Data Analytics and Machine Learning
Department of Informatics
Technical University of Munich

Eexam
Place student sticker here

**Note:**

- During the attendance check a sticker containing a unique code will be put on this exam.
- This code contains a unique number that associates this exam with your registration number.
- This number is printed both next to the code and to the signature field in the attendance check list.

# Maschinelles Lernen

| | | | |
|---|---|---|---|
| **Exam:** | IN2064 / Endterm | **Date:** | Thursday 17th February, 2022 |
| **Examiner:** | Prof. Dr. Stephan Günnemann | **Time:** | 17:00 – 19:00 |

## Working instructions

- This graded exercise consists of **52 pages** with a total of **11** problems and four versions of each problem.
  Please make sure now that you received a complete copy of the graded exercise.
- Use the problem versions specified in your personalized submission sheet on TUMExam. Different problems may have different versions: e.g. Problem 1 (Version A), Problem 5 (Version C), etc. If you solve the wrong version you get **zero** points.
- The total amount of achievable credits in this graded exercise is 96.
- 
- Do **not** submit the problem descriptions (this document) to TUMexam
- You can ignore the “student sticker” box above.

## Problem 1: Probabilistic inference (Version A) (10 credits)

Consider the the following probabilistic model:

$$\mathbb{P}(\theta \mid \lambda, \alpha) = \begin{cases} \frac{\alpha \lambda^{\alpha}}{\theta^{\alpha+1}} & \text{if } \lambda \leq \theta \\ 0 & \text{otherwise} \end{cases}$$

$$\mathbb{P}(x \mid \theta) = \begin{cases} \frac{1}{\theta} & \text{if } 0 \leq x \leq \theta \\ 0 & \text{otherwise} \end{cases}$$

with $\lambda > 0, \alpha > 0$ and a set of observations $\mathcal{D} = \{x_1, ..., x_N\}$ consisting of $N$ samples $x_i \in \mathbb{R}_+$ generated from the above probabilistic model.

0 1 2 3 4 5 6 7 8 9 10

Derive the posterior distribution $\mathbb{P}(\theta \mid \mathcal{D}, \lambda, \alpha)$.

posterior $\propto$ likelihood · prior

$$\mathbb{P}(\theta \mid D, \lambda, \alpha) = \prod_{i=1}^{N} \mathbb{P}(x_i \mid \theta) \cdot \mathbb{P}(\theta \mid \lambda, \alpha)$$

$$= \frac{1}{\theta^N} \cdot 1_{\max(x_i) \leq \theta} \cdot \frac{1}{\theta^{\alpha+1}} \cdot 1_{\max(\lambda) \leq \theta} \cdot \alpha \cdot \lambda^{\alpha}$$

$$\propto \frac{1}{\theta^{N+\alpha+1}} \cdot 1_{\max(x_1, \ldots, x_N, \lambda) \leq \theta}$$

We recognize that the posterior distribution is $\mathbb{P}(\theta \mid \{x_1, \ldots, x_N\}, \lambda, \alpha) =$

$\mathbb{P}(\theta \mid \lambda_{new}, \alpha_{new}) = \text{Pareto}(\lambda_{new} = \max(x_1, \ldots, x_N, \lambda), \alpha_{new} = N + \alpha)$

## Problem 2: Linear regression (Version A) (8 credits)

We want to perform regression on a dataset consisting of $N$ samples $\mathbf{x}_i \in \mathbb{R}^D$ with corresponding targets $y_i \in \mathbb{R}$ (represented compactly as $\mathbf{X} \in \mathbb{R}^{N \times D}$ and $\mathbf{y} \in \mathbb{R}^N$).
Assume that we have fitted a linear regression model and obtained the optimal weight vector $\mathbf{w}^* \in \mathbb{R}^D$ as

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \frac{1}{2} \sum_{i=1}^{N} (\mathbf{w}^T \mathbf{x}_i - y_i)^2.$$

Note that there is no bias term.
Now, assume that we normalize the target variables to have a variance of 1, i.e. $\mathbf{y}_{\text{new}} = \frac{1}{\sigma} \cdot \mathbf{y}$ with $\sigma = Var(\mathbf{y})$, where $Var(\mathbf{y})$ is the sample variance of $\mathbf{y}$.

