Lecture 7: Clustering

Nils Olovsson
nils.olofsson@igp.uu.se

Introduction to data science and Python programming for medical research
Uppsala University

2025

UPPSALA
UNIVERSITET

Content

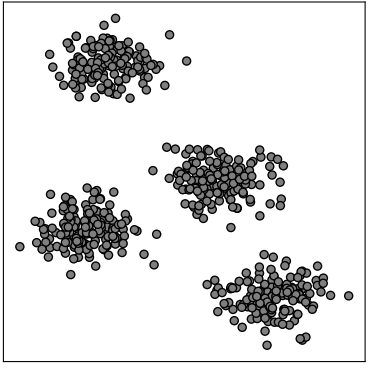
Introduction

Introduction

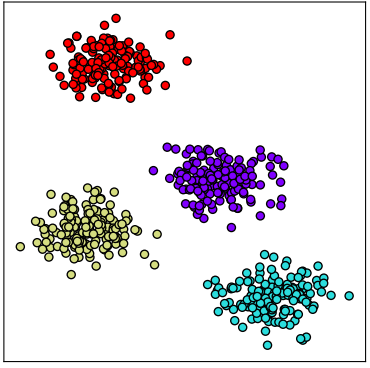
Introduction: Problem

We have some set of unstructured and **unlabeled data**.

We want to **label** this data by **partitioning** it into groups.



Clustering
$\rightarrow$



Introduction: Characteristics

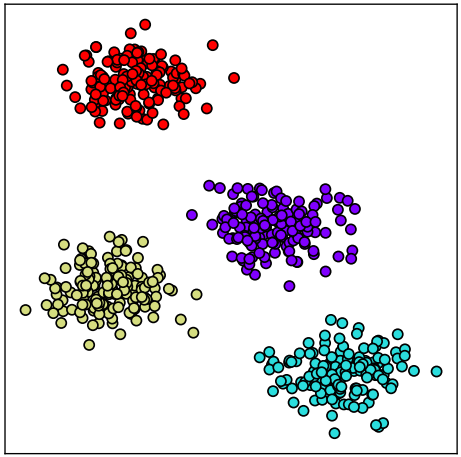
Clustering of data is:

Classification

Assign labels to samples.

Unsupervised

No known target output.



Algorithm

Algorithm: Specification

Let's design an algorithm!

We want a final result such that:

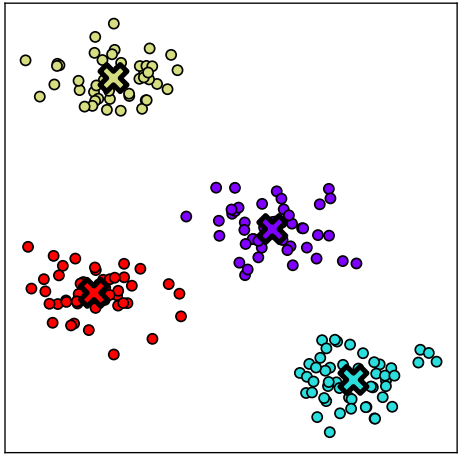
Assignment

Point is closer to its clusters center than to the others clusters centers.

Cluster centers

Center of a cluster is the mean of all points belonging to it.

(This is how we specify it here. This is not generally how all cluster algorithms are designed!)

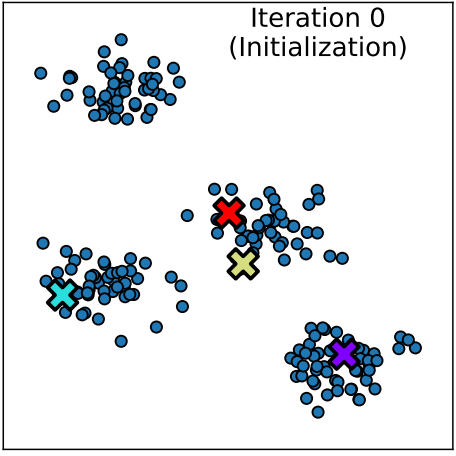


Algorithm: Initialization

Assume that we know the number of clusters, k, in the data.

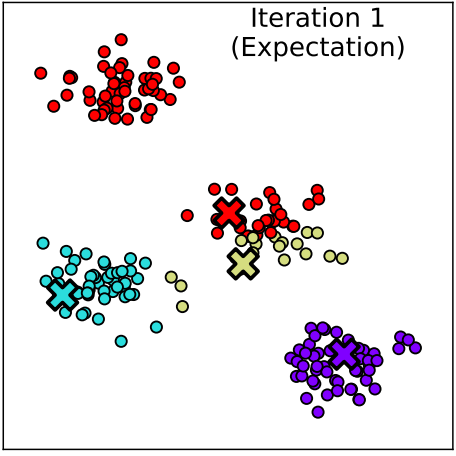
Lets set k=4

Where do we start? Since we do not know any better lets start at the positions of random points in our dataset.



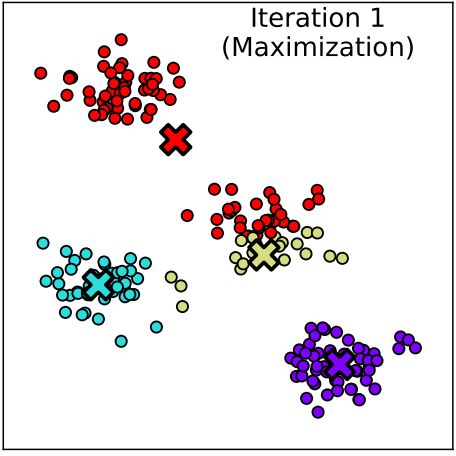
Algorithm: Update our model

Assign each point to the closest cluster center.



