Programming Machine Learning Lab

Exercise 12

General Instructions:

1. You need to submit the PDF as well as the filled notebook file.
2. Name your submissions by prefixing your matriculation number to the filename. Example, if your MR is 12345 then rename the files as **"12345_Exercise_12.xxx"**
3. Complete all your tasks and then do a clean run before generating the final PDF. (*Clear All Ouputs* and *Run All* commands in Jupyter notebook)

Exercise Specific instructions::

1. You are allowed to use only NumPy and Pandas (unless stated otherwise). You can use any library for visualizations.

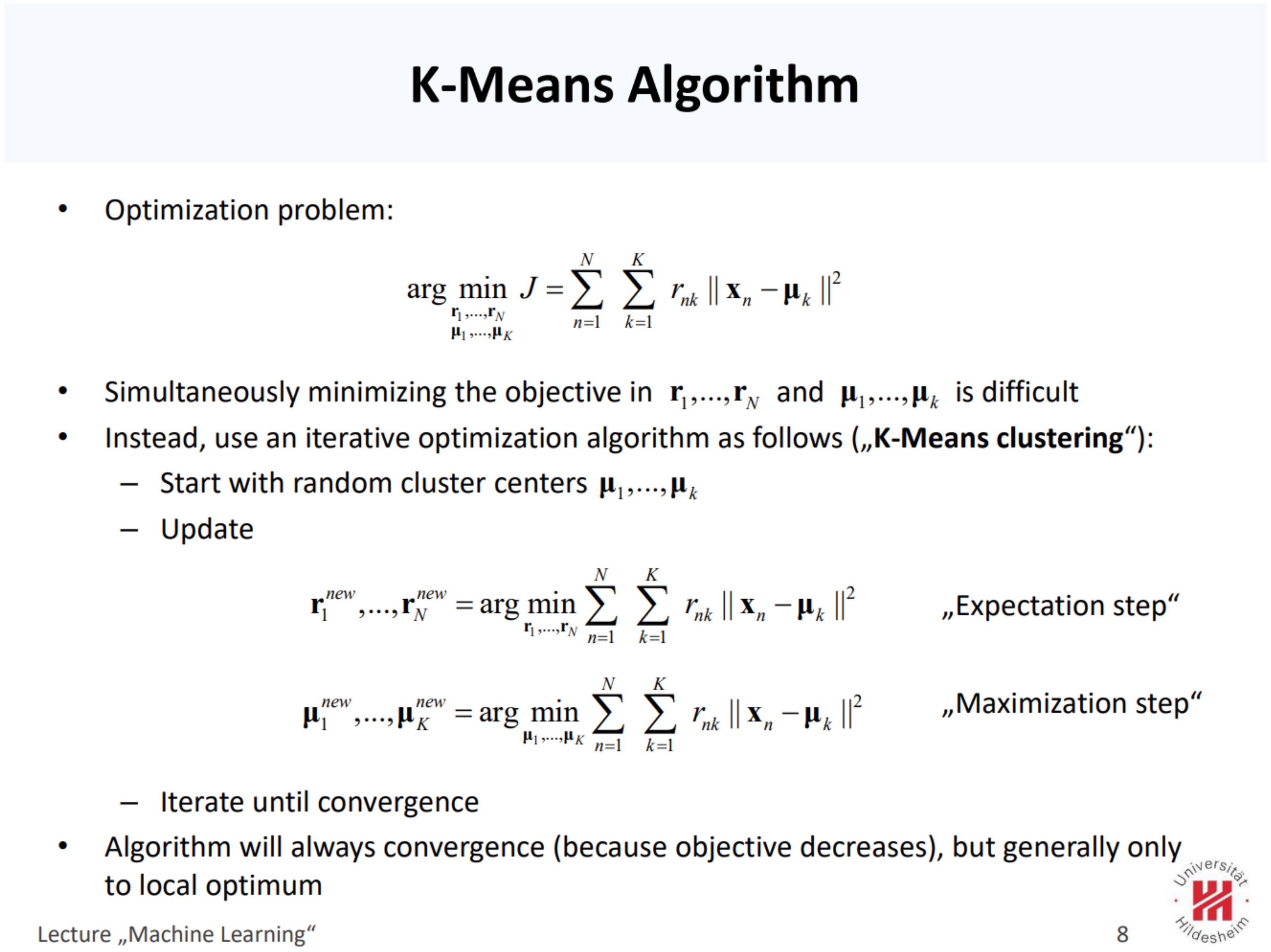
Part 1

K-Means

K-Means algorithm splits a dataset $X \in \{x_1, \ldots, x_N\}$ into K many partitions, where each $X_k \subseteq X \quad \forall k \in \{1, \ldots, K\}$. Clustering algorithms like the *K-Means* is a useful technique when the true labels are unknown. Or in other words, we are basically interested in analyzing patterns within the data and make useful inferences.

In this task, you will implement a *K-Means* algorithm from scratch using the dataset **"HTRU_2.csv"**. The dataset contains 8 continuous variables describing a pulsar candidate (https://archive.ics.uci.edu/ml/datasets/HTRU2). The task is to identify (K) clusters that might best describe the classes within the data.

The basic algorithm is given by:



K-Means Algorithm

- Optimization problem:

$$\underset{\substack{\mathbf{r}_1,\dots,\mathbf{r}_N \\ \boldsymbol{\mu}_1,\dots,\boldsymbol{\mu}_K}}{\arg\min}\ J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \,||\, \mathbf{x}_n - \boldsymbol{\mu}_k \,||^2$$

- Simultaneously minimizing the objective in $\mathbf{r}_1,\dots,\mathbf{r}_N$ and $\boldsymbol{\mu}_1,\dots,\boldsymbol{\mu}_k$ is difficult
- Instead, use an iterative optimization algorithm as follows („**K-Means clustering**“):
  - Start with random cluster centers $\boldsymbol{\mu}_1,\dots,\boldsymbol{\mu}_k$
  - Update

$$\mathbf{r}_1^{new},\dots,\mathbf{r}_N^{new} = \underset{\mathbf{r}_1,\dots,\mathbf{r}_N}{\arg\min} \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \,||\, \mathbf{x}_n - \boldsymbol{\mu}_k \,||^2 \qquad \text{„Expectation step“}$$