0 1 2 3 4 5 6 7 8

Find the data matrix $\mathbf{X}_{\text{new}} \in \mathbb{R}^{N \times D}$ such that the solution to the new problem:

$$\mathbf{w}^*_{\text{new}} = \arg\min_{\mathbf{w}} \frac{1}{2} \sum_{i=1}^{N} (\mathbf{w}^T \mathbf{x}_{new,i} - y_{new,i})^2$$

will be the same as the solution to the previous problem i.e. $\mathbf{w}^*_{\text{new}} = \mathbf{w}^*$. Justify your answer.
*Note*: $\mathbf{x}_{new,i}$ is row $i$ of $\mathbf{X}_{\text{new}}$, represented as a column vector.

$w^* = (X^T X)^{-1} X^T y$

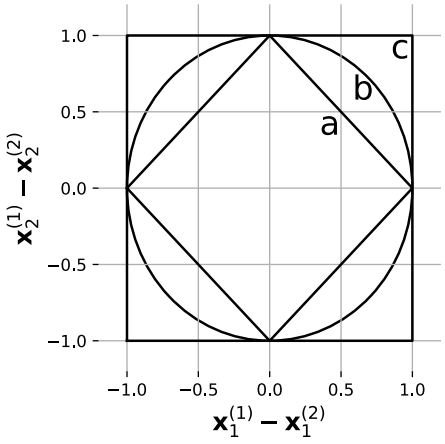
$w^*_{new} = (X_{new}^T X_{new})^{-1} X_{new}^T y_{new} = (X_{new}^T X_{new})^{-1} X_{new}^T \frac{1}{\sigma} y$

$w^*_{new} = w^* \Rightarrow (X_{new}^T X_{new})^{-1} X_{new}^T \frac{1}{\sigma} y = (X^T X)^{-1} X^T y$

$X_{new} = \frac{1}{\sigma} \cdot X$

## Problem 3: k-nearest neighbors (Version A) (3 credits)

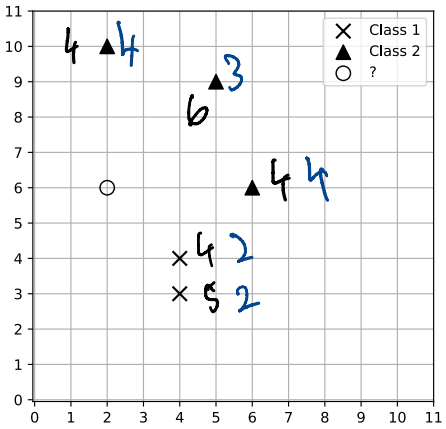
In the following figure we see the unit circles of three distance functions.




0 1 a) Assign each of the following three distance functions its corresponding unit circle (letter a-c) from the figure.

- $L_2$-distance: $\|\mathbf{x}^{(1)} - \mathbf{x}^{(2)}\|_2 = \sqrt{\sum_i \left(\mathbf{x}_i^{(1)} - \mathbf{x}_i^{(2)}\right)^2}$ → b
- $L_1$-distance: $\|\mathbf{x}^{(1)} - \mathbf{x}^{(2)}\|_1 = \sum_i \left|\mathbf{x}_i^{(1)} - \mathbf{x}_i^{(2)}\right|$ → a
- $L_\infty$-distance: $\|\mathbf{x}^{(1)} - \mathbf{x}^{(2)}\|_\infty = \max_i \left|\mathbf{x}_i^{(1)} - \mathbf{x}_i^{(2)}\right|$ → c

In the following figure we see a two-dimensional dataset with two classes. We would like to classify the point $(2, 6)$ marked with a circle using $k$-nearest-neighbors with $k = 3$.




0 1 b) What is the predicted class of the point when using the $L_1$ distance?

Class 2

0 1 c) What is the predicted class of the point when using the $L_\infty$ distance?