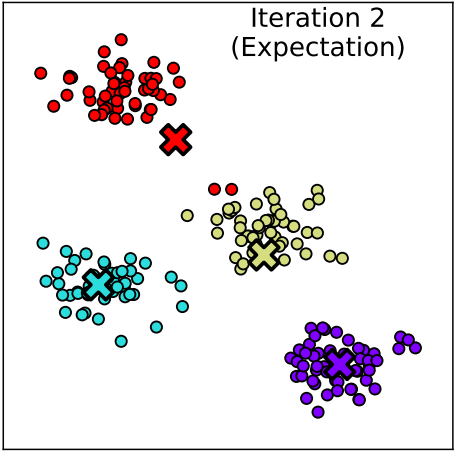
Algorithm: Update our model

Calculate new centers for the clusters as mean of all points belonging to the cluster.



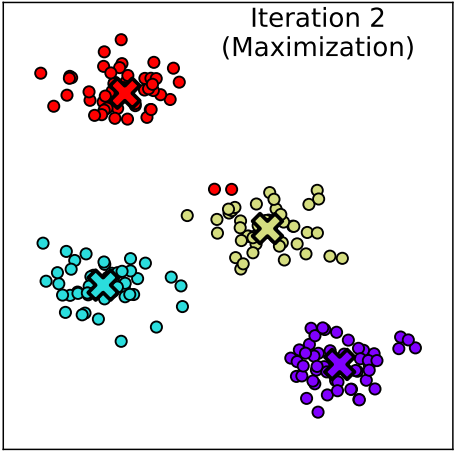
Algorithm: Update our model

Assign each point to the closest cluster center (Again ...).



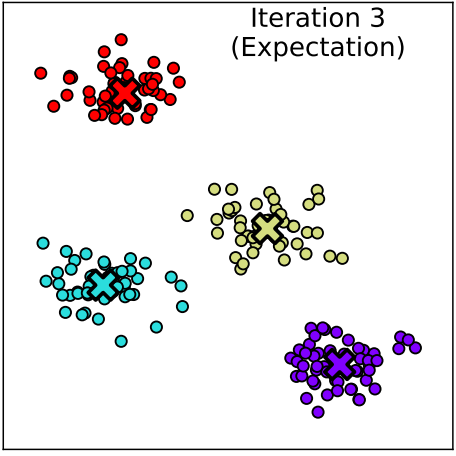
Algorithm: Update our model

Assign each point to the closest cluster center (Again ...).



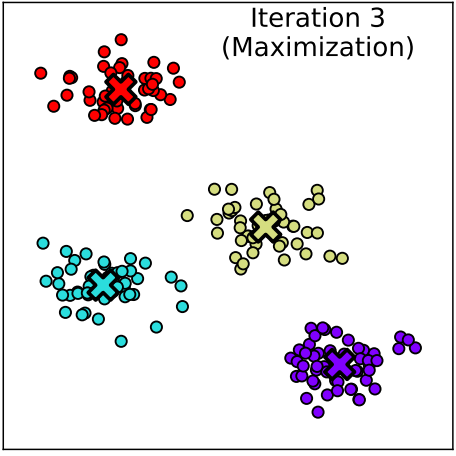
Algorithm: Update our model

Assign each point to the closest cluster center (Again ...).



Algorithm: Update our model

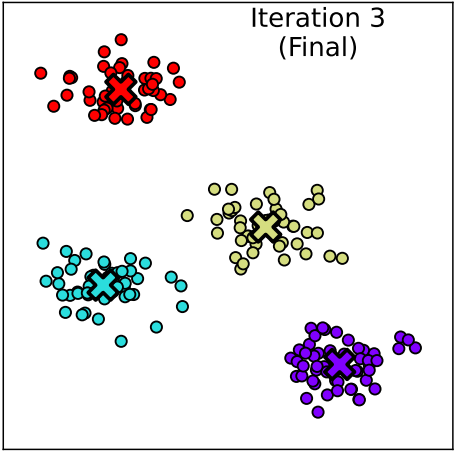
Assign each point to the closest cluster center (Again ...).



Algorithm: Convergence

When we update the centers they just stay the same.

So our method has converged on a solution and we are done.

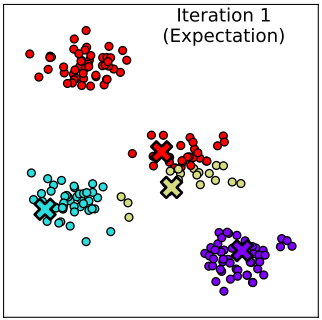


Algorithm: Expectation-Maximization (EM)

The algorithm contains two steps that are repeated.

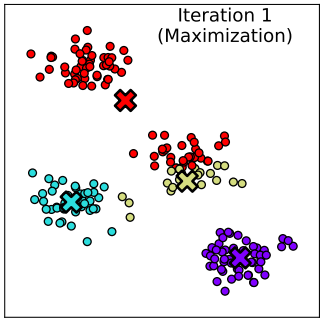
1. Expectation (E)

Assign points to clusters.



2. Maximization (M)

Update the cluster centers.



(Note: Belongs to a family of EM-methods!)

Python: Implementing k-means

This algorithm is called the **k-means** clustering.

Let us implement this in Python.

Need to set **number of clusters** and some **parameters** for when to stop the EM iteration process.

```python
class KMeans:

    def __init__(
        self,
        n_clusters,
        abs_tolerance: float = 1e-3,
        max_iterations: int = 100,
        random_state: int = 0,
    ):
        # Number of clusters assumed used by method.
        # Data can contain more or less clusters!
        self._n_clusters = n_clusters

        # Settings for when to stop the iterations and
        # random seed used in the initialization
        self._abs_tolerance = abs_tolerance
        self._max_iterations = max_iterations
        self._random_state = random_state

        # Result of method is the cluster centers
        self.cluster_centers_ = None
        self.n_iterations_ = -1
        return
```

Python: Expectation-Maximization

When fitting to data, the algorithm will first be **initialized**.
Then at each **iteration** the algorithm will perform two steps.

1. Expectation

Assign points to clusters.

