Machine Learning

Data Clustering and K-Means

Dr. Sherif Saad

Learning Objectives

Introduce the students to clustering analysis and its applications

Introduce the K-Means Clustering algorithm and when we should use it

Learn about the strengths and the limitations of K-Means

Learn how to use KMeans from Scikit-Learn

Data Clustering

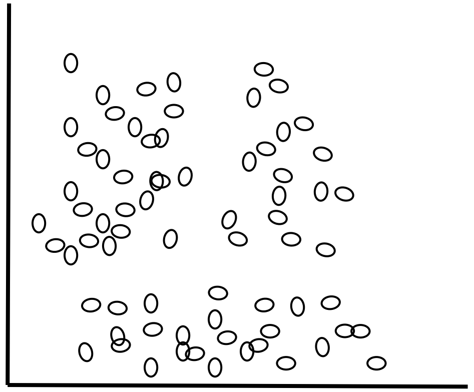
What is Data Clustering?

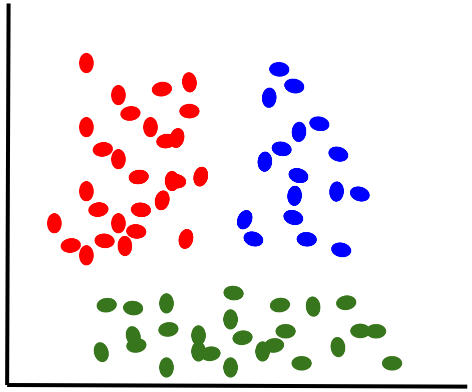
- Data clustering or clustering analysis is unsupervised learning problem that aims at discovering hidden structures in the data where we do not have a target (class label or score)
- The main goal of any clustering algorithm is to find the natural grouping of data items such that items in the same cluster are more similar to each other than those from different clusters
- It is a set of techniques that allow you to group similar instances or samples into one or more group.

Data Clustering

Given an **N** unlabeled samples ***{x1, x2, …., xn}*** and ***K*** the desired number of clusters, group the samples into ***K*** clusters. Where the similarity between any two samples in the same clusters is greater than the similarity between any other two samples from two different cluster.

The main factor that control the clustering is the similarity between the samples and how we define (measure) the similarity





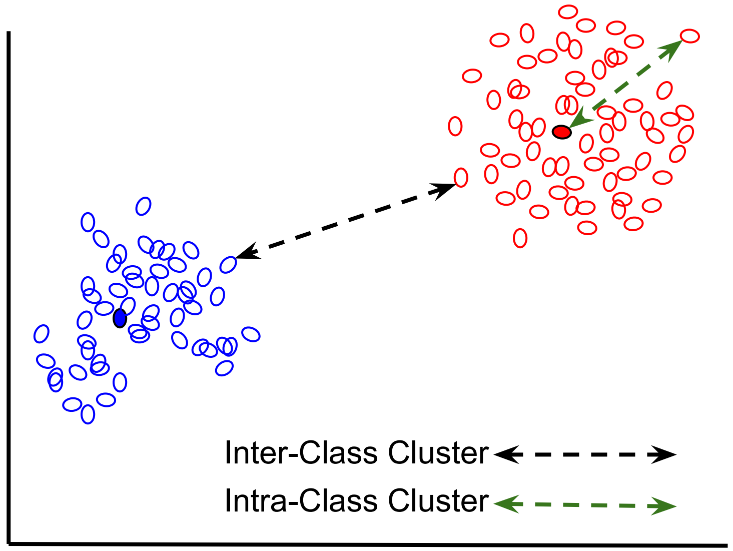
Data Clustering

What is a good clustering algorithm?

- High intra-cluster similarity
- Low inter-cluster similarity

Selecting the similarity measure is critical for the quality of the clustering algorithm.

Different similarity metrics usually lead to different clusterings.



Data Clustering

Examples of Data Clustering Applications:

- Insurance: identify groups of motor insurance policy holders with a high average claim cost
- Credit Card Fraud Detection: identify groups of credit card clients that share similar purchases (money spending) behavior
- Marketing: identify customers with similar and unique preferences and use this information to develop marketing plan

Types Of Data Clustering

In general, there are two main approaches (types) of data clustering techniques:

- Partitioning Algorithms: also known as flat techniques, where the algorithm construct various partitions and then evaluate them by using some criterion (e.g. K-Means, K-Medoids)
- Hierarchical Algorithms: Create a hierarchical decomposition of the set of samples using some criterion (e.g. Bottom-Up and Top-Down). Clusters overlapping exist in hierarchical clustering

Similarity and Distance

Definition: Given a set of objects ***{X1,…,Xn},*** each represented by a m-dimensional vector on m attributes Xi = {xi1, …,xim}, find k clusters classes such that the interclass similarity is minimized and intraclass similarity is maximized.

Distances are usually used to measure the similarity or dissimilarity between two data objects.

While there are many well-defined distance metrics, it is common that each clustering problem will require a domain specific distance or similarity metric.

Similarity and Distance

Minkowski Distance:

$$d(i,j) = \sqrt[q]{(|x_{i_1} - x_{j_1}|^q + |x_{i_2} - x_{j_2}|^q + \ldots + |x_{i_p} - x_{j_p}|^q)}$$

Manhattan Distance:

$$d(i,j) = |x_{i_1} - x_{j_1}| + |x_{i_2} - x_{j_2}| + \ldots + |x_{i_p} - x_{j_p}|$$

Euclidean Distance:

$$d(i,j) = \sqrt{(|x_{i_1} - x_{j_1}|^2 + |x_{i_2} - x_{j_2}|^2 + \ldots + |x_{i_p} - x_{j_p}|^2)}$$

K-Means: Partitional Clustering Method

- The most common clustering algorithm in academia and the industry. It was proposed between 1950s - 1960s by different researchers.
- The K-Means focus on maximizing the intra-cluster similarity.
- The k-Means also known as flat clustering or prototype-based clustering.
- Each cluster is represented by a prototype. This prototype could be the centroid (average) or the medoid (the most frequently occurring point)
- The K-Means works with numerical features.

K-Means Algorithm

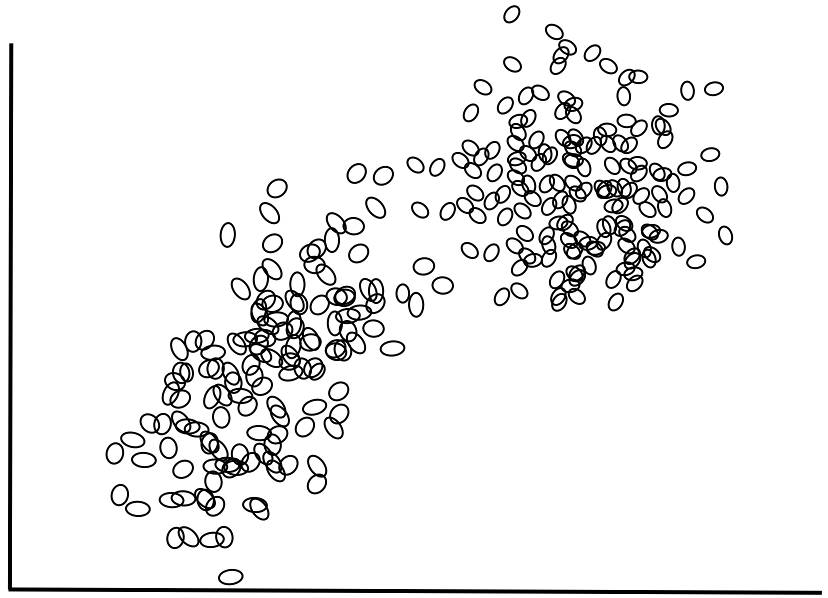
Input: Data Points (observations|samples) and K (desired number of clusters)

1. Randomly pick k centroids as initial cluster centers.
2. Assign each sample to the nearest centroid
3. Recalculate the new centroid of each cluster which will result in moving the centroids to the center of the samples that belongs to this cluster.
4. Repeat steps 2 and 3 until the cluster assignments do not change or for predefined number of iterations or some other termination criteria.