Class 1

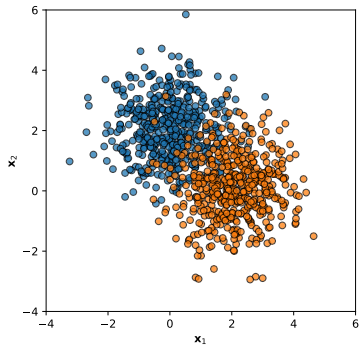
## Problem 4: Classification (Version A) (6 credits)

You are given a balanced dataset with two classes, i.e. $p(y = 0) = p(y = 1)$. Assume that the ground truth class conditional distributions are bivariate Gaussian distributions, i.e. $p(\mathbf{x} \mid c) = \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)$ with mean $\boldsymbol{\mu}_c$ and covariance $\boldsymbol{\Sigma}_c$ for each class $c \in \{0, 1\}$.
Further assume that we can choose between two models to fit the data:

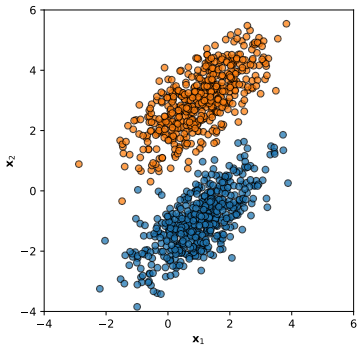
- Linear Discriminant Analysis with Gaussian class conditional distributions
- Naïve Bayes with Gaussian class conditional distributions

For each of the datasets shown below (a, b, c), choose one of the possible options (1,2,3) and justify your answer:

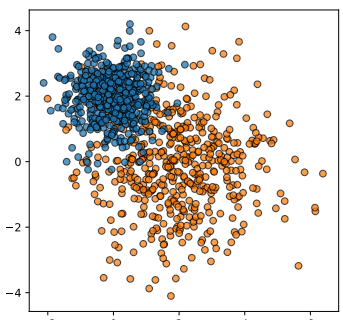
1. We should use Linear Discriminant Analysis.
2. We should use Naïve Bayes.
3. There is no clear reason to prefer one model over the other.




(a)




(b)



(c)

0 1 2

a) Two isotropic bivariate Gaussian distributions with identical covariances matrices
Inconclusive, since both models' assumptions match the data

0 1 2

b) Identical covariance matrices but correlated features
LDA since the covariance matrices match for both classes and the features are correlated

0 1 2

c) Covariance matrices do not match and no apparent correlation between features
Naive Bayes since the features are conditionally independent and variances mismatch between classes ⇒ quadratic decision boundary

## Problem 5: Optimization – Convexity (Version A) (10 credits)

Consider the two functions

$$f(\mathbf{x}) = \max_{i=1,\dots,n} x_i - \min_{i=1,\dots,n} x_i$$

$$g(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} |x_i - \text{median}(\mathbf{x})|$$

with $\mathbf{x} \in \mathbb{R}^n$. You may assume that $n$ is odd.

0 1 2 3 4

a)
Prove or disprove that $f(\mathbf{x})$ is convex in $\mathbf{x}$.

0 1 2 3 4

b)
Prove or disprove that $g(\mathbf{x})$ is convex in $\mathbf{x}$.
*Hint:* $\text{median}(\mathbf{x}) = \arg\min_{t\in\mathbb{R}} \|\mathbf{x} - t\mathbf{1}\|_1$ *with* $\|\cdot\|_1$ *being the sum over* $\mathbf{x}$*'s elements' absolute values.*

0 1 2 3 4 5 6

a)
Consider the function $l_i(x) = x_i$ which is clearly convex in $x$ since it is constant in all dimensions but the $i$-th, in which is linear. From the definition of convexity, it is easy to see that $l_i(\lambda x + (1-\lambda)y) \leq \lambda l_i(x) + (1-\lambda) l_i(y)$ holds
(By induction, rule (2) also holds for more than two arguments). Thus, by rule (2) $\max_{i=1\dots n} x_i$ is convex (plugging in $l_i(x)$ for $f_i(x)$)
Equivalently, $\min_{i=1,\dots,n} x_i$ is concave. By rule (4) it follows that $-\min_{i=1\dots n} x_i$ is convex