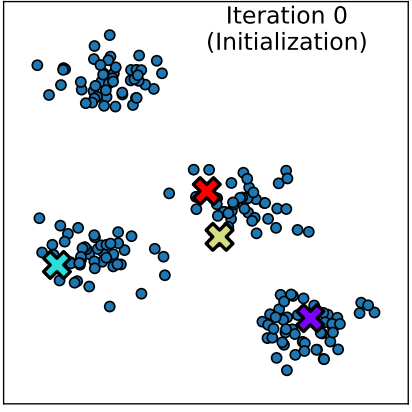
2. Maximization

Update the cluster centers.

```python
def fit(
    self,
    X: np.ndarray,
) -> None:
    """ Fit the cluster centers to data.
    """
    self._initialize(X)
    converged = False
    self.n_iterations_ = 0
    while not converged and self.n_iterations_
      < self._max_iterations:
        self.n_iterations_ += 1
        y = self._expectation(X)
        converged = self._maximization(X, y)
    return
```

Python: Initialization

Start with cluster centers set to random points in the data set.

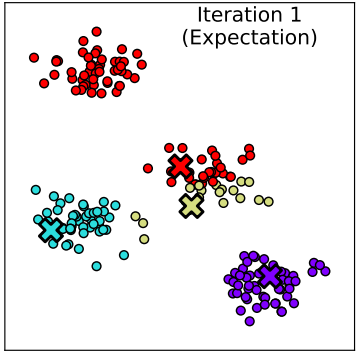


Shuffle a list of indices used to select `_n_clusters` number of rows in the data matrix, `X`.

```
1  def _initialize(
2      self,
3      X: np.ndarray,
4  s -> None:
5      """ Initialize the centers to random
        points in the dataset.
6      """
7      rng = np.random.default_rng(self.
        _random_state)
8
9      # Generate array with indices [0, N-1]
10     indices = np.arange(0, X.shape[0])
11
12     # Shuffle the indices and get the first
        few which are now random
13     rng.shuffle(indices)
14     indices = indices[0:self._n_clusters]
15
16     # Set the cluster centers to random points
        in data set
17     self.cluster_centers_ = X[indices, :]
18
19     return
```

Python: Expectation

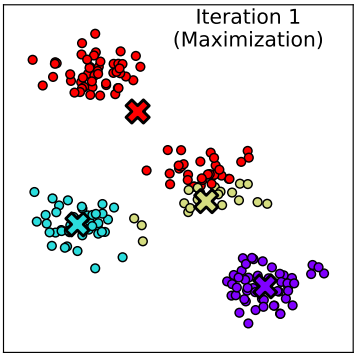
Point belongs to cluster center. Assign points to cluster.




```python
def _expectation(
    self,
    X: np.ndarray,
) -> np.ndarray:
    """ Find the closest cluster center of each data point.
    """

    # Initialize an array of invalid cluster indices
    y = -1 * np.ones(X.shape[0], dtype=int)

    # Check the distance of each data point
    for i in range(X.shape[0]):
        p = X[i,:]
        p_index = -1
        p_distance = 1e8 # Set to large value
        # Check distance of each cluster to each data point
        for cluster_index in range(self._n_clusters):
            c = self.cluster_centers_[cluster_index, :]
            d = np.linalg.norm(c - p)

            # If the current clusters distance is smaller than
            # the previously smallest value we store this index
            # and its distance as the closer cluster.
            if d < p_distance:
                p_distance = d
                p_index = cluster_index
        # Assign the closest cluster index
        y[i] = p_index
    return y
```

Python: Maximization

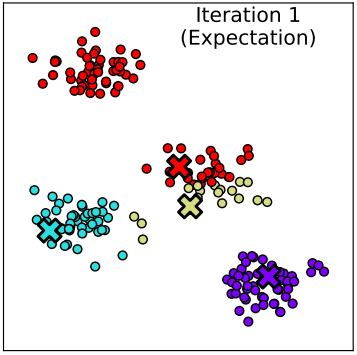
Center of cluster. New center is mean of all points.




```python
def _maximization(
    self,
    X: np.ndarray,
    y: np.ndarray,
) -> bool:
    """ Update the cluster centers.
    Return whether or not the methods has converged.
    """

    # New cluster centers are the men positions of
    # all their associated data points.
    old_clusters = np.copy(self.cluster_centers_)
    for i in range(self._n_clusters):
        X_i = X[y == i, :]
        c = np.mean(X_i, axis=0)
        self.cluster_centers_[i, :] = c

    # Compare the old and the new cluster centers.
    # Use approximate comparisons and check all
    # elements in all center points.
    converged = np.all(np.isclose(old_clusters,
                                  self.cluster_centers_,
                                  atol=self._abs_tolerance))

    return converged
```

Python: Prediction

Don't update our cluster centers.



Only assign label(s) to new data point(s).

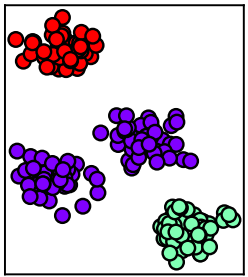
```
def predict(
    self,
    X: np.ndarray,
) -> np.ndarray:
    y = self._expectation(X)
    return y
```

k-Means

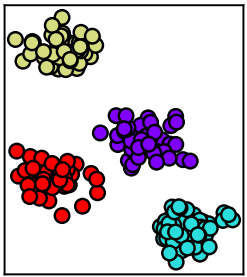
k-Means: "Wrong" value of k

What happens if we have 4 natural clusters in our data but we set k=3?

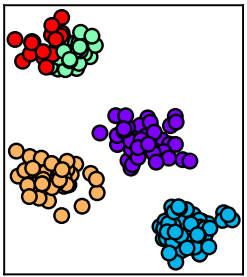
Or we set k=5?



4 natural clusters
k=3



4 natural clusters
k=4



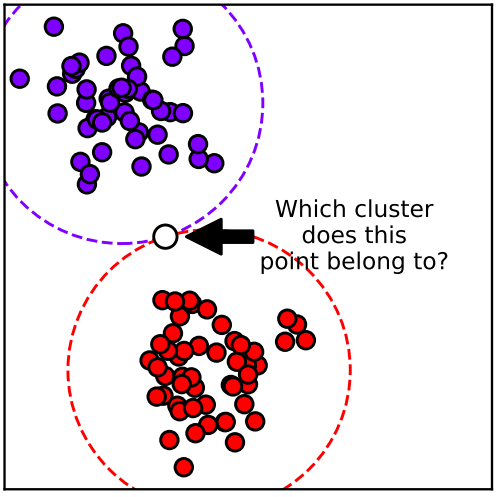
4 natural clusters
k=5

k-Means: Binary classification

Clustering with k-Means is made such that a point either completely belongs to a cluster or it does not.