$$\boldsymbol{\mu}_1^{new},\dots,\boldsymbol{\mu}_K^{new} = \underset{\boldsymbol{\mu}_1,\dots,\boldsymbol{\mu}_K}{\arg\min} \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \,||\, \mathbf{x}_n - \boldsymbol{\mu}_k \,||^2 \qquad \text{„Maximization step“}$$

  - Iterate until convergence
- Algorithm will always convergence (because objective decreases), but generally only to local optimum

Lecture „Machine Learning“ 8

Universität Hildesheim

```
In [ ]: ### Write your code here
```

Evaluation

Since K-means is an unsupervised approach, we need to find a method of finding out the best K for the task. There are multiple methods of determining the ideal K for a given dataset. Read up and implement atleast one of these methods from scratch:

1. Elbow
2. Average silhouette method
3. Gap statistic method

- Create a visualization to show the statistics for the selected method vs different number of clusters K. Comment on how the best *K* is found using the method of your choice.

In case you did not do the first part of this question, you can use sklearn implementation of KMeans clustering for this. You can also implement any other method that you find to determine the optimal K

```
In [ ]: ### Write your code here
```

Visualization

- Principal Components Analysis (PCA) is a widely used method for reducing the number of dimensions to a low dimensional representation of the data. Create a function that performs PCA on the given dataset. (You are allowed to use numpy.linalg.svd for single value decomposition).
- Use (PCA) to reduce the dimensionality of the data and represent the clusters (K-means) as both 2D and 3D visualizations.

In case you did not do the first part of this question, you can use sklearn implementation of KMeans clustering for this.

```
In [ ]: ### Write youir code here
```

Part 2

Gaussian Mixtures

In this exercise, you are required to implement **Gaussian Mixtures** for Soft Clustering using the Expectation Maximization (EM) Algorithm. We will use the same data as from last question.

