Machine Learning I

Chapter 10 - Clustering (Unsupervised)

Prof. Dr. Sandra Eisenreich

December 21 2023

Hochschule Landshut

Applications of Clustering

???

Types of clustering

With respect to the output:

- hard clustering: assigning each instance to a single cluster.
- soft clustering: assigning each instance to each cluster with probability = a certain score (e.g. distance between instance and the centroid, or a similarity score)

With respect to the method: (nee next slides)

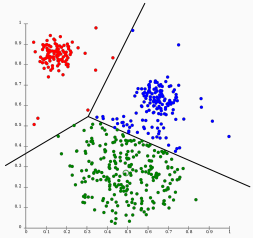
- Centroid-based clustering
- Density-based clustering
- Distribution-based clustering
- Hierarchical clustering (HCA)

We will cover:

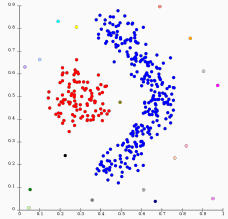
- K-Means Clustering (Centroid-based)
- Gaussian Mixture Models (Distribution-based)
- DBSCAN (Density-based)

Clustering methods

Centroid-based clustering: organizes data into non-hierarchical clusters by finding their centroids. (effective, fast, but sensitive to initial conditions and outliers), e.g K-Means (fastest clustering algorithm with $O(n)$)

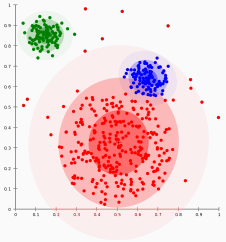


Density-based clustering: organize data into non-hierarchical arbirtary-shaped clusters by finding regions of high data density, e.g. DBSCAN. (can find all sorts of shapes, but difficulties with data of varying density and high dimensions)

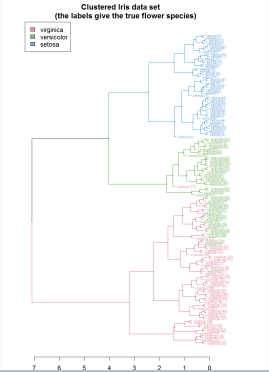


Types of Clustering ctd.

Distribution-based clustering: Describe the data as a sum of distributions (e.g. several Gaussian normal distributions centered around some points), e.g. Gaussian Mixture Model. If you do not know the type of distribution of the data: use something else.



Hierarchical clustering (HCA): Grow a tree in an unsupervised way with some loss function (e.g. distance, there are many different approaches), top-down or bottom-up; but slow ($O(n^3)$ or $O(n^2)$); so we will not talk about this here. Use for data with hierarchical structures you want to uncover in full.



Performance measure: Silhouette score

Basic idea: In a good cluster model, you want to have

- all instances in one cluster close to each other, and
- each cluster far away from all other clusters.

The silhouette coefficient of an instance is defined as

$$\frac{b-a}{\max(a,b)} \in [-1,1], \text{where}$$

$a =$ mean distance to all other instances in the same cluster

$b =$ mean distance to all intances in the closest other cluster

You want a small and b big, so a silhouette coeffictient close to 1 is good. Then the silhouette score is the average of all silhouette coefficients over all instances.

K-Means Clustering

Overview

- task: **centroid-based clustering**
- data: no labels, i.e. **unsupervised** learning
- model: **non-parametric** - it does not make assumptions about the data distribution
- complexity: simple and fast model, but needs several iterations to reach a good optimum.

When to use:

- The data are clearly clustered, and the clusters are more or less circular
- If you know approximately how many clusters you have in your data or do not mind training several models
- If you need speed: K-Means is one of the fastest clustering algorithms ($O(m)$). But: it does not have a global minimum, so it might get stuck in a local minimum.

K-Means

- The center of a cluster is called centroid.
- (loss function) distortion/inertia = the mean squared distance between each instance $\mathbf{x}$ and its closest centroid $\mu(\mathbf{x})$:

$$\sum_{i=1}^{m} \|\mathbf{x}^{(i)} - \mu(\mathbf{x}^{(i)})\|_2^2$$

Idea of K-Means: find K centroids $\mu_1, \ldots, \mu_K$ in space such that distortion/inertia is minimal.

