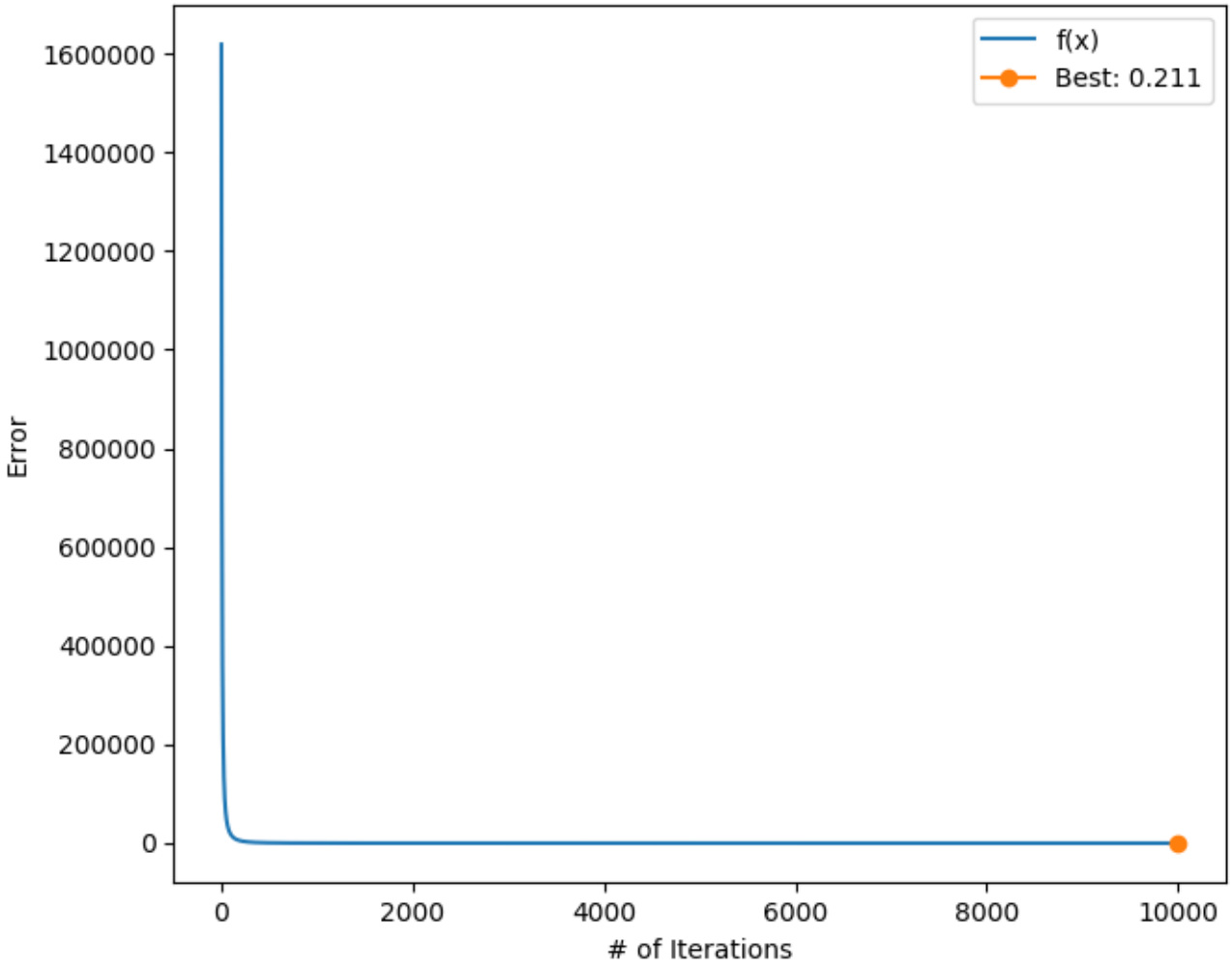
$$f(\boldsymbol{x}) = \sum_{i=1}^{n-1} \left[ 100\left(x_{i+1} - x_i^2\right)^2 + \left(x_i - 1\right)^2 \right].$$

Write in Python a simple gradient descent algorithm and simulate it for 10,000 steps on Rosenbrock's function with $n = 20$. Attach a snippet of your algorithm, discuss the effects of the learning rate and attach a plot of your learning curve with your best learning rate.