Last, by rule (1) it follows that $f(x)$ is convex in $x$

b)
For two arbitrary $x, y \in \mathbb{R}^n$, by definition of $g(x)$: $t_x^* = \text{median}(x) = \arg\min_t \|y - t\cdot 1\|_1$
and $t_y^* = \text{median}(y) = \arg\min_t \|y - t\cdot 1\|_1$, $t_\lambda^* = \text{median}(\lambda x + (1-\lambda) y)$
$= \arg\min_t \|\lambda x + (1-\lambda)y - t\cdot 1\|_n$ for an arbitrary $\lambda \in [0,1]$

$$\lambda g(x) + (1-\lambda) g(y) = \lambda \frac{1}{n}\|x - t_x^* 1\|_n + (1-\lambda)\frac{1}{n}\|y - t_y^* 1\|_n$$

$$= \frac{1}{n}\left[\|\lambda x - \lambda t_x^* 1\|_1 + \|(1-\lambda)y - (1-\lambda)t_y^* 1\|_1\right]$$

triangle inequality $\|a\| + \|b\| \geq \|a+b\|$ →
$$\geq \frac{1}{n}\|\lambda x - \lambda t_x^* 1 + (1-\lambda)y - (1-\lambda)t_y^* 1\|_1$$

definition of $t_\lambda^*$ →
$$\geq \frac{1}{n}\|\lambda x + (1-\lambda)y - t_\lambda^* 1\|_1$$

$$= g(\lambda x + (1-\lambda)y)$$

## Problem 6: Deep learning (Version A) (8 credits)

Suppose $\mathbf{x} \in \mathbb{R}^N$ and $\mathbf{y} \in \mathbb{R}^N$ are two vectors. We define the functions $f : \mathbb{R}^N \times \mathbb{R}^N \to \mathbb{R}^N$ and $g : \mathbb{R}^N \to \mathbb{R}$, and use them to compute

$$\mathbf{z} = f(\mathbf{x}, \mathbf{y})$$
$$t = g(\mathbf{z}).$$

The code below implements the computation of $f$ and $g$, as well as its gradients using backpropagation. Your task is to complete the missing code fragments.

*NOTE: The code is given in Python but you can write the solution in pseudocode as long as it is clear and unambiguous, making sure that the return values have correct shapes.*

```
import numpy as np

class F:
    def forward(self, x, y):
        self.cache = (x, y)
        ##################################################
        # MISSING CODE FRAGMENT #1
        ##################################################
        return out

    def backward(self, d_out):
        # x, y are arrays of shape (N,)
        x, y = self.cache
        d_x = np.sin(y) * d_out
        d_y = x * np.cos(y) * d_out
        return d_x, d_y

class G:
    def forward(self, z):
        self.cache = z
        out = np.mean(z)
        return out

    def backward(self, d_out):
        # z is an array of shape (N,)
        z = self.cache
        ##################################################
        # MISSING CODE FRAGMENT #2
        ##################################################
        return d_z

# Example usage
f, g = F(), G()
x = np.array([1, 2, 3])
y = np.array([4, 5, 6])

z = f.forward(x, y)
t = g.forward(z)

d_z = g.backward(d_out=1.0)
d_x, d_y = f.backward(d_z)
```

out = x · sin(y)

N = len(z)

d_z = np.ones_like(z) / N · d_out

## Problem 6: Deep learning (Version B) (8 credits)

Suppose $\mathbf{x} \in \mathbb{R}^N$ and $\mathbf{y} \in \mathbb{R}^N$ are two vectors. We define the functions $f : \mathbb{R}^N \times \mathbb{R}^N \to \mathbb{R}^N$ and $g : \mathbb{R}^N \to \mathbb{R}$, and use them to compute

$$\mathbf{z} = f(\mathbf{x}, \mathbf{y})$$
$$t = g(\mathbf{z}).$$

The code below implements the computation of $f$ and $g$, as well as its gradients using backpropagation. Your task is to complete the missing code fragments.