K-**means clustering** is the algorithm with

- Prediction function = cluster with the closest centroid: $\mu(\mathbf{x}) = \text{argmin}_k \|\mathbf{x}^{(i)} - \mu(\mathbf{x}^{(i)})\|_2$.
- Cost function = distortion/inertia $\sum_{i=1}^{m} \|\mathbf{x}^{(i)} - \mu(\mathbf{x}^{(i)})\|_2^2$

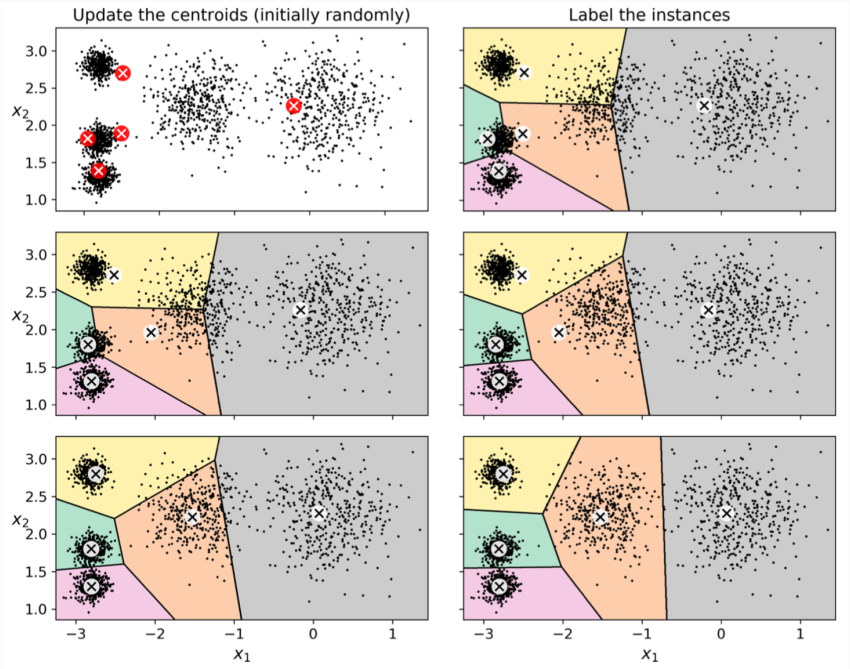
The algorithm

Suppose you were given the centroids, then you could label all the instances in the dataset by assigning each of them to the cluster whose centroid is closest.

Conversely, if you were given all instance labels, you could locate the centroids by computing the mean of the instances for each cluster.

Since you have neither, iterate: ?

The algorithm is guaranteed to converge, but it may not converge to the right solution, but a local optimum (depends on the initialization).


Update the centroids (initially randomly)
Label the instances
x_2
x_1
3.0
2.5
2.0
1.5
1.0
−3
−2
−1
0
1

Complexity

The K-means algorithm is $O(tKmn)$, where t = number of iterations, K = number of clusters, m = number of instances, n= dimension of instances.

Why? For each iteration (times t): ?

K-means with Scikit-Learn

```
from sklearn.cluster import KMeans


k=5
kmeans = KMeans(n_clusters=k)
y_pred = kmeans.fit_predict(X)
```

Output the centroids of the algorithm with `kmeans.cluster_centers_`. You can get the labels of X via `.labels()`, assign new instances to the correct cluster with `.predict()`, and measure the distance from each instance to every centroid with `.transform()`. You get the negative inertia (i.e. the bigger the better) with `.score()`.