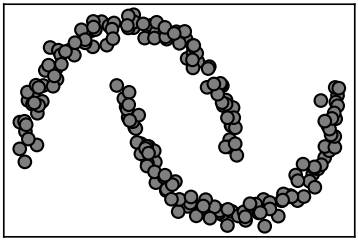
Sometimes we might want to be able to say that it belongs a little to this cluster and somewhat more to another cluster.

Probabilistic interpretation.

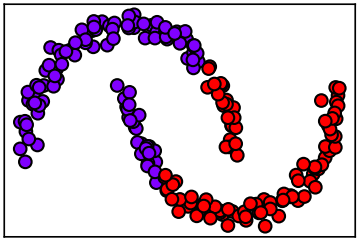


k-Means: Can't cluster non-linear data

The data needs to be linearly separable for the k-Means clustering to work.



Clustering
$\rightarrow$



k-Means: Pros and cons

The k-Means algorithm is a clustering method with a lot of cons but it is very popular because of its simplicity.

Pros

- +Intuitive
- +Easy to implement

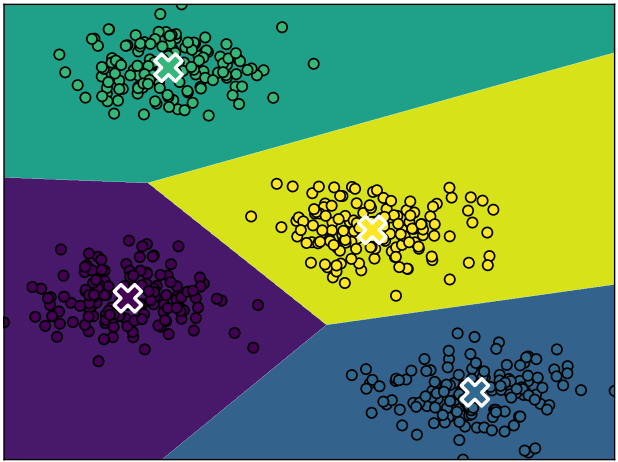
Cons

- -Need to set number of clusters, k!
- -Does not give a probabilistic interpretation
- Can't cluster non-linear data

k-Means: k-Means spatial partition

Divides space into **Voronoi regions**.

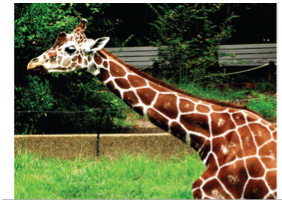
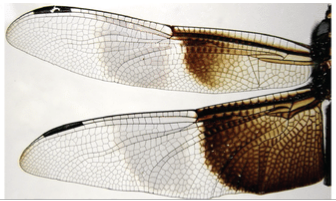
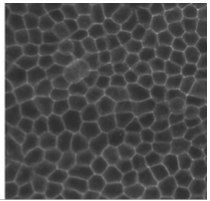
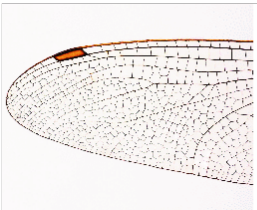
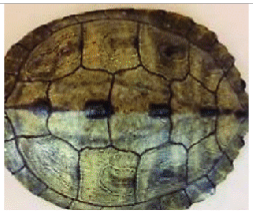
(Not into spherical regions as implied by the book (VanderPlas)!)



k-Means: Voronoi regions

Is important in geometry processing.

Pattern can often be found in nature!


k-Means: Problems

There are some problems with k-Means which make it less suitable to certain types of data distributions.

Will look at some alternative algorithms.

- No probabilistic interpretation.
  (Gaussian mixture models (GMM))
- Can't cluster *non-linear* or *non-convex* data.
  (Spectral clustering)
- Need to know the number of clusters beforehand. Need to set k.
  (Mean shift and DBSCAN)

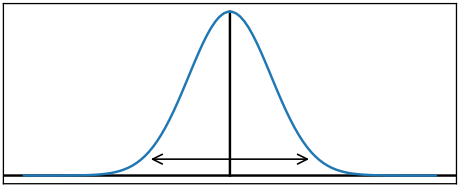
Gaussian mixture models (GMM)

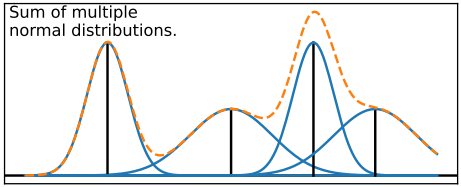
Gaussian mixture models (GMM): Gaussian functions 1D

There are some **drawbacks with k-Means** which make it less suitable to certain types of data distributions.

Instead of distinguishing **clusters** by only their centers we can consider modeling them using **Gaussian** functions.

Gaussian functions are determined by their **centers and variances** (or standard deviations). A **GMM approximates distributions** using **multiple Gaussians**.

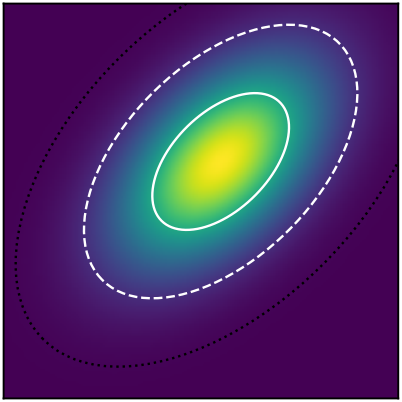


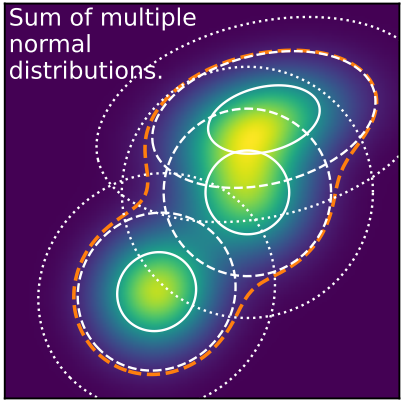


Gaussian mixture models (GMM): Gaussian functions 2D and ND

In higher dimensions **Gaussians** functions are **determined** by their **centers** and their **covariance matrices**.