*NOTE: The code is given in Python but you can write the solution in pseudocode as long as it is clear and unambiguous, making sure that the return values have correct shapes.*

```
import numpy as np

class F:
    def forward(self, x, y):
        self.cache = (x, y)
        ##################################################
        # MISSING CODE FRAGMENT #1
        ##################################################
        return out

    def backward(self, d_out):
        # x, y are arrays of shape (N,)
        x, y = self.cache
        d_x = np.exp(x) / np.exp(y) * d_out
        d_y = -d_x
        return d_x, d_y

class G:
    def forward(self, z):
        self.cache = z
        out = np.sum(z)
        return out

    def backward(self, d_out):
        # z is an array of shape (N,)
        z = self.cache
        ##################################################
        # MISSING CODE FRAGMENT #2
        ##################################################
        return d_z

# Example usage
f, g = F(), G()
x = np.array([1, 2, 3])
y = np.array([4, 5, 6])

z = f.forward(x, y)
t = g.forward(z)

d_z = g.backward(d_out=1.0)
d_x, d_y = f.backward(d_z)
```

## Problem 6: Deep learning (Version C) (8 credits)

Suppose $\mathbf{x} \in \mathbb{R}^N$ and $\mathbf{y} \in \mathbb{R}^N$ are two vectors. We define the functions $f : \mathbb{R}^N \times \mathbb{R}^N \to \mathbb{R}^N$ and $g : \mathbb{R}^N \to \mathbb{R}$, and use them to compute

$$\mathbf{z} = f(\mathbf{x}, \mathbf{y})$$
$$t = g(\mathbf{z}).$$

The code below implements the computation of $f$ and $g$, as well as its gradients using backpropagation. Your task is to complete the missing code fragments.

*NOTE: The code is given in Python but you can write the solution in pseudocode as long as it is clear and unambiguous, making sure that the return values have correct shapes.*

```
import numpy as np

class F:
    def forward(self, x, y):
        self.cache = (x, y)
        ####################################################
        # MISSING CODE FRAGMENT #1
        ####################################################
        return out

    def backward(self, d_out):
        # x, y are arrays of shape (N,)
        x, y = self.cache
        temp = np.cos(x * y) * d_out
        d_x = y * temp
        d_y = x * temp
        return d_x, d_y

class G:
    def forward(self, z):
        self.cache = z
        out = np.prod(z) # Product of array elements
        return out

    def backward(self, d_out):
        # z is an array of shape (N,)
        z = self.cache
        ####################################################
        # MISSING CODE FRAGMENT #2
        ####################################################
        return d_z

# Example usage
f, g = F(), G()
x = np.array([1, 2, 3])
y = np.array([4, 5, 6])

z = f.forward(x, y)
t = g.forward(z)

d_z = g.backward(d_out=1.0)
d_x, d_y = f.backward(d_z)
```

out = np.sin(x * y)

d_z = np.prod(z) / z * d_out

## Problem 6: Deep learning (Version D) (8 credits)

Suppose $\mathbf{x} \in \mathbb{R}^N$ and $\mathbf{y} \in \mathbb{R}^N$ are two vectors. We define the functions $f : \mathbb{R}^N \times \mathbb{R}^N \to \mathbb{R}^N$ and $g : \mathbb{R}^N \to \mathbb{R}$, and use them to compute

$$\mathbf{z} = f(\mathbf{x}, \mathbf{y})$$
$$t = g(\mathbf{z}).$$

The code below implements the computation of $f$ and $g$, as well as its gradients using backpropagation. Your task is to complete the missing code fragments.

*NOTE: The code is given in Python but you can write the solution in pseudocode as long as it is clear and unambiguous, making sure that the return values have correct shapes.*

```
import numpy as np

class F:
    def forward(self, x, y):
        self.cache = (x, y)
        ##################################################
        # MISSING CODE FRAGMENT #1
        ##################################################
        return out

    def backward(self, d_out):
        # x, y are arrays of shape (N,)
        x, y = self.cache
        d_x = (1 + y) * d_out
        d_y = x * d_out
        return d_x, d_y

class G:
    def forward(self, z):
        self.cache = z
        out = np.dot(z, z) # Dot product
        return out

    def backward(self, d_out):
        # z is an array of shape (N,)
        z = self.cache
        ##################################################
        # MISSING CODE FRAGMENT #2
        ##################################################
        return d_z

# Example usage
f, g = F(), G()
x = np.array([1, 2, 3])
y = np.array([4, 5, 6])

z = f.forward(x, y)
t = g.forward(z)

d_z = g.backward(d_out=1.0)
d_x, d_y = f.backward(d_z)
```

out = Xy + X

d_z = 2 * z * d_out

## Problem 7: Dimensionality reduction (Version A) (12 credits)

We would like to perform binary classification on a dataset $(\mathbf{X}, \mathbf{y})$, where $\mathbf{X} \in \mathbb{R}^{N \times D}$ and $\mathbf{y} \in \{0,1\}^N$.
Assume that we first reduce the dimensionality of $\mathbf{X}$ via PCA to obtain the matrix $\tilde{\mathbf{X}} \in \mathbb{R}^{N \times K}$ (where $K < D$).