Initialization

Since there might be many local minima, different initializations of the centroids lead to different clusters. 2 Solutions:

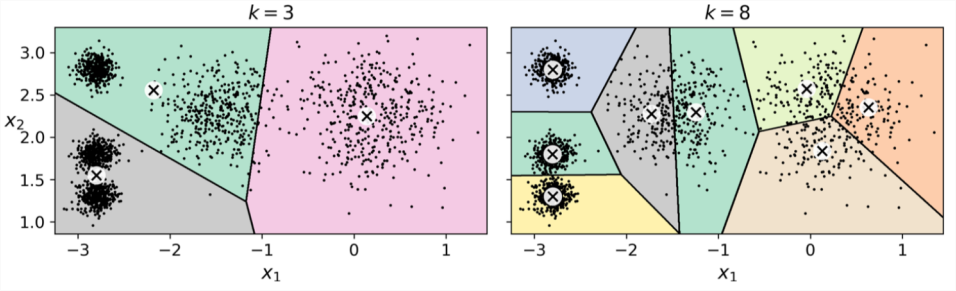
- If you know approximately where the centroids should be, initialize them by hand and set the initialization in the KMeans-Algorithm eg by setting

```
good_init = np.array([[-3, 3], [-3, 2], [-3, 1], [-1, 2], [0, 2]])
kmeans = KMeans(n_clusters=5, init=good_init, n_init=1)
```

- Run the algorithm multiple times with different random initializations and keep the best solution. It's done automatically by Scikit if the number of random initializations is determined by the `n_init` hyperparameter.
- Use the K++ algorithm: has a smarter initialization. (select hyperparameter `init = "k-means++"`)

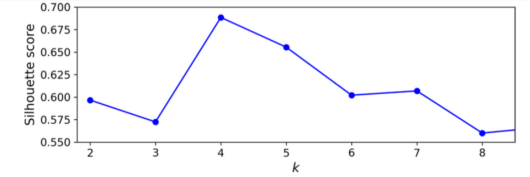
Finding the optimal number of clusters

If you choose the wrong number of clusters, clustering will not work well:



Finding the optimal number of clusters

To find a good number of clusters, you can use the silhouette score: Try out different K, compute the silhouette scores, and plot the silhouette score against the number of clusters K.



You can get the silhouette score with Scikit-learn as follows:

```
from sklearn.metrics import silhouette_score
silhouette_score(X, kmeans.labels_)
```

Gaussian Mixture Models

Overview

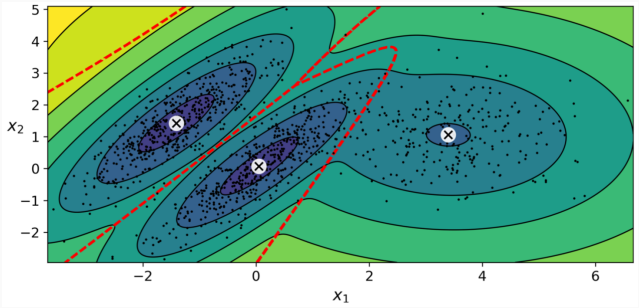
- task: **distribution-based clustering**
- data: no labels, i.e. **unsupervised** learning
- model: **parametric**
- complexity: linear in m=number of instances, but gets slow quickly for high dimensions.

When to use:

- The data are clearly clustered in ellipsoids.
- Only use when there are not many outliers (the algorithms is sensitive to them)!
- If you know approximately how many clusters you have in your data or do not mind trying out several models.

Basic Idea and when to use

Basic Idea: Assume the data come from Gaussian normal distributions. Their means and covariance matrices are the parameters the model can learn from the data.



Gaussian Mixture Model A Gaussian Mixture Model (GMM) is a probabilistic model which is a combination of K (= number of clusters) multivariate Gaussian normal distributions:

$$p(\mathbf{x}|\theta) = \sum_{j=1}^{K} \pi_j N\left(\mu^{(j)}, \mathbf{\Sigma}^{(j)}\right).$$

The probability distribution means the following:

- For each instance, a cluster is picked randomly from among k clusters. The probability of choosing the jth cluster is defined by the cluster's weight π_j. The index of the cluster chosen for the ith instance is noted $z^{(i)}$.
- If $z^{(i)} = j$ (i.e. $\mathbf{x}^{(i)}$ has been assigned to the jth cluster), then $\mathbf{x}^{(i)} \sim N\left(\mu^{(j)}, \mathbf{\Sigma}^{(j)}\right)$

Training a GMM: EM-Algorithm

The EM algorithm is used to find (local) maximum likelihood parameters of a statistical model in cases where the equations cannot be solved directly

This is done using the Expectation Maximization algorithm (EM): initialize the cluster parameters randomly, and repeat the following two steps until convergence:

- expectation step: assign the instances to clusters: the algorithm estimates the probability that it belongs to each cluster (based on current cluster parameters)
- maximization step: update the Gaussian distribution parameters by computing mean and variance using all instances for each cluster.

