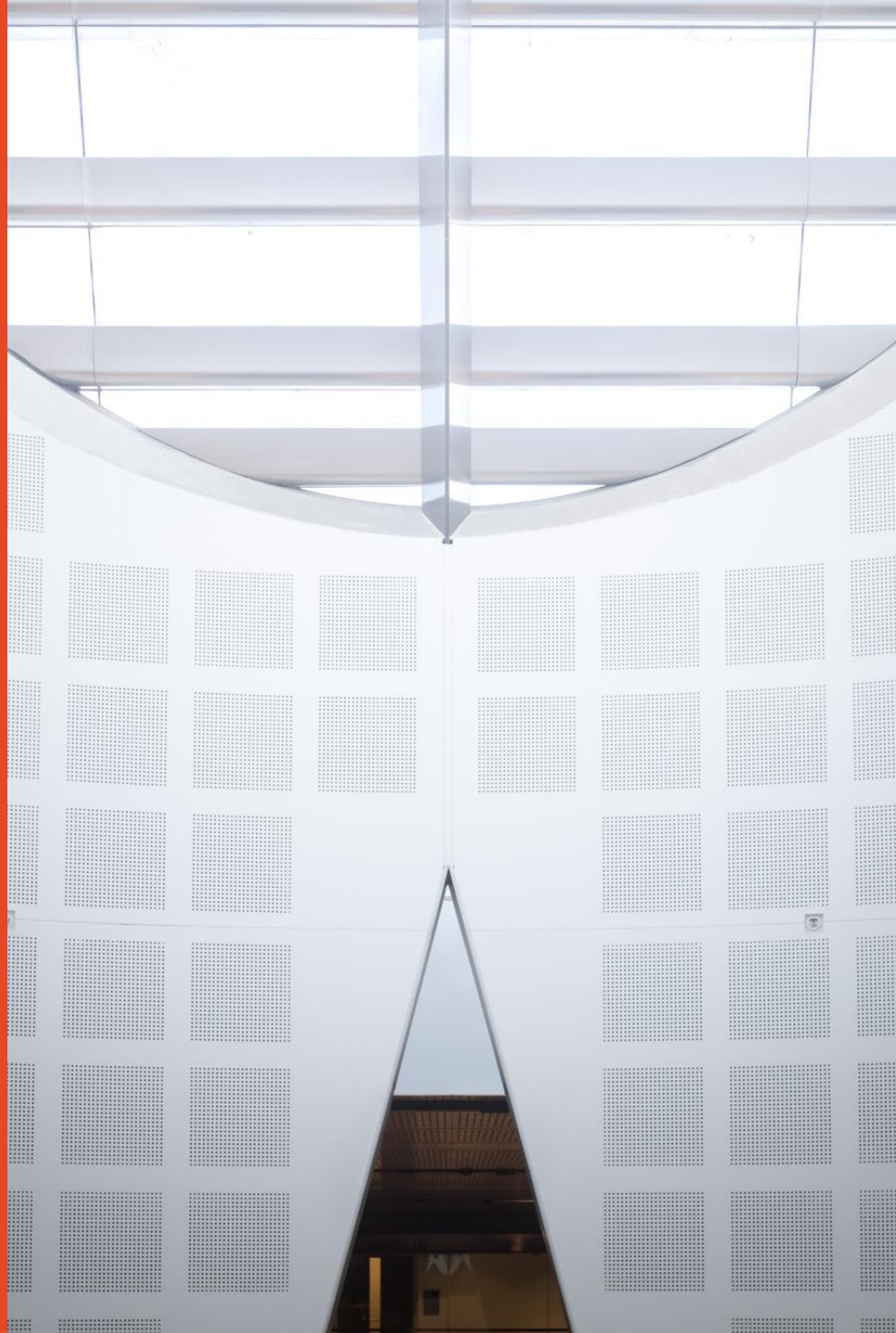
Genomic Data Analysis for Personalized Medicine & Clustering Tutorial

Reference: Healthcare Data Analytics, Chapter 6

THE UNIVERSITY OF SYDNEY



Review

In last tutorial, we mainly learn how to use sklearn package to train and test common clinical prediction models for clinical tasks:

- Linear regression

class sklearn.linear_model.**LinearRegression**(*, *fit_intercept=True*, *normalize='deprecated'*, *copy_X=True*, *n_jobs=None*, *positive=False*) [source]

- k-NN

class sklearn.neighbors.**KNeighborsClassifier**(*n_neighbors=5*, *, *weights='uniform'*, *algorithm='auto'*, *leaf_size=30*, *p=2*, *metric='minkowski'*, *metric_params=None*, *n_jobs=None*) [source]

- SVM

class sklearn.svm.**SVC**(*, *C=1.0*, *kernel='rbf'*, *degree=3*, *gamma='scale'*, *coef0=0.0*, *shrinking=True*, *probability=False*, *tol=0.001*, *cache_size=200*, *class_weight=None*, *verbose=False*, *max_iter=- 1*, *decision_function_shape='ovr'*, *break_ties=False*, *random_state=None*) [source]

- Preceptron

class sklearn.linear_model.**Perceptron**(*, *penalty=None*, *alpha=0.0001*, *l1_ratio=0.15*, *fit_intercept=True*, *max_iter=1000*, *tol=0.001*, *shuffle=True*, *verbose=0*, *eta0=1.0*, *n_jobs=None*, *random_state=0*, *early_stopping=False*, *validation_fraction=0.1*, *n_iter_no_change=5*, *class_weight=None*, *warm_start=False*) [source]

All these classifiers need the label of the data for training:

- fit(X, **y**)

They are called supervised learning which uses the label to help the optimization of the model.

Introduction

In this tutorial, we focus on three algorithms under clustering in unsupervised learning:

- Centroid-Based Clustering - K-Means
- Density-Based Clustering - DBSCAN
- Hierarchical-Based Clustering

In this tutorial you will learn their basic concepts and how to implement them by fetching the relevant packages

K-Means Clustering

Centroid-Based Clustering - K-Means

- K-means clustering is the most commonly used clustering algorithm. It's a centroid-based algorithm and the simplest unsupervised learning algorithm.
- This algorithm tries to minimize the variance of data points within a cluster. It's also how most people are introduced to unsupervised machine learning.
- K-means is best used on smaller data sets because it iterates over all of the data points. That means it'll take more time to classify data points if there are a large amount of them in the data set.

demo: http://47.74.87.250/visualization/visualizing-k-means-clustering/demo.html

Centroid-Based Clustering – K-Means

The Algorithm in a Nutshell:

Objective: Clusters the data into **K** groups.

1. **Init step**: Select k points at random as cluster centres.
2. **Assignment step**: Assign each instance to the cluster with the closest mean according to the distance function.
3. **Update step**: Calculate the new means to be the centroids of the instances in the new clusters.
4. Repeat steps 2 and 3 until the assignment no longer change.

- ***Note**: There is no guarantee that the optimum is found using this algorithm.*

Centroid-Based Clustering – K-Means

Get started by generating a set of data using the `make_blobs` class.

sklearn.datasets.make_blobs

sklearn.datasets.**make_blobs**(*n_samples=100, n_features=2, *, centers=None, cluster_std=1.0, center_box=(- 10.0, 10.0), shuffle=True, random_state=None, return_centers=False*) [source]

Parameters of `make_blobs`:

- `n_samples`: The total number of points equally divided among clusters.
  - Choose a number from 10-1500

Centroid-Based Clustering – K-Means

sklearn.datasets.make_blobs

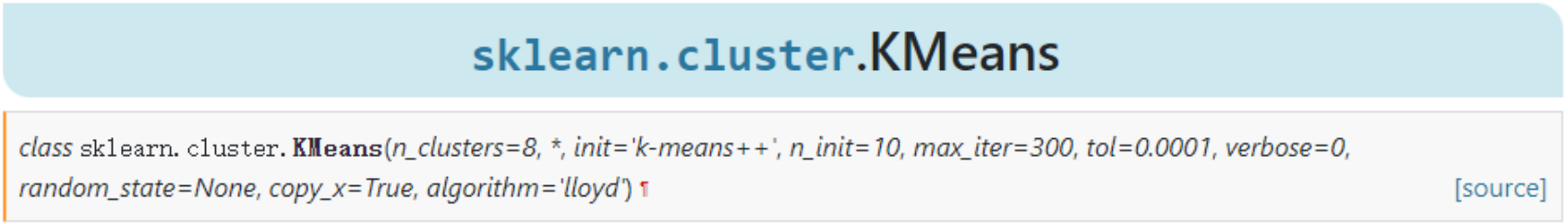
sklearn.datasets.**make_blobs**(*n_samples=100, n_features=2, *, centers=None, cluster_std=1.0, center_box=(- 10.0, 10.0), shuffle=True, random_state=None, return_centers=False*) [source]