The basic algorithm is given by:

EM-Algorithm: Summary

- Summary of EM-Algorithm:
  - Start with randomly initialized $\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}$
  - For *t*=0,1,2,... until convergence:
    - Expectation step: compute responsibilities

$$\gamma(z_{nk}) = \mathbb{E}[z_{nk} \mid \mathbf{X}, \boldsymbol{\theta}_t] = p(z_{nk} = 1 \mid \mathbf{X}, \boldsymbol{\theta}_t)$$

    - Maximization step:

$$\pi_k^* = \frac{N_k}{N} \qquad N_k = \sum_{n=1}^{N} \gamma(z_{nk})$$

$$\boldsymbol{\mu}_k^* = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk}) \mathbf{x}_n$$

$$\boldsymbol{\Sigma}_k^* = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk}) (\mathbf{x}_n - \boldsymbol{\mu}_k^*)(\mathbf{x}_n - \boldsymbol{\mu}_k^*)^T$$

  - Gaussian mixture model with EM can be seen as a soft version of K-Means

- Initialize clusters by drawing randomly from a uniform distribution.
- Clearly specify the Expectation step and the Maximization step.

In [ ]:
```
import pandas as pd
from sklearn.mixture import GaussianMixture
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
```

In [ ]:
```
### Write your code here

df = pd.read_csv("HTRU_2.csv", header=0, index_col=None)
df.describe()
```

Out[ ]:

	Mean of the integrated profile	Standard deviation of the integrated profile	Excess kurtosis of the integrated profile	Skewness of the integrated profile	Mean of the DM-SNR curve	Standard deviation of the DM-SNR curve	Excess kurtosis of the DM-SNR curve	Skewness of the DM-SNR curve
count	17898.000000	17898.000000	17898.000000	17898.000000	17898.000000	17898.000000	17898.000000	17898.000000
mean	111.079968	46.549532	0.477857	1.770279	12.614400	26.326515	8.303556	104.857709
std	25.652935	6.843189	1.064040	6.167913	29.472897	19.470572	4.506092	106.514540
min	5.812500	24.772042	-1.876011	-1.791886	0.213211	7.370432	-3.139270	-1.976976
25%	100.929688	42.376018	0.027098	-0.188572	1.923077	14.437332	5.781506	34.960504
50%	115.078125	46.947479	0.223240	0.198710	2.801839	18.461316	8.433515	83.064556
75%	127.085938	51.023202	0.473325	0.927783	5.464256	28.428104	10.702959	139.309330
max	192.617188	98.778911	8.069522	68.101622	223.392141	110.642211	34.539844	1191.000837