Very simple and easy to implement and debug. In addition relatively efficient with complexity of O(TKN)

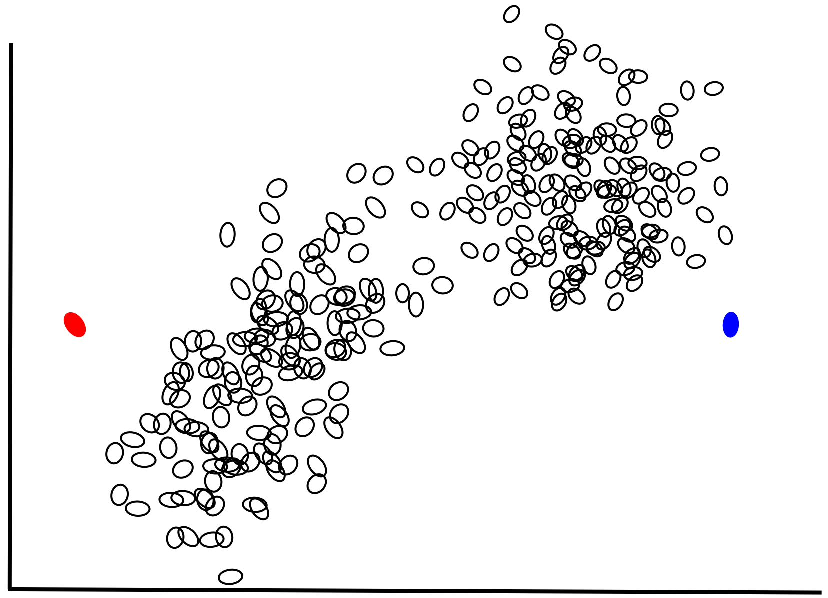
K-Means Algorithm

Given the following data points and K = 2



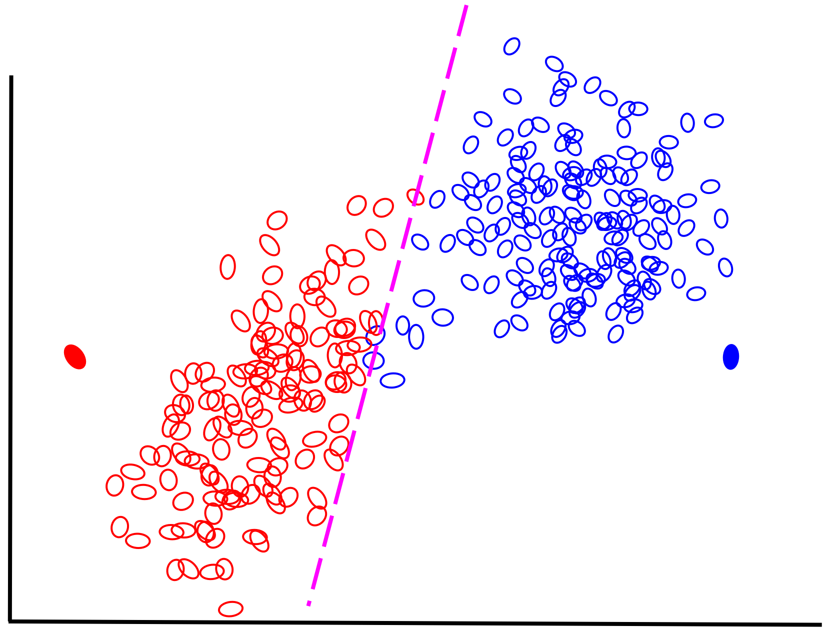
K-Means Algorithm

Initialization set K=2 and randomly select two points



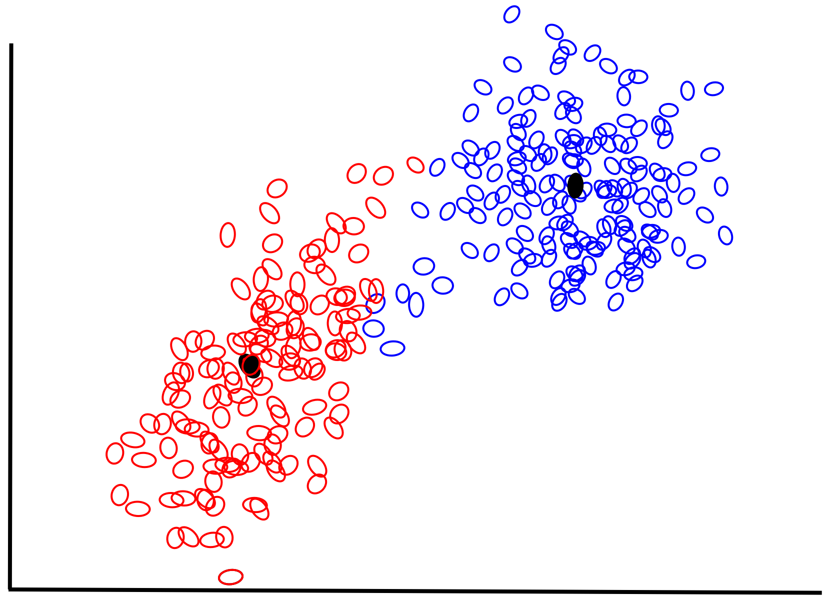
K-Means Algorithm

Assign points based on the selected clusters centers



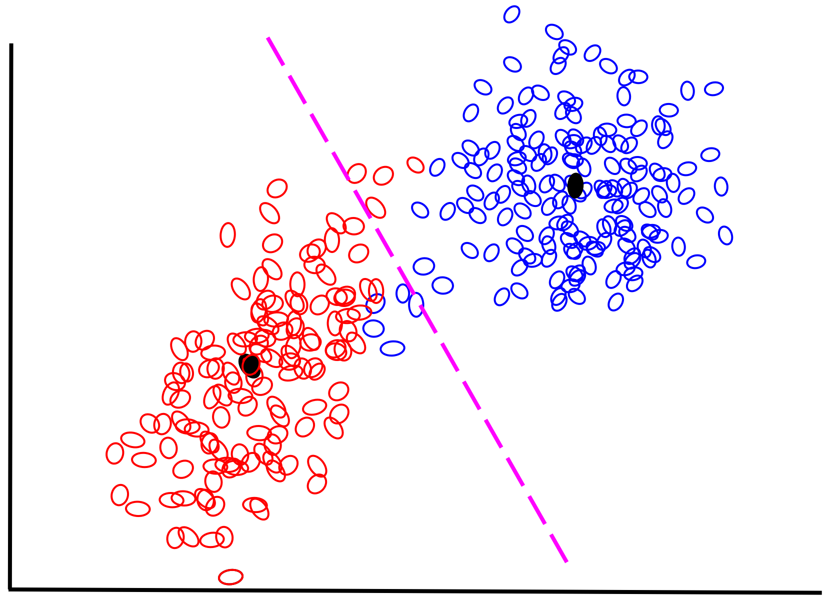
K-Means Algorithm

Iteration 1: Recompute the cluster centers



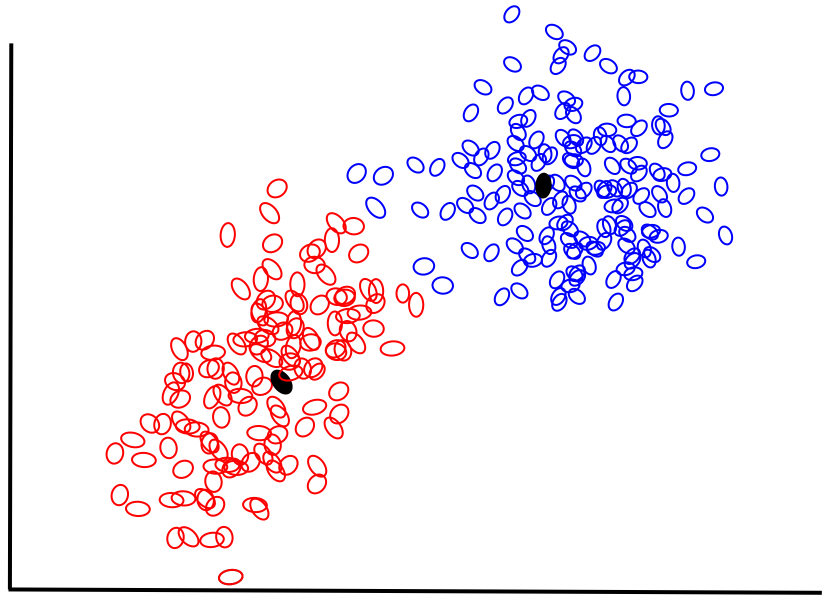
K-Means Algorithm

Iteration 1: Reassign points based on the new clusters centers



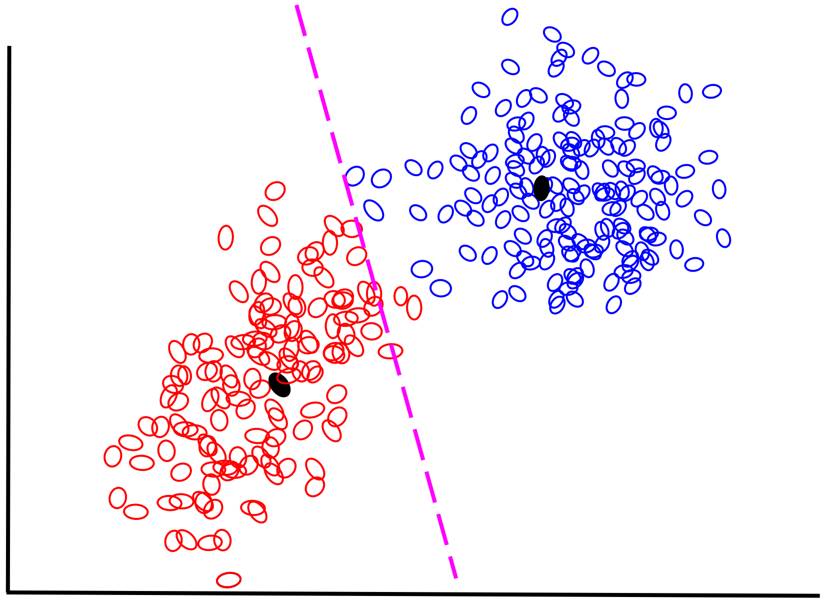
K-Means Algorithm

Iteration 2: Recompute the cluster centers



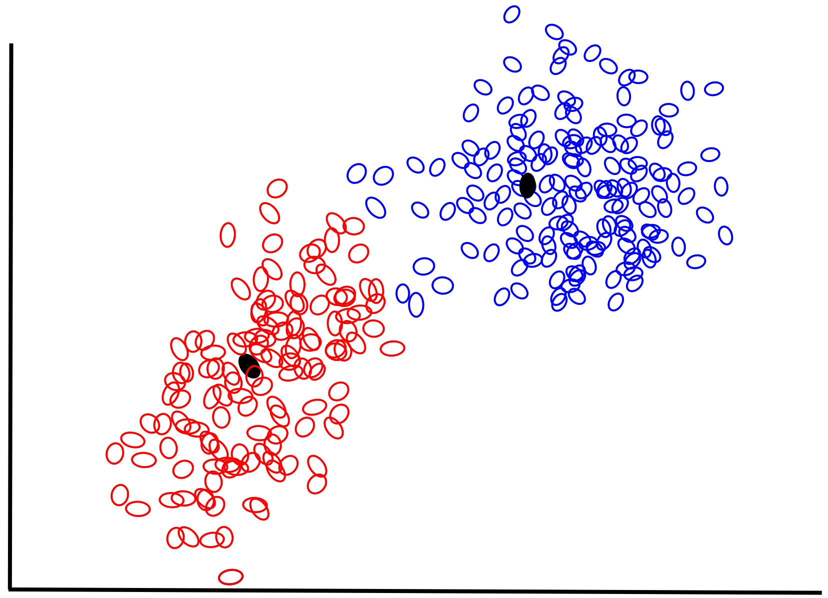
K-Means Algorithm

Iteration 2: Reassign points based on the new clusters centers



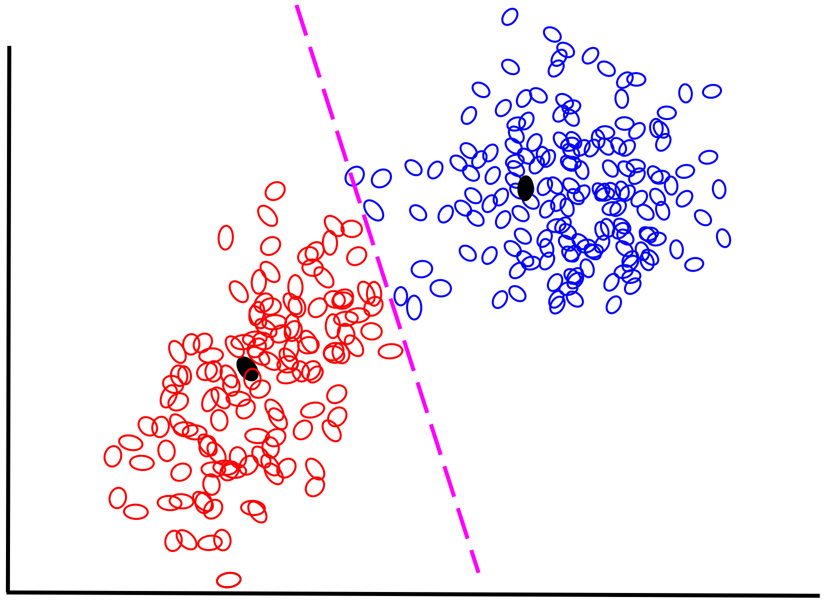
K-Means Algorithm

Iteration 3: Recompute the cluster centers



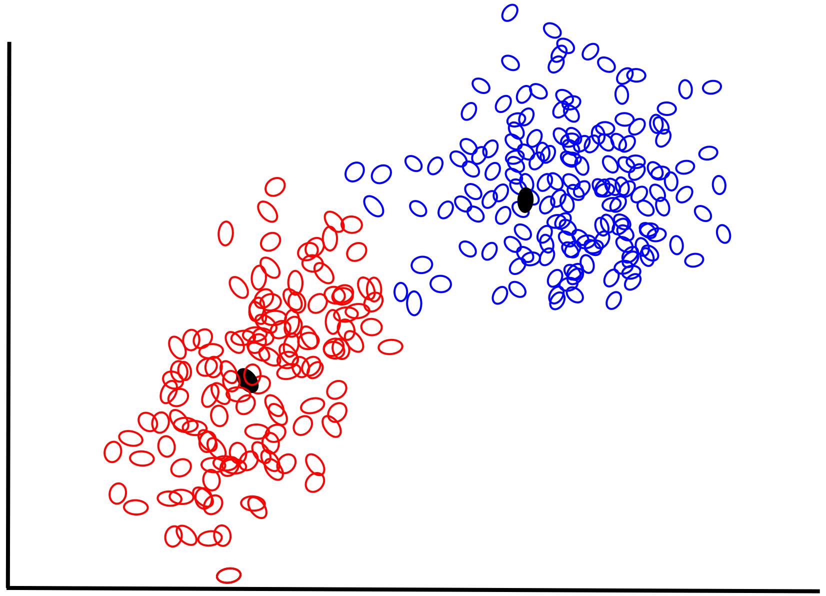
K-Means Algorithm

Iteration 3: Reassign points based on the new clusters centers