*The code snippet implements the gradient descent for the Rosenbrock function with random starting integers between 0 and 10. The best learning rate was 0.000003. Increasing the learning rate returns worse results, because the algorithm overshoots and eventually ends up with nan values. Reducing the learning rate slows down the convergence of the algorithm, it is only able to reach a value close to the minimum after more iterations. Hence, the learning rate adjusts the speed of convergence, however, values too large cause GD to overshot its optimization target.*

```
import numpy as np
from matplotlib import pyplot as plt


def rosen(X):
    # global minimum at (1,) * n
    return sum(100.0 * (X[1:] - X[:-1] ** 2) ** 2
               + (X[:-1] - 1) ** 2)


def d_rosen(X):
    xm = X[1:-1]
    der = np.zeros_like(X)

    # derivative
    der[1:-1] = 200 * (xm - X[:-2] ** 2) \
                - 400 * (X[2:] - xm ** 2) \
                * xm - 2 * (xm - 1)
    # derivative for first element
    der[0] = -400 * X[0] * (X[1] - X[0] ** 2) \
             - 2 * (X[0] - 1)
    # derivative for last element
    der[-1] = 200 * (X[-1] - X[-2] ** 2)
    return der


def gradient_descent(steps, n, lr):
    np.random.seed(2)
    X = np.random.randint(0, 10, n).astype("float64")

    # collect values of rosenbrock function
    history = np.zeros((steps,))

    for i in range(0, steps):
        d_r = d_rosen(X)
        X -= lr * d_r

        # break in case lr is too high
        if all(x == np.nan for x in X):
```

```
            break

        history[i] = rosen(X)

    return X, history


steps = 10000
X, history = gradient_descent(steps=steps,
                              n=20, lr=0.000003)
print("Best Vector: {}".format(X))
plt.plot(history, label="f(x)")
plt.plot(steps, rosen(X), marker="o",
         label="Best: {}".format(np.round(rosen(X), 3)))
plt.xlabel("# of Iterations")
plt.ylabel("Error")
plt.legend()
plt.show()
```



d) **Natural Gradient [10 Bonus Points]**

Let $\theta \in \mathbb{R}^n$ be a parameter vector and $J : \mathbb{R}^n \to \mathbb{R}$ a cost function. The negative gradient $-\nabla J(\theta)$ is sometimes called the *steepest descent direction*. But is it really? To be able to claim that it is *the* steepest descent direction, we should compare it to other descent directions and pinpoint what is so unique about the negative gradient direction.

**Covariant gradient.** A fair way to compare descent directions is to make a small step of fixed length, say $\varepsilon$, in every direction $\Delta\theta$ and check which direction leads to the greatest decrease in $J(\theta)$. Since we assume that the step size is small, we can evaluate the decrease in $J(\theta)$ using its first-order Taylor approximation

$$J(\theta + \Delta\theta) - J(\theta) \approx \nabla J(\theta)^T \Delta\theta.$$

To make precise what we mean by 'small' step size, we need to introduce a norm (or a distance) in the space of parameters $\theta$. A good choice, that among other advantages captures the intuition that some parameters may influence the objective function more than others, is the generic quadratic norm

$$\|\Delta\theta\|^2 = \frac{1}{2}\Delta\theta^T F(\theta)\Delta\theta$$

with a positive-definite matrix $F(\theta)$; note that in general $F$ may depend on $\theta$.

**1)** Find the direction $\Delta\theta$ that yields the largest decrease in the linear approximation of $J(\theta)$ for a fixed step size $\varepsilon$. Does this direction coincide with $-\nabla J(\theta)$? The direction that you found is known as the negative covariant gradient direction.