- `centers`: The number of centers to generate, or the fixed center locations.
  - Choose arrays of x,y coordinates for generating the centers. Have 1-10 centers (ex. centers=[[1,1], [2,5]])

- `cluster_std`: The standard deviation of the clusters. The larger the number, the further apart the clusters
  - Choose a number between 0.5-1.5

Centroid-Based Clustering – K-Means

The scikit-learn library has an implementation of the k-means algorithm. We can use it directly.



Parameters are the values you need to define when initise a K-Means instance. The primary parameters includes:

- n_clusters
- n_init
- max_iter

...

Check this link about how KMeans works step by step: https://www.youtube.com/watch?v=5I3Ei69I40s

Centroid-Based Clustering – K-Means

sklearn.cluster.KMeans

class sklearn.cluster.**KMeans**(*n_clusters=8, *, init='k-means++', n_init=10, max_iter=300, tol=0.0001, verbose=0, random_state=None, copy_x=True, algorithm='lloyd'*) [source]

Parameter name: `n_clusters`

Meaning: The number of clusters.

Type: int, default=8

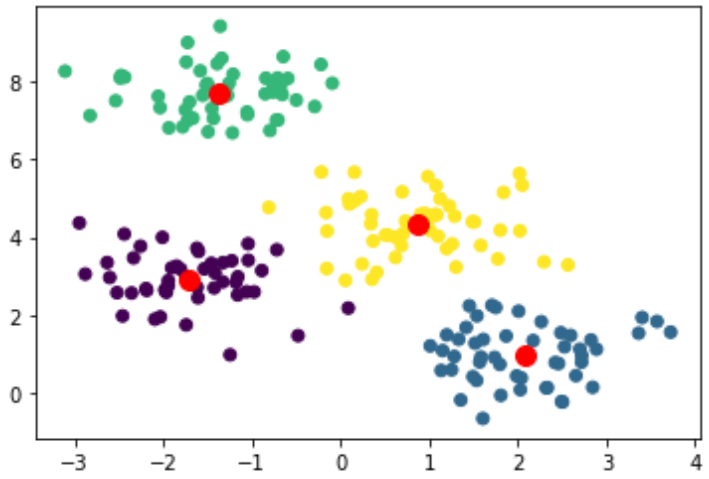
The number of clusters to form as well as the number of centroids to generate.

Centroid-Based Clustering – K-Means

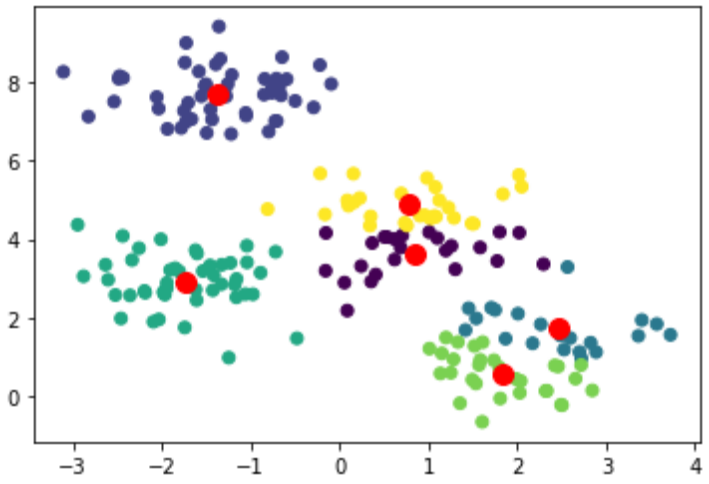
sklearn.cluster.KMeans

class sklearn.cluster.**KMeans**(*n_clusters=8, *, init='k-means++', n_init=10, max_iter=300, tol=0.0001, verbose=0, random_state=None, copy_x=True, algorithm='lloyd'*) [source]