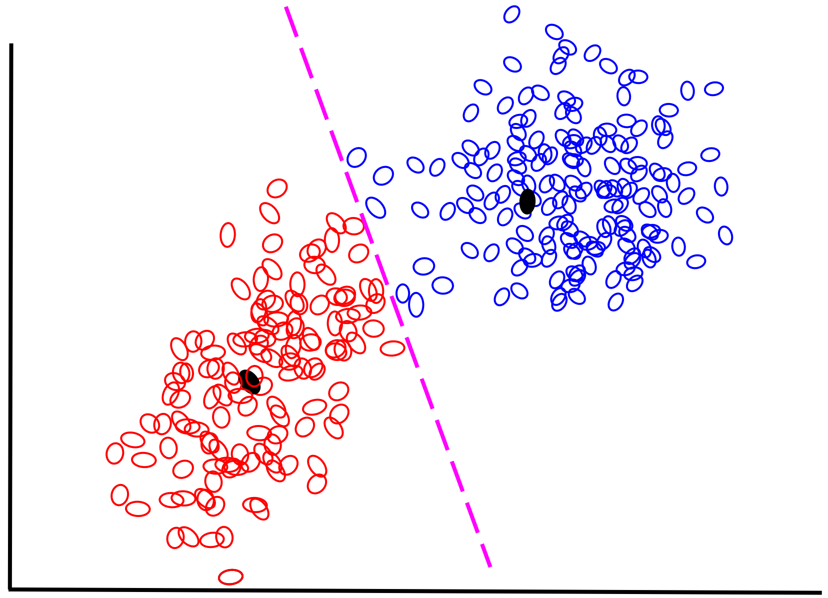
K-Means Algorithm

Iteration 4: Recompute the cluster centers



K-Means Algorithm

Iteration 4: Reassign points based on the new clusters centers



K-Means: Similarity Function

The default similarity metric used by the K-Means algorithm is the Squared Euclidean distance

$$d(\boldsymbol{x},\boldsymbol{y})^2 = \sum_{j=1}^{m}\left(x_j - y_j\right)^2 = \left\|\boldsymbol{x}-\boldsymbol{y}\right\|_2^2$$

We can model the K-Means as an optimization problem that aims at minimizing the within cluster sum of squared errors

$$SSE = \sum_{i=1}^{n}\sum_{j=1}^{k} w^{(i,j)} \left\|\boldsymbol{x}^{(i)} - \boldsymbol{\mu}^{(j)}\right\|_2^2$$

*Where **m**^{j}* is the centroid for cluster ***j*** and ***w***^{i,j} = 1 if the sample **x**^{i} is in cluster ***j*** and 0 if not.

K-Means; When Should you use it

You have unlabeled data items that you want to group based on some similarity measurements.

When you have a good idea of the number of distinct clusters your unlabeled dataset should be segmented into.

If you want to use the calculated centroids to summarize your data or to predict the cluster of new data items