**Natural gradient.** In statistical models, parameter vector $\theta$ often contains parameters of a probability density function $p(x;\theta)$ (for example, mean and covariance of a Gaussian density); thus, the cost function $J$ depends on $\theta$ indirectly through $p(x;\theta)$. This two-level structure gives a strong hint as to what matrix $F$ to pick for measuring the distance in the parameter space in the most 'natural' way. Namely, one can carry over the notion of 'distance' between probability distributions $p(x;\theta+\Delta\theta)$ and $p(x;\theta)$ (which is known from information theory to be well captured by the Kullback-Leibler divergence) to the distance between the corresponding parameter vectors $\theta + \Delta\theta$ and $\theta$.

**2)** Obtain the quadratic Taylor approximation of the KL divergence from $p(x;\theta)$ to $p(x;\theta+\Delta\theta)$ in the form

$$KL\big(p(x;\theta+\Delta\theta)\,||\,p(x;\theta)\big) \approx \frac{1}{2}\Delta\theta^T F(\theta)\Delta\theta.$$

Covariant gradient with the matrix $F(\theta)$ that you found is known as the natural gradient.

e) **Gradient Descent Variants [5 Bonus Points]**

Throughout this class we have seen that gradient descent is one of the most used optimization techniques in Machine Learning. This question asks you to deepen the topic by conducting some research by yourself.

**1)** There are several variants of gradient descent, namely *batch, stochastic* and *mini-batch*. Each variant differs in how much data we use to compute the gradient of the objective function. Discuss the differences among them, pointing out pros and cons of each one.

**2)** Many gradient descent optimization algorithms use the so-called *momentum* to improve convergence. What is it? Is it always useful?

***1)*** *Batch, stochastic and mini-batch gradient descent try to minimize an unknown function $f$ estimated by functions $f_1, ..., f_n$ at different timesteps.*

***Batch gradient descent*** *calculates $x_{n+1} = x_{n+1} - \frac{\gamma}{n} \sum_{k=1}^{n} \nabla f_k(x)$ in every iteration. This is equivalent to using the average of the „normal" gradient of the functions $f_1, ..., f_n$. However, a problem is, that the term $\nabla f_k(x)$ has to be calcualted for all k. This slows down the overall process of optimization. With todays architectures this could be done in parallel, but it is still computational expensive, especially for large datasets.*

***Stochastic gradient descent*** *select in each iteration a random k, which is used to compute one step of the gradient descent. As a consequence, computing one step is significantly faster than batch gradient descent. Further, the information of the individual $f_k$ is kept. However, the learning speed is random, because each iteration is dependent on chance. But, the algorithm can still converge with a suitable learning rate.*

***Mini-batch gradient descent*** *combines batch and stochastic gradient descent. In each iteration a random subset $f_{k_1}, ..., f_{k_m}$ of the functions $f_1, ..., f_k$ is chosen. Similar to the batch gradient descent $x_{n+1} = x_{n+1} - \frac{\gamma}{m} \sum_{i=1}^{m} \nabla f_{k_i}(x)$ is computed. Further, similar to stochatic gradient descent, only some of the gradients have to be calculated. Therefore, this approch tries to utilize the benifits of both approaches by choosing a random subset of $f_1, ..., f_k$. At least some of the individual information is kept, but using multiple functions, makes the algorithm less prone to chance. Sometimes, the noise in the gradient, i.e. the deviation from the gradient with batch gradient desecent, is seen as additional factor to discover the search space and find a better optimum. In practice, mini-batch gradient descent often results in a faster convergence than batch gradient descent and a smoother progress than stochastic gradient descent.*

***2)*** *When using gradient descent with momentum the updates are calculated by $x_{n+1} = x_n - \Delta x_n$ where $\Delta x_n = \gamma \nabla f\left(x_n\right) + \alpha \Delta x_{n-1}$. In addition to the gradient at $x_n$ momentum also considers the previous updates with a factor $\alpha$. $\alpha$ can also be ajusted in order to achieve faster progress at the beginning and better convergence near the optimum. The momentum keeps the gradient going in the principal direction to the minimum. This allows the algorithm to be more robust and less prone to oscillation ,even when using a larger learning rate or stochastic gradient descent. Often momentum is desribed as ball rolling down a hill. The ball will keep going in the principal direction and will not stop to make a turn. This can also help to overcome local minima.*