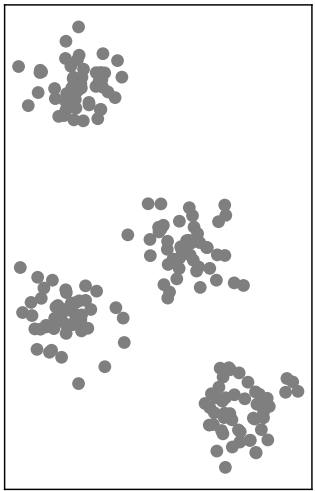
They can similarly be used to **model clusters** and **distributions**.



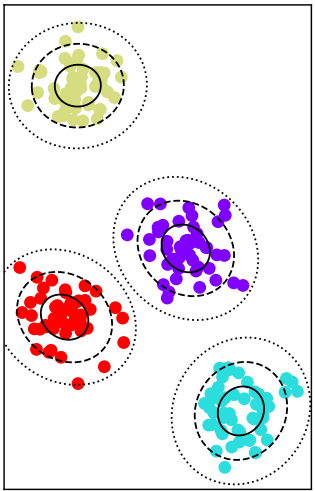


Gaussian mixture models (GMM): Clustering with GMM

Behaves similarly to kMeans.



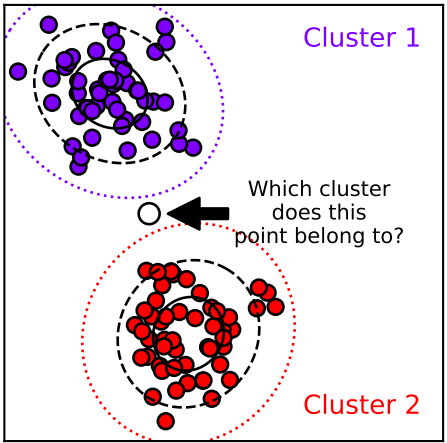
Clustering
$\rightarrow$



Gaussian mixture models (GMM): Probabilistic cluster assignment

```python
# Create a Gaussian mixture model and fit to data
gm = GaussianMixture(n_components=2,
                     random_state=0)
gm.fit(X)
# Create a new point right between the clusters
c = gm.means_
new_point = 0.5 * c[0, :] + 0.5 * c[1, :]
# Predict the probability of point
# beloning to either cluster
p = gm.predict_proba(new_point[np.newaxis, :]).
    flatten()
for i in range(len(p)):
    print(f"Probability of belonging to cluster {(i
     +1)}: {(100.0*p[i]):.1f} %")
```

```
>> Probability of belonging to cluster 1: 54.2 %
>> Probability of belonging to cluster 2: 45.8 %
```



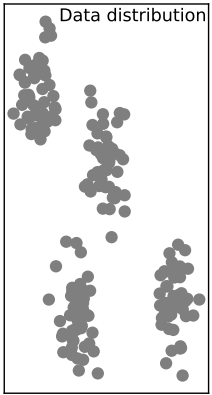
Gaussian mixture models (GMM): Expectation-Maximization

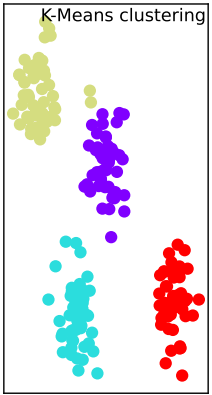
EM is used to fit the GMM.

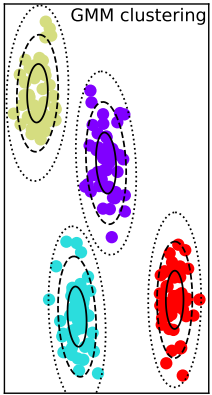
1. Choose a starting guess for the location and shape.
2. Repeat until convergence:
   - *E-step*. For each point, find weights encoding the probability of membership in each cluster.
   - *M-step*. For each cluster, update its location, normalization, and shape based on all data points, making use of the weights.

Gaussian mixture models (GMM): Elongated clusters

GMM can successfully identify elongated clusters while K-Means can not.





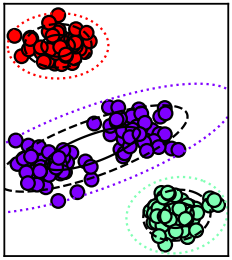


Gaussian mixture models (GMM): Wrong value of k

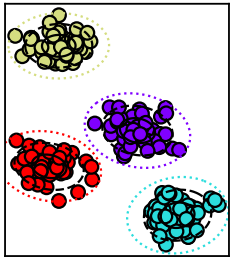
Just like k-Means and GMM the spectral clustering is sensitive to how many clusters we assume to be in the data.

Setting k too low will group clusters that do not belong together.

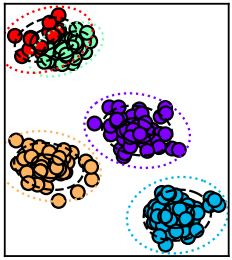
Setting k too high will split clusters of data that do belong together.



4 natural clusters
k=3



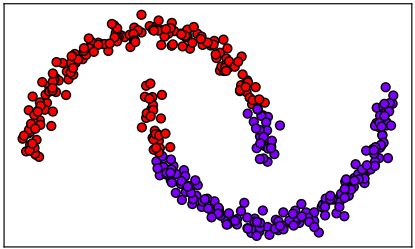
4 natural clusters
k=4

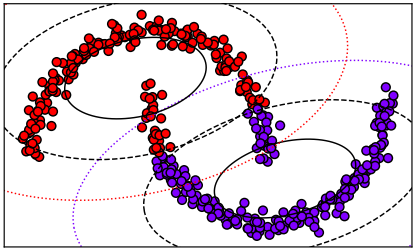


4 natural clusters
k=5

Gaussian mixture models (GMM): Can't cluster non-linear data

GMM is no solution to our problem with non-linear or non-convex data.