Finally, to label partially-labeled dataset for supervised learning.

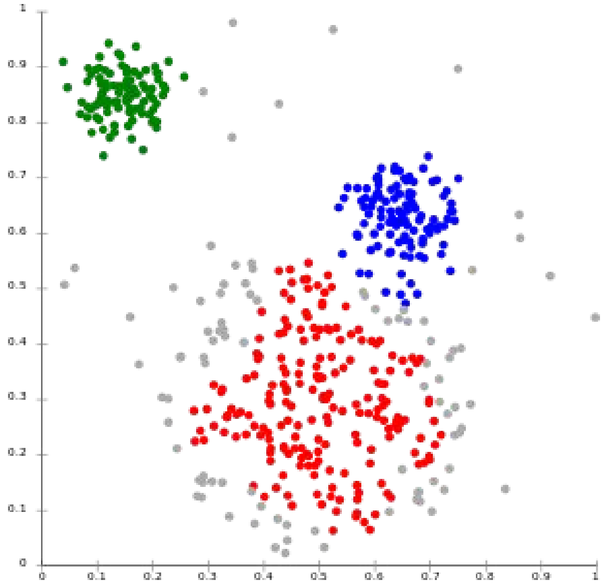
K-Means: Limitations

- The initial selection of the number of clusters (K) and the centers of each cluster has great effect on the quality of the generated cluster.
- Finding the optimal number of clusters (K) is an NP-hard problem.
- The K-Means algorithm uses a heuristic approach that converges to a local optimum.
- The K-Means use a hard assignment clustering method. For a given cluster each sample either belongs to that cluster or not belongs at all.

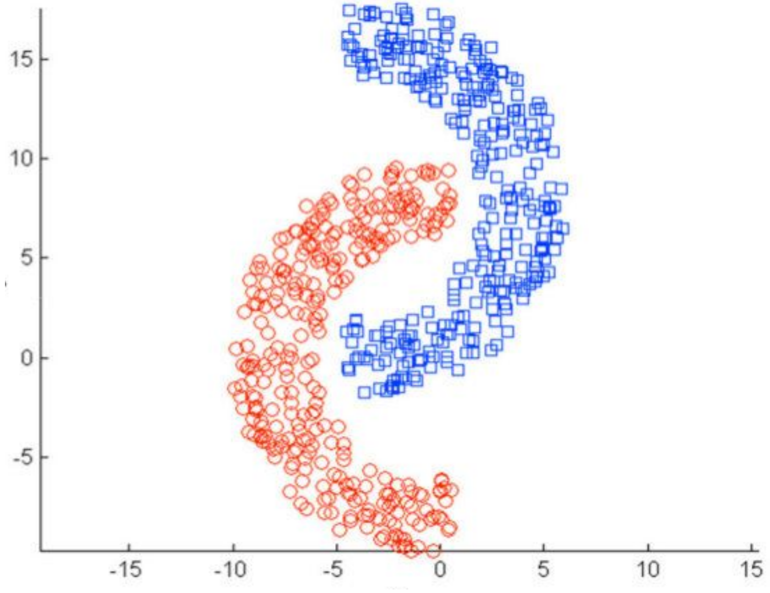
K-Means: Limitations

- K-Means does not support soft clustering where the same instance or sample can belong to more than one cluster with different membership strength (probability or score). One example for such techniques are Fuzzy-C Means
- Sensitive to outliers, they can easily affect the cluster center that is based on the mean. Therefore we can use K-Medians instead of the mean
- Perform badly with with non round shaped or roughly equal sizes/density clusters.