0 1 2 3 4 5 6

a) Suppose the original dataset $(\mathbf{X}, \mathbf{y})$ is linearly separable. Is the dataset $(\tilde{\mathbf{X}}, \mathbf{y})$ also guaranteed to be linearly separable? If yes, prove it. If not, provide a counterexample.

No


0 1 2 3 4 5 6

b) Suppose the original dataset $(\mathbf{X}, \mathbf{y})$ is *NOT* linearly separable. Is the dataset $(\tilde{\mathbf{X}}, \mathbf{y})$ guaranteed to *NOT* be linearly separable either?

- If yes (i.e. $(\tilde{\mathbf{X}}, \mathbf{y})$ is *NOT* linearly separable), prove it.
- If no (i.e. $(\tilde{\mathbf{X}}, \mathbf{y})$ may be linearly separable), provide a counterexample.

Yes (Proof by contradiction)

Assume that $(X, y)$ was not linearly separable, but $(\tilde{X}, y)$ was linearly separable

Let $T \in \mathbb{R}^{D \times K}$ be the top-k eigenvectors of the covariance matrix (the matrix we use to perform dimensionality reduction)

Since $(\tilde{X}, y)$ is linearly separable, there is a linear classifier $\mathbb{I}[w^T x \geq b]$ with parameters $w \in \mathbb{R}^K$ and $b \in \mathbb{R}$ that correctly classifies all points in $(\tilde{X}, y)$

Based on the definition of $\tilde{X}$ ($\tilde{X} = T^T X$), this means that any point in $(X)$ can be correctly classified by first performing dimensionality with PCA and then applying the linear classifier specified above

However, this procedure in itself is a linear classifier $\mathbb{I}[(w^T T^T) x \geq b]$ with weight vector $w' = T \cdot w$

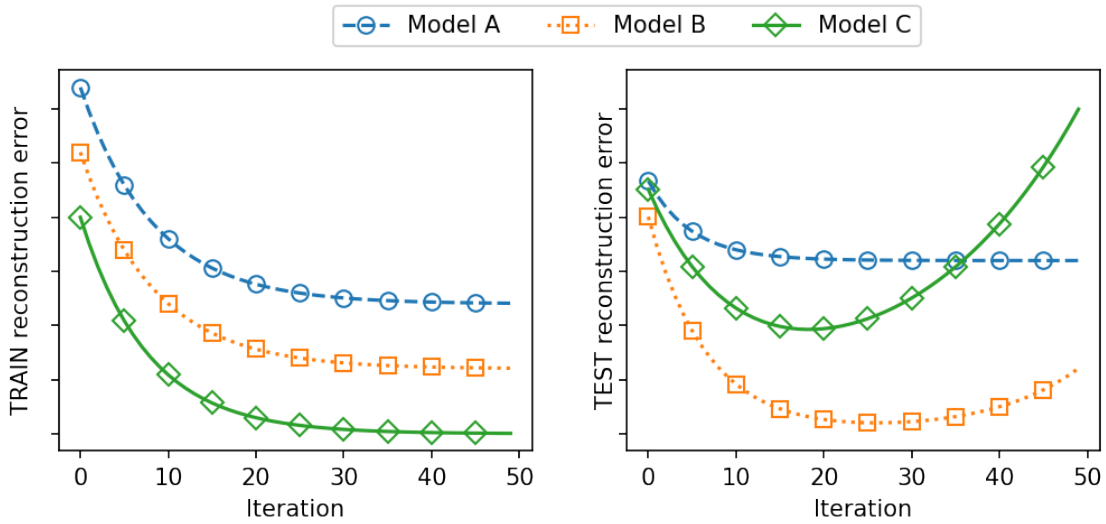
This contradicts our assumption that $(X, y)$ is not linearly separable

## Problem 8: Matrix factorization (Version A) (6 credits)

We would like to perform recommendation using matrix factorization. We have trained 3 latent factor models on the same dataset with gradient descent. These models are identical, except using a different value of $k$ (number of latent factors):

- Model 1: $k = 5$
- Model 2: $k = 20$
- Model 3: $k = 50$

The figure below shows the reconstruction error for different models at each optimization step.