Then each instance is assigned to

- the closest cluster in case of hard clustering.
- to each cluster with probability corresponding to the responsibility in case of soft clustering.

Complexity

Training a GMM with the EM-Algorithm is $O(mKn^3)$ per iteration, where m is the number of data points, K is the number of Gaussian components and n is the data dimension, i.e. $O(mKn^3t)$ in total, where t = number of iterations.

Inverting the covariance matrix ($n \times n$-matrix) is $O(n^3)$. You have to For each EM-iteration:

- expectation step: for each each instance (times m), for each cluster (times K), compute the probability of the instance belonging to the cluster: this involves inverting the covariance matrix ($n \times n$-matrix) ($O(n^3)$). in total: $O(mKn^3)$.
- maximization step: computing mean and variance for K clusters is non-dominant.

GMM with Scikit Learn

```
from sklearn.mixture import GaussianMixture
gm = GaussianMixture(n_components=3, n_init=10)
gm.fit(X)
```

Like K-Means, the optimization can end up converging to poor local solutions, so it also needs to be run several times. Set `n_init` to eg 10. Check whether or not the algorithm converged and how many iterations it took like

```
>>> gm.converged_
True
>>> gm.n_iter_
3
```

Use the `predict()` method for hard clustering (), and the `predict_proba()` for soft clustering.

Hyperparameters

Hyperparameter: `covariance_type`

- “spherical“ all clusters must be spherical, but can have different diameters.
- “diag“ Clusters can take on any ellipsoidal shape of any size, but the axes must be parallel to the coordinate axes.
- “tied“ All clusters must have the same ellipsoidal shape, size, and orientation.
- default: “full“

DBSCAN

Overview

- task: **density-based clustering**
- data: no labels, i.e. **unsupervised** learning
- model: **non-parametric**
- complexity: quadratic in the number of samples m.

When to use:

- The clusters have non-ellipsoid shapes
- Only use when the density of all clusters is similar.
- not neccessary to know the number of clusters, but you need to experiement with how "greedy" the algorithm is (hyperparameter ϵ)

DBSCAN

How the algorithm works:

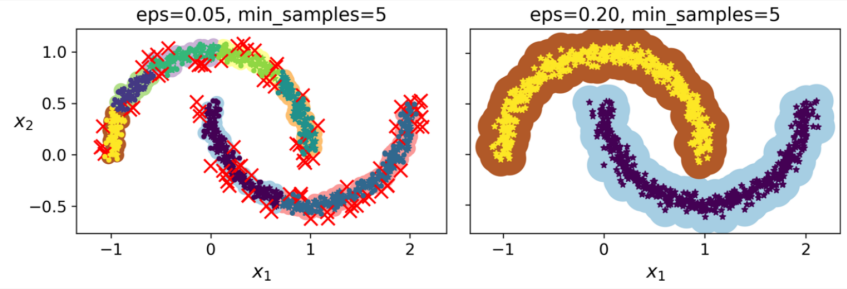
- for each instance, the algorithm counts how many instances are located within a small distance ϵ (the so-called ϵ-neighborhood) from it.
- if an instance has at least `min_samples` instances in its ϵ-neighborhood, it is considered a core instance.
- all instances in the neighborhood of a core instance belong to the same cluster
- any instance that is not a core instance and does not have one in its neighborhood is considered an anomaly.

Training DBSCAN takes $O(m^2)$ at worst.

Without proof.

The hyperparameter ϵ

Advantage: you don't have to specify the number of clusters! It will simply be a consequence of how large you choose ϵ, but this is a hyperparameter you need to optimize:



DBSCAN with Scikit Learn

```
from sklearn.cluster import DBSCAN
from sklearn.datasets import make_moons

X, y = make_moons(n_samples = 1000, noise = 0.05)
dbscan = DBSCAN(eps=0.05, min_samples=5)
dbscan.fit(X)
dbscan.labels_
```

Some instances have label -1, which means they are anomalies. The indices of the core instances are available in the `core_sample_indices_`, the core samples are available in the `components_` instance variable.