K-Means: Limitations



K-Means will perform poorly on when dealing with different density clusters



K-Means will perform poorly on Non-convex/non-round shape cluster

K-Means: Initialization

The selection of the number of clusters and their centers affects the quality of the generated clusters.

A poorly initialized K-Means will result in a poor intra and inter clustering and slow convergence.

It is recommend to try multiple initializations and initialize the centers using real samples that as far as possible from each others.

Use more advanced techniques to selected the number of clusters, such as the K-Means++ algorithm

K-Means++

Proposed by D. Arthur and S. Vassilvitskii in 2007 "*k-means++: The Advantages of Careful Seeding*"

While the traditional K-Means use random seed to place the initial centroids, the K-Means++ place the initial centroids far away from each other.

The K-Means++ leads to better and more consistent results than the classic k-means.

The K-Means++ does not amis at finding the optimal number of K clusters

K-Means++ Algorithm

Input: Data Points (observations|samples) and K (desired number of clusters)

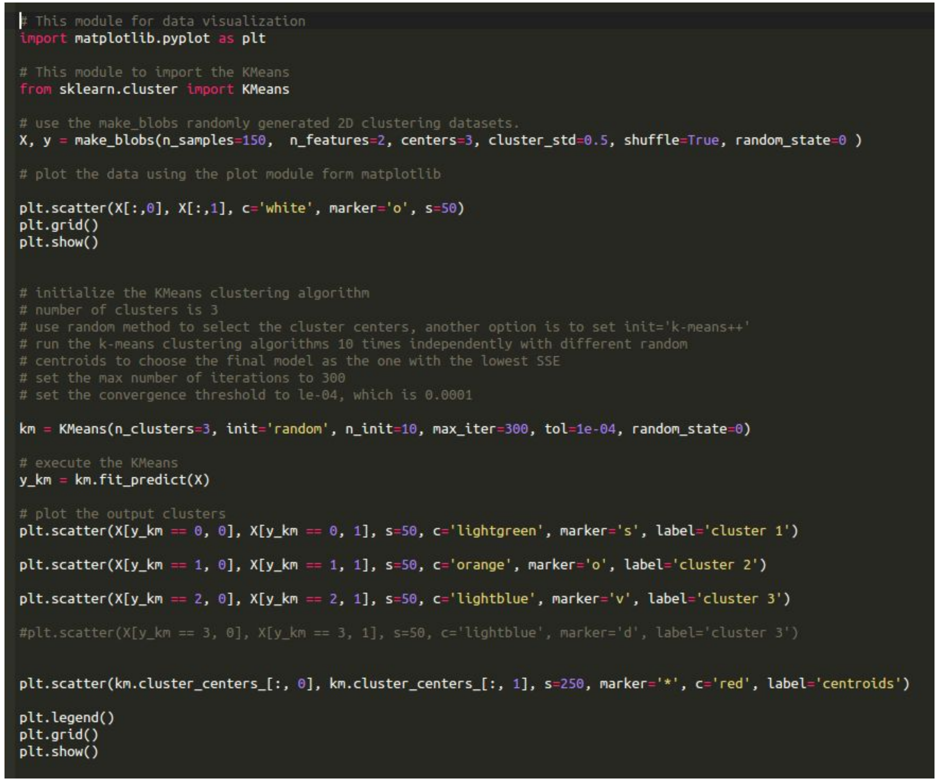
1. Initialize an empty set ***M*** to store the k centroids being selected
2. Randomly choose the first centroid $\boldsymbol{\mu_j}$ from the input samples and assign it to ***M*** .
3. For each sample $\boldsymbol{x_i}$ that is not in ***M*** , find the minimum squared distance $\boldsymbol{d(x_i, M)}$ to any of the centroids in ***M*** .
4. To randomly select the next centroid $\boldsymbol{\mu_p}$, we use a weighted probability distribution.
$$new(\mu_p) = \frac{d(\mu_p, M)^2}{\sum_i^n d(x_i, M)^2}$$
5. Repeat steps 2 and 3 until k centroids are chosen.
6. Proceed with the classic k-means algorithm.

K-Means with Scikit-Learn

K-Means implementation in Scikit-learn only work with numerical data.

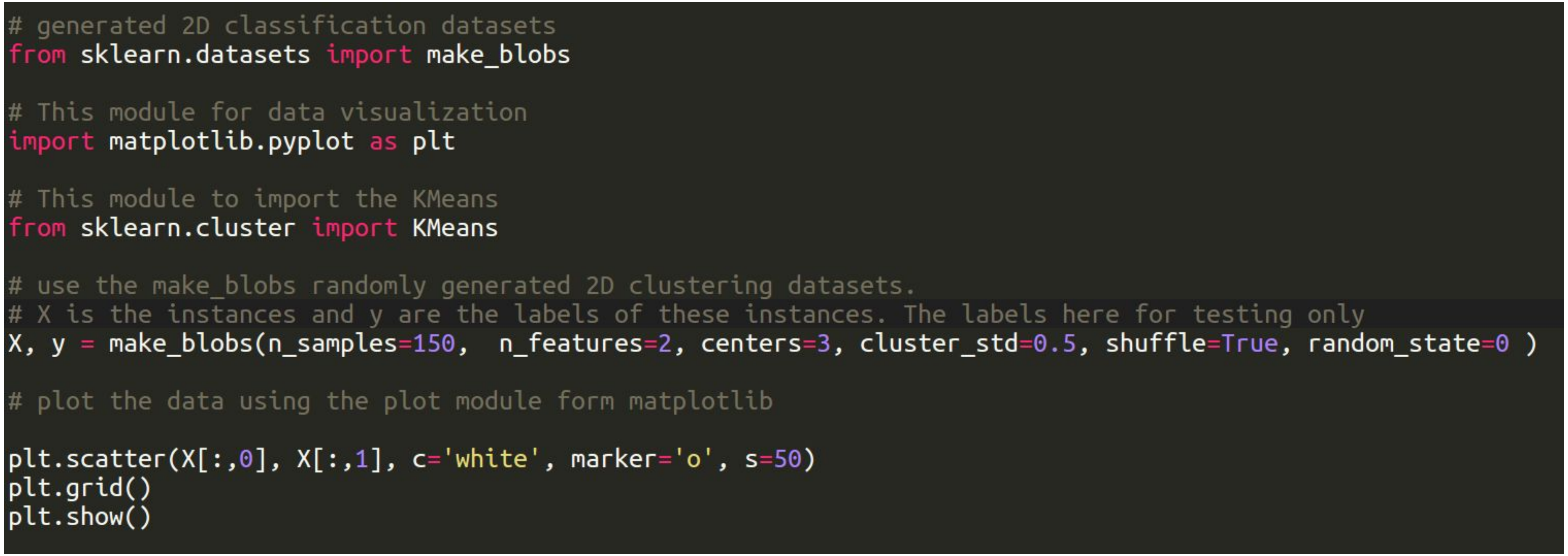
K-Means implementation in Scikit-learn use by default the K-Means++ to initialize the centers of the clusters.