In [ ]:
```
def initialize_parameters(n_components, n_features):
    # Initialize means randomly
    means = np.random.rand(n_components, n_features)

    # Initialize covariance matrices as identity matrices
    covariances = [np.eye(n_features) for _ in range(n_components)]

    # Initialize mixing coefficients uniformly
    mixing_coeffs = np.ones(n_components) / n_components
```

```python
    return means, covariances, mixing_coeffs

def compute_gaussian_pdf(x, mean, covariance):
    # Compute Gaussian probability density function
    det_cov = np.linalg.det(covariance)
    inv_cov = np.linalg.inv(covariance)
    exponent = -0.5 * np.dot(np.dot((x - mean), inv_cov), (x - mean).T)
    return (1 / np.sqrt((2 * np.pi) ** len(x) * det_cov)) * np.exp(exponent)

def e_step(X, means, covariances, mixing_coeffs):
    N, n_features = X.shape
    n_components = len(means)
    responsibilities = np.zeros((N, n_components))

    for i in range(N):
        for k in range(n_components):
            responsibilities[i, k] = mixing_coeffs[k] * compute_gaussian_pdf(X[i], means[k], covariances[k])
        responsibilities[i] /= np.sum(responsibilities[i])

    return responsibilities

def m_step(X, responsibilities):
    N, n_features = X.shape
    n_components = responsibilities.shape[1]

    means = np.zeros((n_components, n_features))
    covariances = [np.zeros((n_features, n_features)) for _ in range(n_components)]
    mixing_coeffs = np.zeros(n_components)

    for k in range(n_components):
        N_k = np.sum(responsibilities[:, k])
        means[k] = np.sum(responsibilities[:, k].reshape(-1, 1) * X, axis=0) / N_k
        for i in range(N):
            diff = X[i] - means[k]
            covariances[k] += responsibilities[i, k] * np.outer(diff, diff)
        covariances[k] /= N_k
        mixing_coeffs[k] = N_k / N

    return means, covariances, mixing_coeffs

def gmm(X, n_components, max_iters=50, tol=1e-6):
    means, covariances, mixing_coeffs = initialize_parameters(n_components, X.shape[1])

    for epoch in range(max_iters):
        prev_log_likelihood = 0
        responsibilities = e_step(X, means, covariances, mixing_coeffs)
        means, covariances, mixing_coeffs = m_step(X, responsibilities)

        # Compute log-likelihood
        log_likelihood = np.sum(np.log(np.sum(responsibilities, axis=1)))
        mae = np.abs(log_likelihood - prev_log_likelihood)
        if np.abs(log_likelihood - prev_log_likelihood) < tol:
            break
        prev_log_likelihood = log_likelihood

        print(f"epoch {epoch+1} of {max_iters} - mae: {mae}")

    return means, covariances, mixing_coeffs

# Example usage:
# Assuming df contains your data
X = df.values
n_components = 3
means, covariances, mixing_coeffs = gmm(X, n_components)
```

Visualization

- Use (PCA) to reduce the dimensionality of the data and represent the clusters (K-means) as both 2D and 3D visualizations.

In case you did not implement this in the first question, you can use sklearn implementation of PCA for this.

In [ ]:

```
### Write your code here
def perform_pca(X, n_components):
    # Standardize the data (optional but recommended)
    X_std = (X - np.mean(X, axis=0)) / np.std(X, axis=0)

    # Calculate the covariance matrix
    cov_matrix = np.cov(X_std, rowvar=False)

    # Perform SVD
    _, _, Vt = np.linalg.svd(cov_matrix)
```

```python
    # Select the top n_components principal components
    top_components = Vt[:n_components]

    # Project the data onto the selected components
    reduced_data = np.dot(X_std, top_components.T)

    return reduced_data

# Example usage:
# Assuming X is your data matrix (N samples x D features)
# Set n_components to the desired number of dimensions
n_components = 2
reduced_data = perform_pca(X, n_components)


# Compute the likelihood of each data point for each Gaussian component
gmm = GaussianMixture(n_components=n_components, means_init=means, covariances_init=covariances, weights_init=mixing_
gmm.fit(reduced_data)
cluster_assignments = gmm.predict(reduced_data)


# Create a 2D scatter plot
if reduced_data.shape[1] == 2:
    plt.figure(figsize=(8, 6))
    plt.scatter(reduced_data[:, 0], reduced_data[:, 1], c=cluster_assignments, cmap='viridis')
    plt.title("2D Scatter Plot")
    plt.xlabel("Principal Component 1")
    plt.ylabel("Principal Component 2")
    plt.colorbar(label="Cluster")
    plt.show()

# Create a 3D scatter plot
elif reduced_data.shape[1] == 3:
    fig = plt.figure(figsize=(10, 8))
    ax = fig.add_subplot(111, projection='3d')
    ax.scatter(reduced_data[:, 0], reduced_data[:, 1], reduced_data[:, 2], c=cluster_assignments, cmap='viridis')
    ax.set_title("3D Scatter Plot")
    ax.set_xlabel("Principal Component 1")
    ax.set_ylabel("Principal Component 2")
    ax.set_zlabel("Principal Component 3")
    plt.colorbar(label="Cluster")
    plt.show()
```

```
else:
    print("Invalid number of dimensions after PCA. Please check your PCA implementation.")
```