OST
Ostschweizer
Fachhochschule

# Lab 11

FTP MachLe MSE
HS 2024

## Unsupervised Learning: Clustering

Machine Learning
WÜRC

After this unit, ...

**Lernziele/Kompetenzen**

- you know the *three* clustering algorithms: *k-means*, *dbscan* and a*agglomerative clustering* (using average, complete or Ward linkage).
- you are able to explain the working principle of *k-means*, *dbscan* and *agglomerative clustering*, their advantages and disadvantages and to apply them to data using `scikit-tlearn` in Python.
- you are able to plot the `inertia` and to determine the elbow point of this curve to find an optimum number of clusters as *hyperparameter*.
- you know the way how to *evaluate* a cluster algorithm using *metrics*, namely using `ARI` (*adjusted rand index*), `NMI` (*normalized mutual information*), `SC` (*silhouette score*) and *inertia*.
- you are able to correctly *scale* the data before clustering is applied especially (MinMax, StandardScaler, RobustScaler) or to meaningfully `transform` the data (eg. using `PCA`, `t-SNE` or `NMF`) before a clustering algorithm is applied.
- you know what a *Gaussian Mixture Model* `GMM` is and how the *expectation maximization algorithm* (`EM` algorithm) works. You are able to interpret the `kMeans` algorithm as a form of an `EM` algorithm with an E-step and and M-Step.
- your are able to apply clustering on the faces dataset (agglomerative, k-means and dbscan) to detect and *group* similar faces.
- your are able to apply a *hierarchical cluster analysis* on a voting dataset.

## 1. Clustering Algorithms [M,I]

This clustering algorithm initially assumes that each data instance represents a single cluster.

| Welche der folgenden Aussagen sind wahr und welche falsch? | wahr | falsch |
|---|---|---|
| **a)** agglomerative clustering | ○ | ○ |
| **b)** t-SNE | ○ | ○ |
| **c)** k-means clustering | ○ | ○ |
| **d)** expectation maximization | ○ | ○ |

## 2. Elbow Curve and `sklearn.cluster.KMeans` [A,I]

Using the following code lines, you can generate two-dimensional data clusters that can be used for testing clustering algorithms. In this exercise, you will learn how to apply k-means and how to determine the optimum number of clusters using the elbow criterium of the inertia plot.

```
from sklearn.datasets import make_blobs

X, y = make_blobs(n_samples=250,
  n_features=8,
  centers=8,
  cluster_std=0.85,
  shuffle=True,
  random_state=0)
```

**a)** Generate a distribution of 8 clusters with 250 samples and plot them as a scatterplot. How many clusters do you recognize with your eye. Try to change the cluster standard deviation `cluster_std` until it will be hard for you to discriminate the 8 different clusters.

**b)** Import the method `KMeans` from `sklearn.cluster`. Instantiate a model `km` with 8 clusters (`n_clusters=8`). Set the maximum number of iterations to `max_iter=300` and `n_init=10`. Fit the model to the data and predict the cluster label using `km.fit_predict(X)`.
*Hint: One way to deal with convergence problems is to choose larger values for `tol`, which is a parameter that controls the tolerance with regard to the changes in the within-cluster sum-squared-error to declare convergence. Try a tolerance of 1e-04.*

**c)** Use the function `PlotClusters` to display the clustered data.

```
#used for cycling through all defined colors
from matplotlib import colors as mcolors
colors = dict(mcolors.BASE_COLORS, **mcolors.CSS4_COLORS)
ColorNames=list(colors.keys())
HSV=colors.values()

def PlotClusters(X,y, km):

  for ClusterNumber in range(km.n_clusters):
    plt.scatter(X[y_km == ClusterNumber, 0],
      X[y_km == ClusterNumber, 1],
      s=50, c=ColorNames[ClusterNumber+1],
      marker='s', edgecolor='black',
      label='cluster {0}'.format(ClusterNumber+1))
```

```
plt.scatter(km.cluster_centers_[:, 0],
km.cluster_centers_[:, 1],
s=250, marker='*',
c='red', edgecolor='black',
label='centroids')
plt.legend(scatterpoints=1)
plt.grid()
plt.tight_layout()
#plt.savefig('Clusters.png', dpi=300)
plt.show()
```

**d)** Vary the number of clusters `n_clusters=8` in your `KMeans` clustering algorithm from 4 to 8 and display each time the result using the function `PlotClusters`.

**e)** Vary in a `for` loop the number of clusters from `n_clusters=8` to `n_clusters=15` and cluster the data each time using the `km.fit_predict` method. Read out the inertia `km.inertia_` and store it in a list called `distortions` as function of the number of clusters using the `append` method. Display the inertia as function of the number of clusters and determine the optimum number of clusters from the elbow curve.

**f)** Without explicit defintion, a random seed is used to place the initial centroids, which can sometimes result in bad clusterings or slow convergence. Another strategy is to place the initial centroids far away from each other via the `k-means++` algorithm, which leads to better and more consistent results than the classic k-means. This can be selected in `sklearn.cluster.KMeans` by setting `init=k-means++`.