K-Means with Scikit-Learn



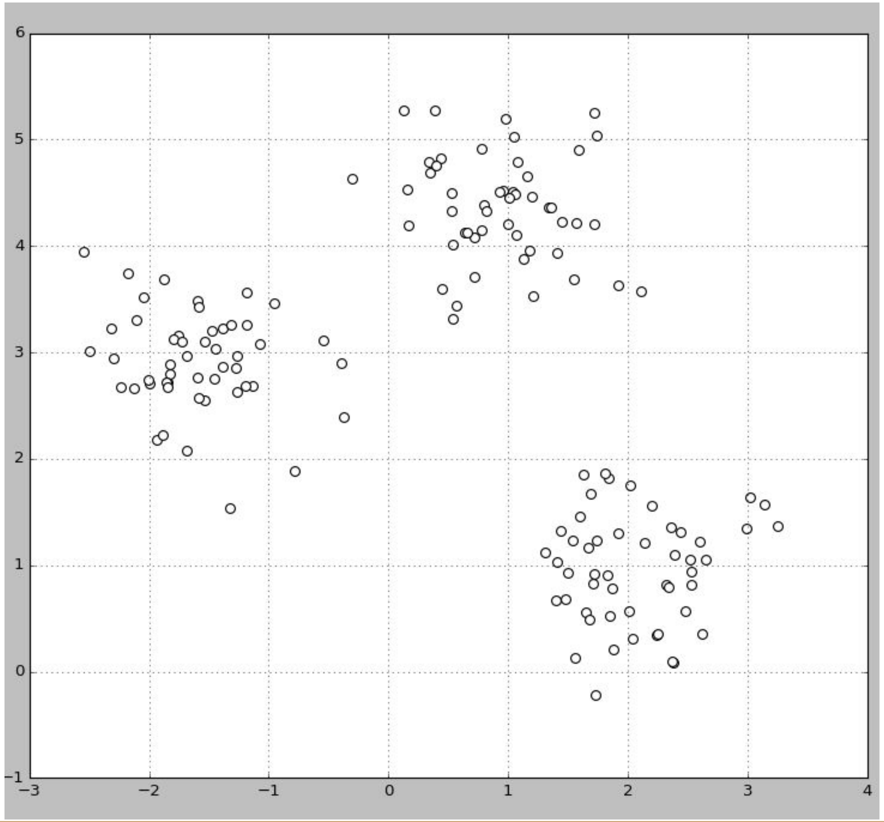
```python
# This module for data visualization
import matplotlib.pyplot as plt

# This module to import the KMeans
from sklearn.cluster import KMeans

# use the make_blobs randomly generated 2D clustering datasets.
X, y = make_blobs(n_samples=150,  n_features=2, centers=3, cluster_std=0.5, shuffle=True, random_state=0 )

# plot the data using the plot module form matplotlib

plt.scatter(X[:,0], X[:,1], c='white', marker='o', s=50)
plt.grid()
plt.show()


# initialize the KMeans clustering algorithm
# number of clusters is 3
# use random method to select the cluster centers, another option is to set init='k-means++'
# run the k-means clustering algorithms 10 times independently with different random
# centroids to choose the final model as the one with the lowest SSE
# set the max number of iterations to 300
# set the convergence threshold to le-04, which is 0.0001

km = KMeans(n_clusters=3, init='random', n_init=10, max_iter=300, tol=1e-04, random_state=0)

# execute the KMeans
y_km = km.fit_predict(X)

# plot the output clusters
plt.scatter(X[y_km == 0, 0], X[y_km == 0, 1], s=50, c='lightgreen', marker='s', label='cluster 1')

plt.scatter(X[y_km == 1, 0], X[y_km == 1, 1], s=50, c='orange', marker='o', label='cluster 2')

plt.scatter(X[y_km == 2, 0], X[y_km == 2, 1], s=50, c='lightblue', marker='v', label='cluster 3')

#plt.scatter(X[y_km == 3, 0], X[y_km == 3, 1], s=50, c='lightblue', marker='d', label='cluster 3')


plt.scatter(km.cluster_centers_[:, 0], km.cluster_centers_[:, 1], s=250, marker='*', c='red', label='centroids')

plt.legend()
plt.grid()
plt.show()
```

K-Means with Scikit-Learn

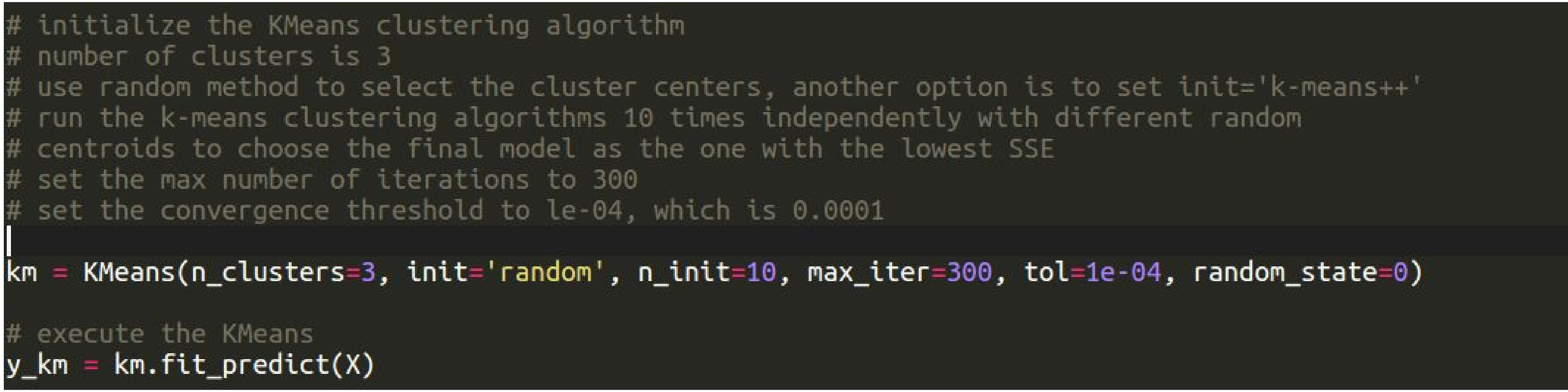


```python
# generated 2D classification datasets
from sklearn.datasets import make_blobs

# This module for data visualization
import matplotlib.pyplot as plt

# This module to import the KMeans
from sklearn.cluster import KMeans

# use the make_blobs randomly generated 2D clustering datasets.
# X is the instances and y are the labels of these instances. The labels here for testing only
X, y = make_blobs(n_samples=150,  n_features=2, centers=3, cluster_std=0.5, shuffle=True, random_state=0 )

# plot the data using the plot module form matplotlib

plt.scatter(X[:,0], X[:,1], c='white', marker='o', s=50)
plt.grid()
plt.show()
```