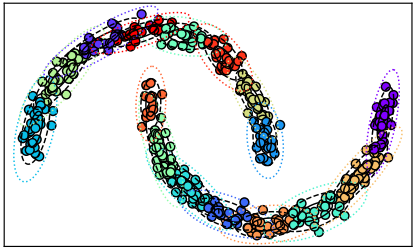


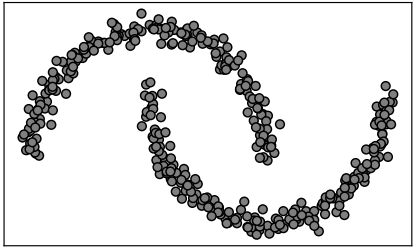


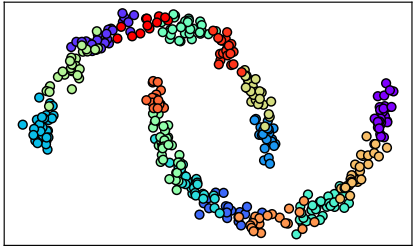
Gaussian mixture models (GMM): Model the data distribution

GMM can **model distributions**. **Sample from** this **distribution** to generate data. A very simple **generative model**!

(This is not the same thing as successfully clustering the data!)







Gaussian mixture models (GMM): Pros and cons

GMM offers a similar method to k-means but with some advantages.

Pros

- +Probabilistic interpretation
- +Can cluster elongated shapes
- +Distribution density estimation

Cons

- -Still need to set number of clusters, k
  (This can be found in an optimal way for density estimation but not for clustering with GMM!))
- -Still can't cluster non-linear or non-convex data.

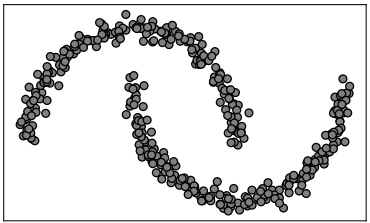
Spectral clustering

Spectral clustering

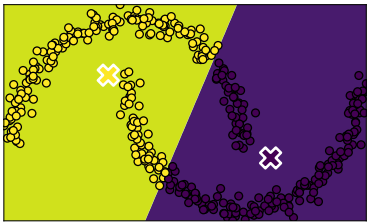
Spectral clustering: Non-convex data

Neither k-Means nor GMM can separate non-linear clusters.

These two clusters can't be separated by by placing a straight line between them.



Two clusters.



K-means clusters with linear separation.

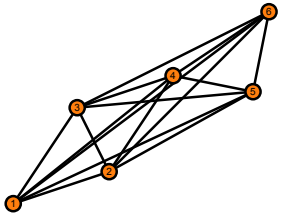
Spectral clustering: Distance matrix

Pairwise distances between points, *e.g.* in matrix X. $M \times M$, where M is samples in the data. Distance matrices are square and, typically, symmetric.

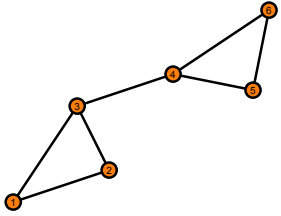
Can connect all points (dense) or subset of points (sparse).

$$D = \begin{pmatrix} 0.0 & 0.6 & 0.7 & 1.3 & 1.7 & 2.0 \\ 0.6 & 0.0 & 0.4 & 0.7 & 1.0 & 1.4 \\ 0.7 & 0.4 & 0.0 & 0.6 & 1.1 & 1.3 \\ 1.3 & 0.7 & 0.6 & 0.0 & 0.5 & 0.7 \\ 1.7 & 1.0 & 1.1 & 0.5 & 0.0 & 0.5 \\ 2.0 & 1.4 & 1.3 & 0.7 & 0.5 & 0.0 \end{pmatrix} \quad D = \begin{pmatrix} 0.0 & 0.6 & 0.7 & 0.0 & 0.0 & 0.0 \\ 0.6 & 0.0 & 0.4 & 0.0 & 0.0 & 0.0 \\ 0.7 & 0.4 & 0.0 & 0.6 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.6 & 0.0 & 0.5 & 0.7 \\ 0.0 & 0.0 & 0.0 & 0.5 & 0.0 & 0.5 \\ 0.0 & 0.0 & 0.0 & 0.7 & 0.5 & 0.0 \end{pmatrix}$$

```
1 from sklearn.datasets import load_diabetes
2 import sklearn.metrics
3 X, y = load_diabetes(return_X_y=True)
4 D = sklearn.metrics.pairwise_distances(X)
```



Fully connected graph.



Connects only neighboring points.

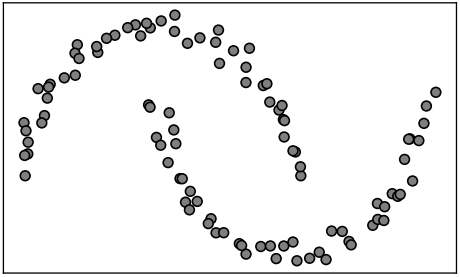
Spectral clustering: Project and cut

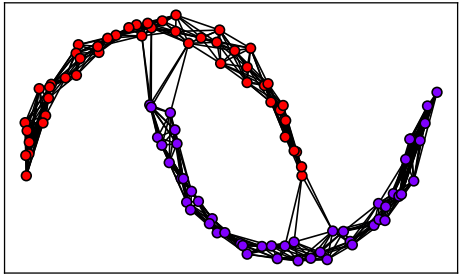
Projects the data such that hopefully the clusters can be separated and assigned a correct label.

For **two clusters**, **cut** along border of **weak connections**.