(D. Arthur and S. Vassilvitskii. k-means++: The Advantages of Careful Seeding. In Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms, pages 1027–1035. Society for Industrial and Applied Mathematics, 2007). `http://ilpubs.stanford.edu:8090/778/1/2006-13.pdf`

## 3. k-Means, Gaussian Mixture Models and the EM algorithm [A,II]

Open the Jupyter notebook `Lab11_A3_EM_KMeans_MixtureModels.jpynb` that was originally created by *Sebastian Raschka* (`https://sebastianraschka.com/books.html`) and that also contains code from Jake Vanderplas' Python Data Science Handbook (`https://github.com/jakevdp/PythonDataScienceHandbook`). Work through the code and answer the following questions.

**a)** What are the basic assumptions of the k-Means algorithm? How does it work? Study the implementation in cell [8] and play with the interacitve code from Jake Vanderplas.

**b)** What are the *limitations* of kMeans? What can be done to overcome these limitations?

**c)** What is *hard* and what is *soft* clustering?

**d)** Explain how the `expecation maximization algorithm` (`EM`) works and list a five data science applications where it can be used. How would you prove that the `EM` algorithm converges to the maximum likelihood estimate of the hypothesis made?

**e)** What is a *Gaussian mixture model* (`GMM`)? What are the advantages of soft clustering using a `GMM` compared to kMeans?

### 4. Image compression using `kMeans` [A, II]

Clustering can be used to reduce colors in an image. Similar colors will be assigned to the same cluster label or color palette. In In the following exercise, you will load an image as a $[w, h, 3]$ `numpy.array` of type `float64` , where $w$ and $h$ are the width and height in pixels respecively. The last dimension of the three dimensional array are three the RGB color channels. Using `kMeans`, we will reduce the color depth from 24 bits to 64 colors (6 bits) and to 16 colors (4 bits).

**a)** Start by reading in an image from the Python imaging library `PIL` (`https://en.wikipedia.org/wiki/Python_Imaging_Library`) in your Jupyter notebook.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.utils import shuffle
from PIL import Image

# First we read and flatten the image.
original_img = np.array(Image.open('tree.jpg'), dtype=np.float64) / 255
print(original_img.shape)
original_dimensions = tuple(original_img.shape)
width, height, depth = tuple(original_img.shape)
```

**b)** Flatten the image to a $[w \cdot h, 3]$-dimensional numpy.array and shuffle the pixels using `sklearn.utils.shuffle`.

**c)** Create an instance of the `kMeans` class called `estimator`. Use the fit method of `kMeans` to create sixty-four clusters `(n_clusters=64)` from a sample of one thousand randomly selected colors, e.g. the first 1000 colors of the shuffled pixels. The new color palette is given by the cluster centers that are accessible in `estimator.cluster_centers_`.

**d)** Assign the cluster labels to each pixel in the original image using the `.predict` method of your `kMeans` instance. Now, you know to which color in your reduced palette each pixel belongs to.

**e)** Loop over all pixels and assign the new color palette corresponding to the label of the pixel and create a new, reduced color picture. Plot the images using `plt.imshow`, compare the original image and the 64 color image. Try the same with 32 and 16 colors.

## 5. Detecting similar faces using DBSCAN [A, II]

The *labelled* faces dataset of sckit-learn contains gray scale images of 62 differnet famous personalites from politics. In this exercise, we assume that there are no target labels, i.e. the names of the persons are unknown. We want to find a method to cluster similar images. This can be done using a dimensionality reduction algorithm like `PCA` for feature generation and a subsequent clustering e.g. using `DBSCAN`.

**a)** Open the Jupyter notebook `DBSCAN_DetectSimilarFaces.jpynb` and have a look at the first few faces of the dataset. Not every person is represented equally frequent in this *unbalanced* dataset. For classification, we would have to take this into account. We extract the first 50 images of each person and put them into a flat array called `X_people`. The correspinding targets ($y$-values, names), are storeed in the `y_people` array.

**b)** Apply now a principal component analysis `X_pca=pca.fit_transform(X_people)` and extract the first 100 components of each image. reconstruct the first 10 entries of the dataset using the 100 components of the PCA transformed data by applying the `pca.inverse_transform` method and reshaping the image to the original size using `np.reshape`. What is the minimum number of components necessary such that you recognize the persons? Try it out.

**c)** Import `DBSCAN` class from `sklearn.cluster`, generate an instance called `dbscan` and apply it to the pca transformed data `X_pca` and extract the cluster labels using `labels = dbscan.fit_predict(X_pca)`. Use first the standard parameters for the method and check how many unique clusters the algorithm could find by analyzing the number of unique entries in the predicted cluster labels.

**d)** Change the parameter `eps` of the `dbscan` using `dbscan(min_samples=3, eps=5)`. Change the value of `eps` in the range from 5 to 10 in steps of 0.5 using a `for` loop and check for each value of `eps` how many clusters could be determined.

**e)** Select the value of *eps* where the numbers of clusters found is maximum and plot the members of the clusters found using the follwing python code.

```
dbscan = DBSCAN(min_samples=3, eps= ...)
labels = dbscan.fit_predict(X_pca)

for cluster in range(max(labels) + 1):
mask = labels == cluster
n_images =  np.sum(mask)
fig, axes = plt.subplots(1, n_images, figsize=(n_images * 1.5, 4),
subplot_kw={'xticks': (), 'yticks': ()})
for image, label, ax in zip(X_people[mask], y_people[mask], axes):

ax.imshow(image.reshape(image_shape), vmin=0, vmax=1)
ax.set_title(people.target_names[label].split()[-1])
```