K-Means with Scikit-Learn

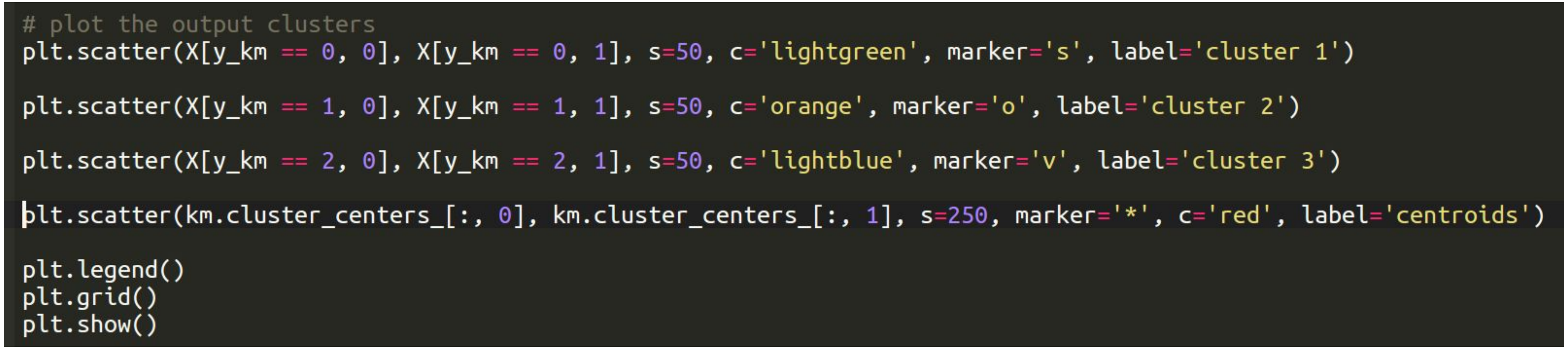


K-Means with Scikit-Learn



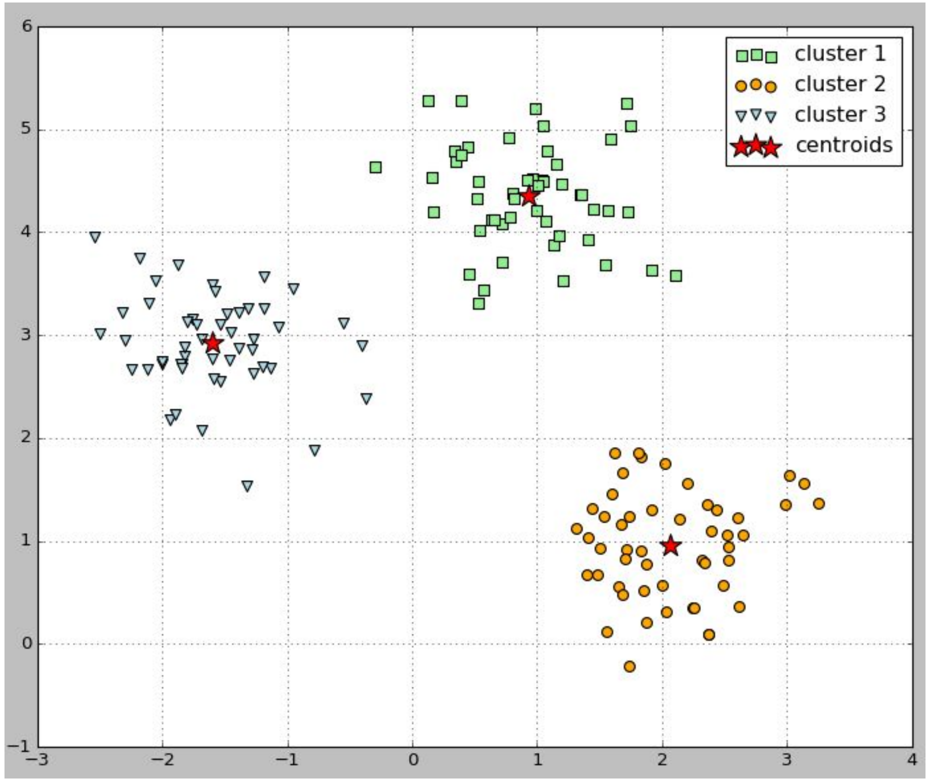
```
# initialize the KMeans clustering algorithm
# number of clusters is 3
# use random method to select the cluster centers, another option is to set init='k-means++'
# run the k-means clustering algorithms 10 times independently with different random
# centroids to choose the final model as the one with the lowest SSE
# set the max number of iterations to 300
# set the convergence threshold to le-04, which is 0.0001

km = KMeans(n_clusters=3, init='random', n_init=10, max_iter=300, tol=1e-04, random_state=0)

# execute the KMeans
y_km = km.fit_predict(X)
```