For **more clusters** the affinity matrix is **Eigen decomposed** and **clusters** are found using *e.g.* K-means of the **eigenvectors**.

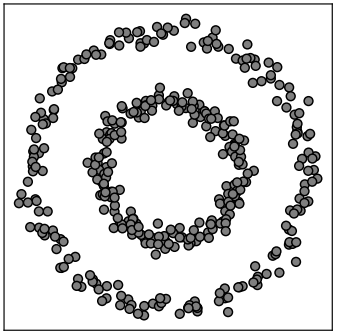
Can now isolate clusters in our non-convex data!

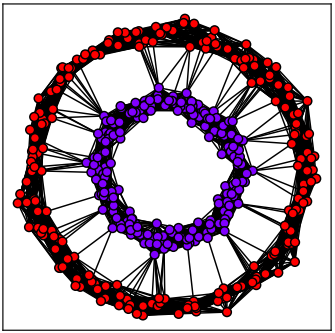


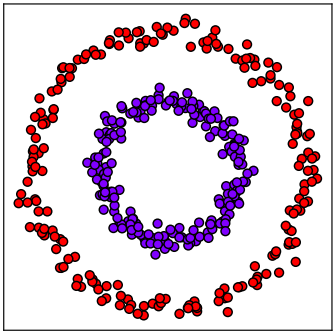


Spectral clustering: Circular data

These two clusters can't be separated by by placing a straight line between them.
Can be solved using spectral clustering.





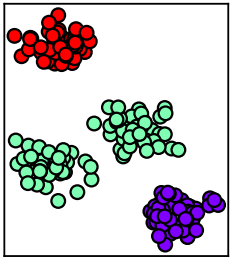


Spectral clustering: Wrong value of k

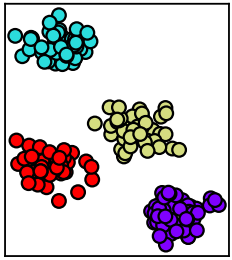
Just like k-Means and GMM the spectral clustering is sensitive to how many clusters we assume to be in the data.

Setting k too low will group clusters that do not belong together.

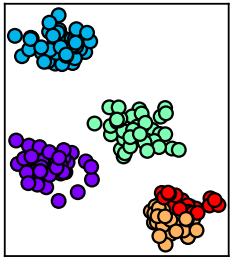
Setting k too high will split clusters of data that do belong together.



4 natural clusters
k=3



4 natural clusters
k=4



4 natural clusters
k=5

Spectral clustering: Pros and cons

Spectral clustering is a completely different approach to the clustering problem compared with k-Means and GMM.

Pros

- +Can cluster non-convex data.
- +No need to initiate to random state ("More" deterministic)

Cons

- -Still need to set number of clusters, k!
- -Can't predict a cluster assignment of a new data point.

Other

Other

Other: Methods without need to set number of clusters

All methods we have talked about require the user to specify the number of clusters that the data will be partitioned into.

There are methods that can find the number of clusters on their own.

Mean shift

Centroid based method which aims at discovering blobs of data.

DBSCAN

Density-Based Spatial Clustering of Applications with Noise.
Finds core samples of high density and expands clusters from them. Good for data which contains clusters of similar density.

https://scikit-learn.org/stable/modules/generated/sklearn.cluster.MeanShift.html
https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html

Other: Many other methods

There are **many other cluster methods** available in sklearn.

They have slightly different parameter settings and behave differently.

E.g. we have seen several examples where the simple K-Means method fails.

When choosing a method, **look into examples and the documentation**.

sklearn.cluster

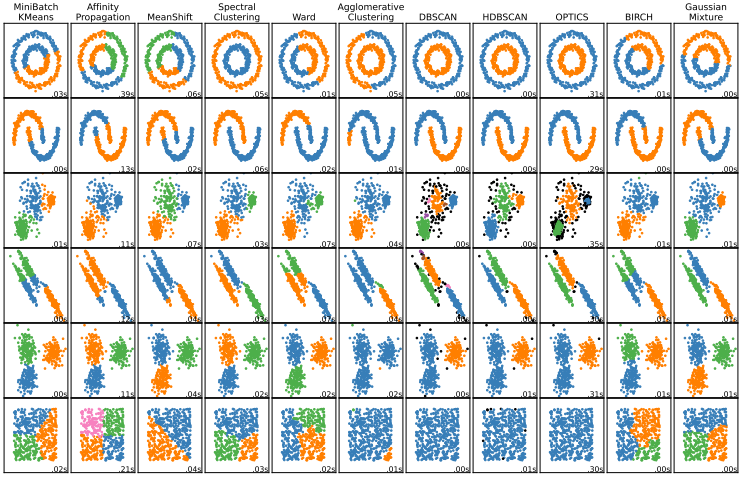
Popular unsupervised clustering algorithms.

User guide. See the Clustering and Biclustering sections for further details.

AffinityPropagation	Perform Affinity Propagation Clustering of data.
AgglomerativeClustering	Agglomerative Clustering.
Birch	Implements the BIRCH clustering algorithm.
BisectingKMeans	Bisecting K-Means clustering.
DBSCAN	Perform DBSCAN clustering from vector array or distance matrix.
FeatureAgglomeration	Agglomerate features.
HDBSCAN	Cluster data using hierarchical density-based clustering.
KMeans	K-Means clustering.
MeanShift	Mean shift clustering using a flat kernel.
MiniBatchKMeans	Mini-Batch K-Means clustering.
OPTICS	Estimate clustering structure from vector array.
SpectralBiclustering	Spectral biclustering (Kluger, 2003).
SpectralClustering	Apply clustering to a projection of the normalized Laplacian.
SpectralCoclustering	Spectral Co-Clustering algorithm (Dhillon, 2001).

https://scikit-learn.org/stable/auto_examples/cluster/plot_cluster_comparison.html