Predicting with DBSCAN

DBSCAN does not have a predict method, so it cannot predict which cluster a new instance belongs to. Instead, DBSCAN is just a labeling method so you can use a classification algorithm to predict the cluster of a new instance, e.g. KNN:

```
from sklearn.neighbors import KNeighborsClassifier


knn = KNeighborsClassifier(n_neighbors=50)
knn.fit(dbscan.components_, dbscan.labels_[dbscan.core_sample_indices_])
```

Now we can use `knn.predict` on new instances.

If you want to find anomalies in new instances, this works by introducing a maximum distance of an instance to the nearest cluster center and put the index of instances with bigger distance as -1 (=anomaly).

Chapter 10 Summary

Chapter 10 Summary - Clustering

- Clustering is used for customer segmentation, data preprocessing, dimensionality reduction, cluster affinity, anomaly/outlier detection, label propagation in semi-supervised learning, search engines, segmenting an image
- The center of a cluster is called centroid.
- Hard clustering assings a cluster to each instance, soft clustering assigns a probability vector of belonging to the clusters to each instance.
- Performance measure = silhouette score: it is the average of all silhouette coefficients of the instances, which are defined as

$$\frac{b-a}{\max(a,b)} \in [-1,1], \text{where}$$

a = mean distance to all other instances in the cluster

b = mean distance to all intances in the closest other cluster

You can use it to find the optimal numer of clusters.

Chapter 10 Summary - K-Means

- K-Means Clustering finds K centroids $\mu_1, \ldots, \mu_K$ in space such that the distances between the instances and their closest centroid is minimal. It is the model with
- - prediction function: $\mu(\mathbf{x}) = \text{argmin}_k \|\mathbf{x}^{(i)} - \mu(\mathbf{x}^{(i)})\|_2$.
  - cost function = distortion $\sum_{i=1}^{m} \|\mathbf{x}^{(i)} - \mu(\mathbf{x}^{(i)})\|_2^2$

  K-means algorithm: start with random centroids, label the instances, then update the centroids, label the instances again with the new centroids, etc. Since the solution depends on initialization of the centroids, you have to initialize carefully (e.g. KMeans++) or let the algorithm run several times with different initialization.
- The K-means algorithm is $O(tKmn)$, where $t =$ number of iterations, $K =$ number of clusters, $m =$ number of instances, $n=$ dimension of instances.

Chapter 10 Summary - Gaussian Mixture Model (GMM)

- A Gaussian Mixture Model (GMM) is a probabilistic model that assumes that the instances were generated from a mixture of K multivariate Gaussian distributions whose parameters are unknown. Training the model means determining these parameters. The underlying probabilistic density function of observing instance $\mathbf{x}$ given the parameters θ:

$$p(\mathbf{x}|\theta) = \sum_{j=1}^{k} \pi_j N\left(\mu^{(j)}, \mathbf{\Sigma}^{(j)}\right).$$

- It is trained with the Expectation Maximization algorithm:
  - expectation step: assign the instances to clusters, given the current parameters
  - maximization step: update the Gaussian distribution parameters by computing mean and variance using all instances for each cluster.
- Training a GMM with the EM-Algorithm is $O(mKn^3)$ per iteration, where m is the number of data points, K is the number of Gaussian components and n is the data dimension, i.e. $O(mKn^3t)$ in total, where t = number of iterations.

Chapter 10 Summary: DBSCAN

- the algorithm counts how many instances are located within a small distance ϵ (the so-called ϵ-neighborhood) from it.
- if an instance has at least `min_samples` instances in its ϵ-neighborhood, it is considered a core instance
- all instances in the nbhd of a core belong to the same cluster
- any instance that is not a core instance and does not have one in its neighborhood is considered an anomaly.
- Training DBSCAN takes $O(m^2)$ at worst.