K-Means with Scikit-Learn

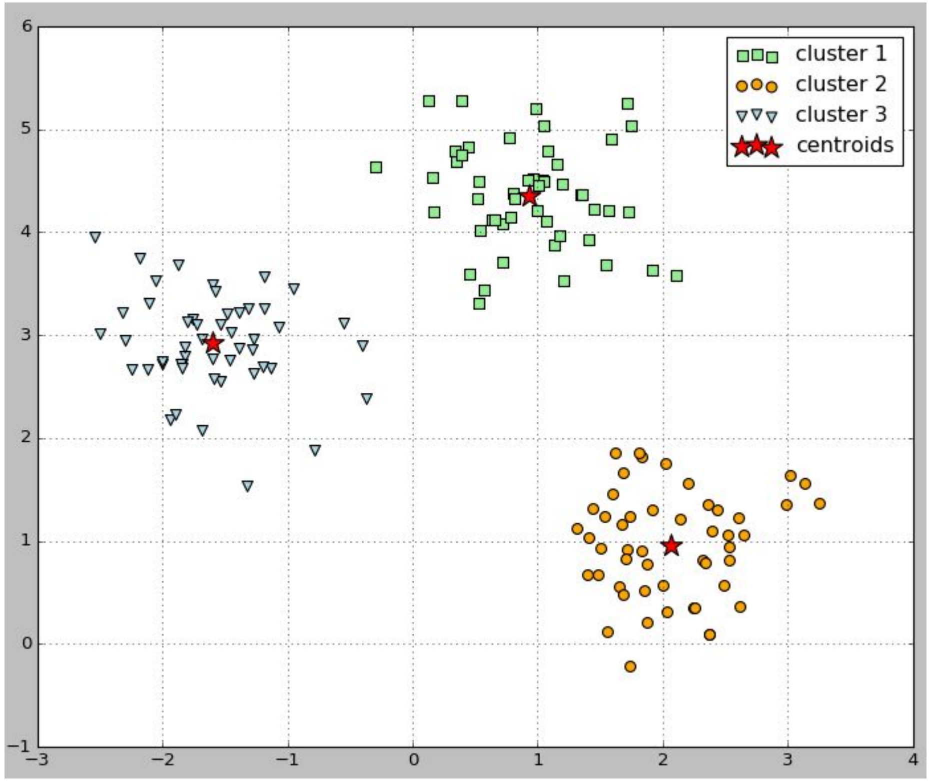


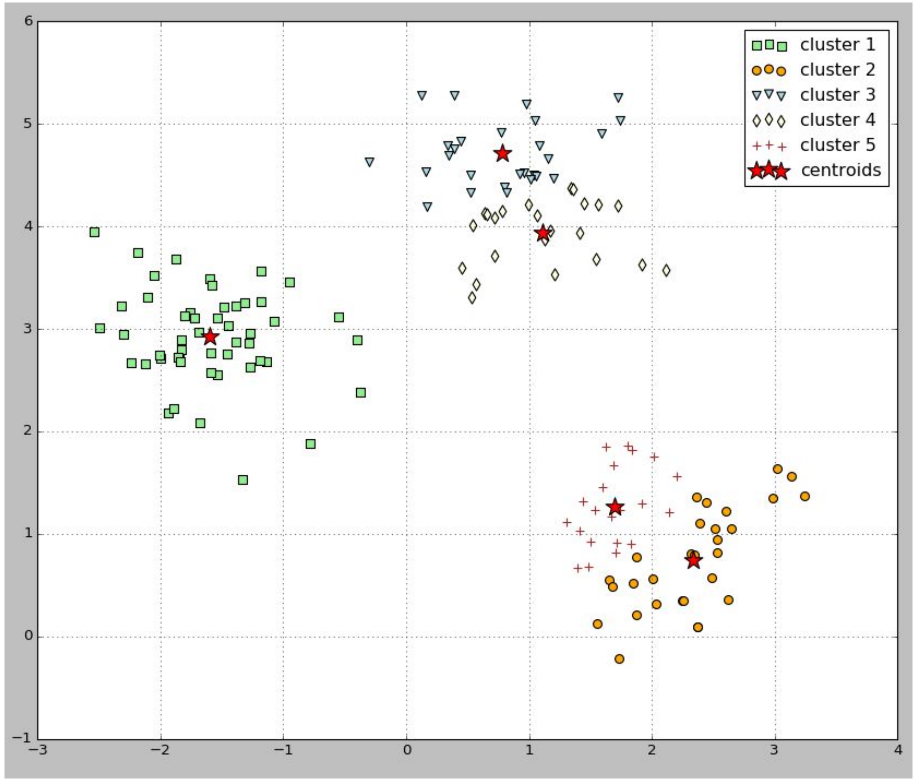
```
# plot the output clusters
plt.scatter(X[y_km == 0, 0], X[y_km == 0, 1], s=50, c='lightgreen', marker='s', label='cluster 1')

plt.scatter(X[y_km == 1, 0], X[y_km == 1, 1], s=50, c='orange', marker='o', label='cluster 2')

plt.scatter(X[y_km == 2, 0], X[y_km == 2, 1], s=50, c='lightblue', marker='v', label='cluster 3')

plt.scatter(km.cluster_centers_[:, 0], km.cluster_centers_[:, 1], s=250, marker='*', c='red', label='centroids')

plt.legend()
plt.grid()
plt.show()
```

K-Means with Scikit-Learn

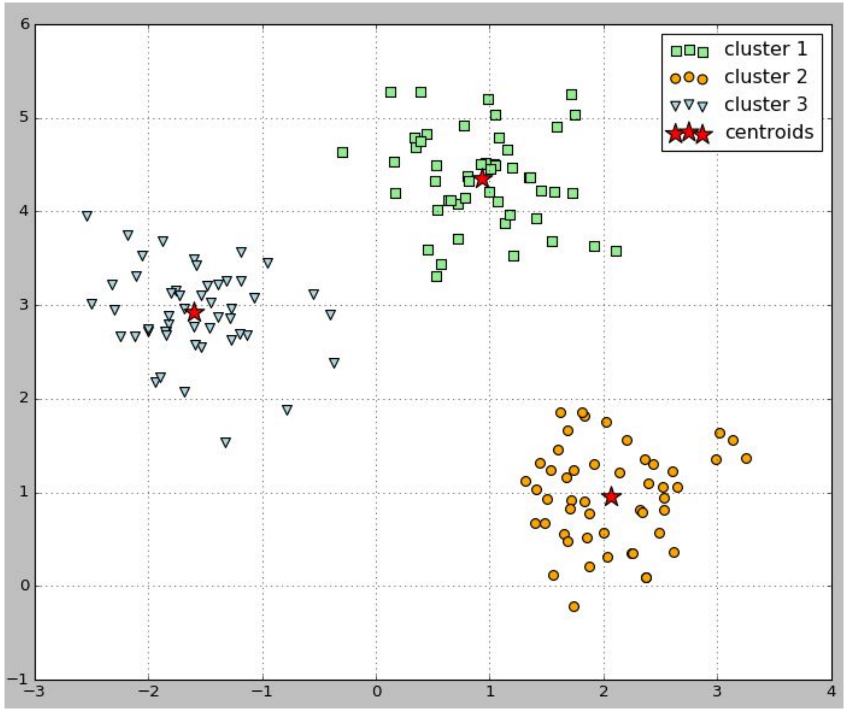


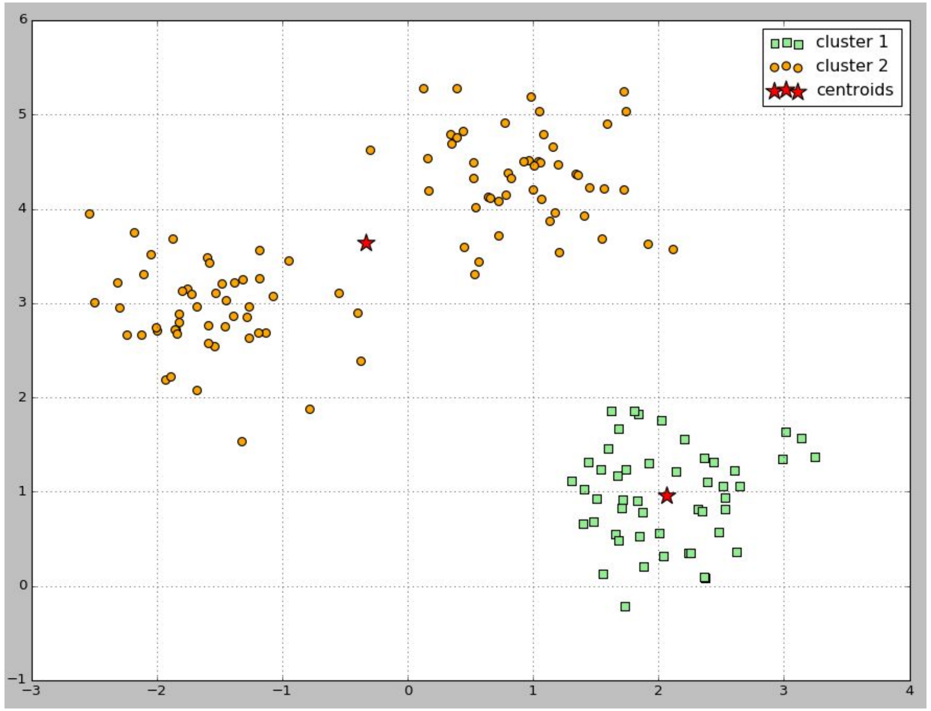
K-Means with Scikit-Learn





K-Means with Scikit-Learn





Questions