0 1 2 3 4 5 6

Your task is to assign the different models (1, 2, 3) to the loss curves in the figure above (A, B, C). Justify your answer.

A → 5 (1) : Both the training and test losses are high – the model underfits (this may happen if k is too low)

B → 20 (2) : by exclusion

C → 50 (3) : The train loss is low, but the test loss increases sharply after a few iterations – the model overfits (this may happen if k is too high)

## Problem 9: Clustering (Version A) (12 credits)

Consider the following mixture model with $K$ components and a uniform prior over the components:

$$p(z_i = k) = \frac{1}{K} \qquad p(\mathbf{x}_i | z_i = k, \mu_1, ..., \mu_K) = \prod_{d=1}^{D} \frac{(\mu_{kd} x_{id})^{(x_{id}-1)} \exp(-\mu_{kd} x_{id})}{x_{id}!},$$

with parameters $\mu_k = (\mu_{k1}, ..., \mu_{kD})^T \in [0,1]^D$ for $k \in \{1, ..., K\}$.

Suppose we are given a dataset consisting of $N$ data points $\{\mathbf{x}_1, ..., \mathbf{x}_N\}$, where each data point is represented by a $D$-dimensional vector of positive natural number, that is $\mathbf{x}_i = (x_{i1}, ..., x_{iD})^T \in \{1, 2, 3, ...\}^D$.

Derive the M-step of the EM algorithm for the above mixture model, assuming that the responsibilites $\gamma_t(z_{ik})$ are given.

First, we write down the objective function

$$\mathbb{E}_{z \sim \gamma_t(z)} [\log p(x_i z | \mu_1, \dots, \mu_K)] = \sum_{i=1}^{N} \sum_{k=1}^{K} \gamma_t(z_i = k) \log \left( \frac{1}{K} \prod_{d=1}^{D} \frac{(x_{id} \mu_{kd})^{(x_{id}-1)} \exp(-\mu_{kd} x_{id})}{x_{id}!} \right)$$

$$= C + \sum_{i=1}^{N} \sum_{k=1}^{K} \gamma_t(z_i = k) \sum_{d=1}^{D} \left( (x_{id} - 1) \log \mu_{kd} - \mu_{kd} x_{id} \right)$$

$$= C + \sum_{k=1}^{K} \sum_{d=1}^{D} \underbrace{\sum_{i=1}^{N} \gamma_t(z_i = k) \left[ (x_{id} - 1) \log \mu_{kd} - \mu_{kd} x_{id} \right)}_{=: L_{kd}}$$

$$= C + \sum_{k=1}^{K} \sum_{d=1}^{D} L_{kd}$$

We see that the objective can be decomposed as a sum, where each parameter $\mu_{kd}$ is only encountered in a single term $L_{kd}$

Therefore, to find the optimal $\mu_{kd}$, we only need to maximize the respective term $L_{kd}$

$$\frac{\partial L_{kd}}{\partial \mu_{kd}} = \frac{\partial}{\partial \mu_{kd}} \left( \sum_{i=1}^{N} \gamma_t(z_i = k) \left( (x_{id} - 1) \log \mu_{kd} - \mu_{kd} x_{id} \right) \right)$$

$$= \sum_{i=1}^{N} \gamma_t(z_i = k) \left( \frac{x_{id} - 1}{\mu_{kd}} - x_{id} \right) \overset{!}{=} 0$$

$$\mu_{kd} = \frac{\sum_{i=1}^{N} \gamma_t(z_i = k) (x_{id} - 1)}{\sum_{i=1}^{N} \gamma_t(z_i = k) x_{id}}$$

**Additional space for solutions–clearly mark the (sub)problem your answers are related to and strike out invalid solutions.**