Other: Comparison of methods

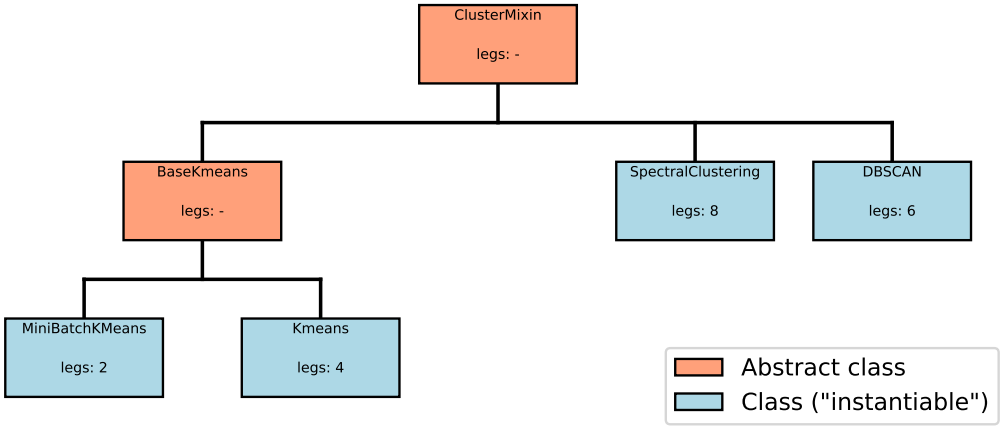


https://scikit-learn.org/stable/auto_examples/cluster/plot_cluster_comparison.html

sklearn API design

sklearn API design

sklearn API design: Inheritance



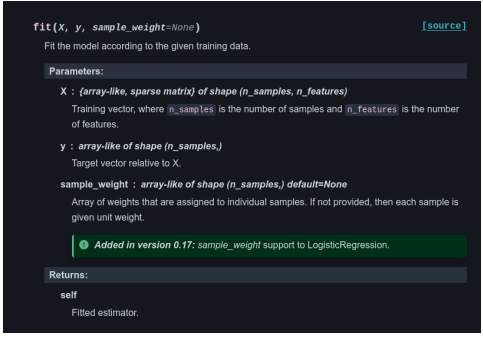
sklearn API design: Common functions

The classes for regression, classification, clustering etc all have a set of common, similar, functions.

- fit(...)
- predict(...)

Depending in type they may also have

- fit_predict(...)
- predict_proba(...)



`fit(X, y, sample_weight=None)` [source]

Fit the model according to the given training data.

Parameters:

X : *{array-like, sparse matrix} of shape (n_samples, n_features)*

Training vector, where `n_samples` is the number of samples and `n_features` is the number of features.

y : *array-like of shape (n_samples,)*

Target vector relative to X.

sample_weight : *array-like of shape (n_samples,) default=None*

Array of weights that are assigned to individual samples. If not provided, then each sample is given unit weight.

Added in version 0.17: *sample_weight* support to LogisticRegression.

Returns:

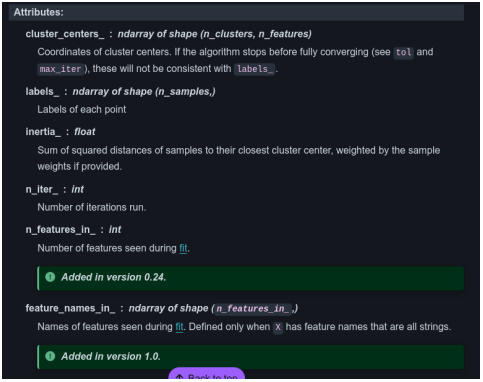
self

Fitted estimator.

sklearn API design: Member variables set after call to `fit(...)`

After the estimator has been fit to data, resulting variables are set.
Their names hare appended with an underscore.

LogisticRegression



Attributes:

cluster_centers_ : *ndarray of shape (n_clusters, n_features)*
Coordinates of cluster centers. If the algorithm stops before fully converging (see `tol` and `max_iter`), these will not be consistent with `labels_`.

labels_ : *ndarray of shape (n_samples,)*
Labels of each point

inertia_ : *float*
Sum of squared distances of samples to their closest cluster center, weighted by the sample weights if provided.

n_iter_ : *int*
Number of iterations run.

n_features_in_ : *int*
Number of features seen during fit.

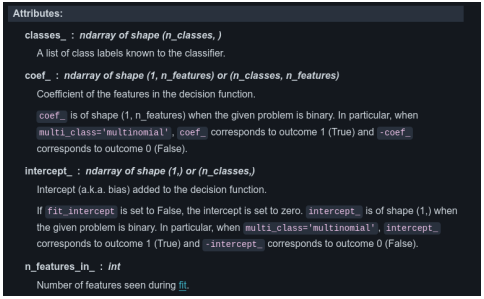
Added in version 0.24.

feature_names_in_ : *ndarray of shape (`n_features_in_`,)*
Names of features seen during fit. Defined only when `X` has feature names that are all strings.

Added in version 1.0.

Back to top

LogisticRegression



Attributes:

classes_ : *ndarray of shape (n_classes,)*
A list of class labels known to the classifier.

coef_ : *ndarray of shape (1, n_features) or (n_classes, n_features)*
Coefficient of the features in the decision function.

`coef_` is of shape (1, n_features) when the given problem is binary. In particular, when `multi_class='multinomial'`, `coef_` corresponds to outcome 1 (True) and `-coef_` corresponds to outcome 0 (False).

intercept_ : *ndarray of shape (1,) or (n_classes,)*
Intercept (a.k.a. bias) added to the decision function.

If `fit_intercept` is set to False, the intercept is set to zero. `intercept_` is of shape (1,) when the given problem is binary. In particular, when `multi_class='multinomial'`, `intercept_` corresponds to outcome 1 (True) and `-intercept_` corresponds to outcome 0 (False).

n_features_in_ : *int*
Number of features seen during fit.

Conclusion

Conclusion

Conclusion:

Introduction
Algorithm
Python
k-Means
Gaussian mixture models (GMM)
Spectral clustering
Other
sklearn API design
Conclusion

Utility of these methods is somewhat limited.

Can be used for data exploration.

We will see methods very similar to spectral clustering when we talk about manifold learning later in the course.