Parameter name: `n_clusters`



`n_clusters = 4`



`n_clusters = 6`

Centroid-Based Clustering – K-Means

sklearn.cluster.KMeans

class sklearn.cluster.**KMeans**(*n_clusters=8, *, init='k-means++', n_init=10, max_iter=300, tol=0.0001, verbose=0, random_state=None, copy_x=True, algorithm='lloyd'*) [source]

Parameter name: `n_init`

Meaning: Number of time the k-means algorithm will be run with different centroid seeds. The final results will be the best output of n_init consecutive runs in terms of inertia.

Type: int, default=10

Hint: Try different n_init, observe the result.

Centroid-Based Clustering – K-Means

sklearn.cluster.KMeans

class sklearn.cluster.**KMeans**(*n_clusters=8, *, init='k-means++', n_init=10, max_iter=300, tol=0.0001, verbose=0, random_state=None, copy_x=True, algorithm='lloyd')* [source]

Parameter name: `max_iter`

Meaning: Maximum number of iterations of the k-means algorithm for a single run.

Type: int, default=300

Centroid-Based Clustering – K-Means

sklearn.cluster.KMeans

class sklearn.cluster.**KMeans**(*n_clusters=8, *, init='k-means++', n_init=10, max_iter=300, tol=0.0001, verbose=0, random_state=None, copy_x=True, algorithm='lloyd')* [source]

Parameter name: `algorithm`

Meaning: K-means algorithm to use. The classical EM-style algorithm is `"auto"`. The `"elkan"` variation can be more efficient on some datasets with well-defined clusters, by using the triangle inequality. However it's more memory intensive.

Type: {"lloyd", "elkan", "auto", "full"}, default="auto"

Note: `"auto"` and `"full"` are deprecated and they will be removed in Scikit-Learn 1.3. They are both aliases for `"lloyd"`.

Centroid-Based Clustering – K-Means

Parameter name: `algorithm`

Expectation–maximization (E–M) is a powerful algorithm that comes up in a variety of contexts within data science. K-means is a particularly simple and easy-to-understand application of the algorithm, and we will walk through it briefly here. In short, the expectation–maximization approach here consists of the following procedure:

1. Guess some cluster centres
2. Repeat until converged
    1. E-Step: assign points to the nearest cluster centre
    2. M-Step: set the cluster centres to the mean

Centroid-Based Clustering – K-Means

Parameter name: `algorithm`

Although the E–M procedure is guaranteed to improve the result in each step, there is no assurance that it will lead to the global best solution.

E–M approach can be converged, but may not converged to a globally optimal configuration. For this reason, it is common for the algorithm to be run for multiple starting guesses, as indeed Scikit-Learn does by default (set by the ``n_init`` parameter, which defaults to 10).

Centroid-Based Clustering – K-Means

sklearn.cluster.KMeans

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

KMeans also has some supporting methods:

Method name: `fit`

Meaning: Compute k-means clustering.

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

Note: Training instances to cluster. It must be noted that the data will be converted to C ordering, which will cause a memory copy if the given data is not C-contiguous. If a sparse matrix is passed, a copy will be made if it's not in CSR format.

DBSCAN Clustering

Density-Based Clustering - DBSCAN

The DBSCAN algorithm takes a different approach. Rather than having to provide the number of clusters, *K*, you define parameters related to *neighbourhoods* and *target density*.

DBSCAN algorithm works with two parameters:

- **MinPoints:** This refers to the minimum number of points needed to construct a cluster. We consider MinPoints as a threshold for considering a cluster as a cluster. A cluster is only recognized if the number of points is greater than or equal to the MinPts.

- **Epsilon (Eps):** This is the least distance required for two points to be termed as a neighbour. This distance is known as Epsilon (Eps). Thus we consider Eps as a threshold for considering two points as neighbours, i.e., if the distance between two points is utmost Eps, then we consider the two points to be neighbours.

demo: http://47.74.87.250/visualization/visualizing-dbscan-clustering/demo.html

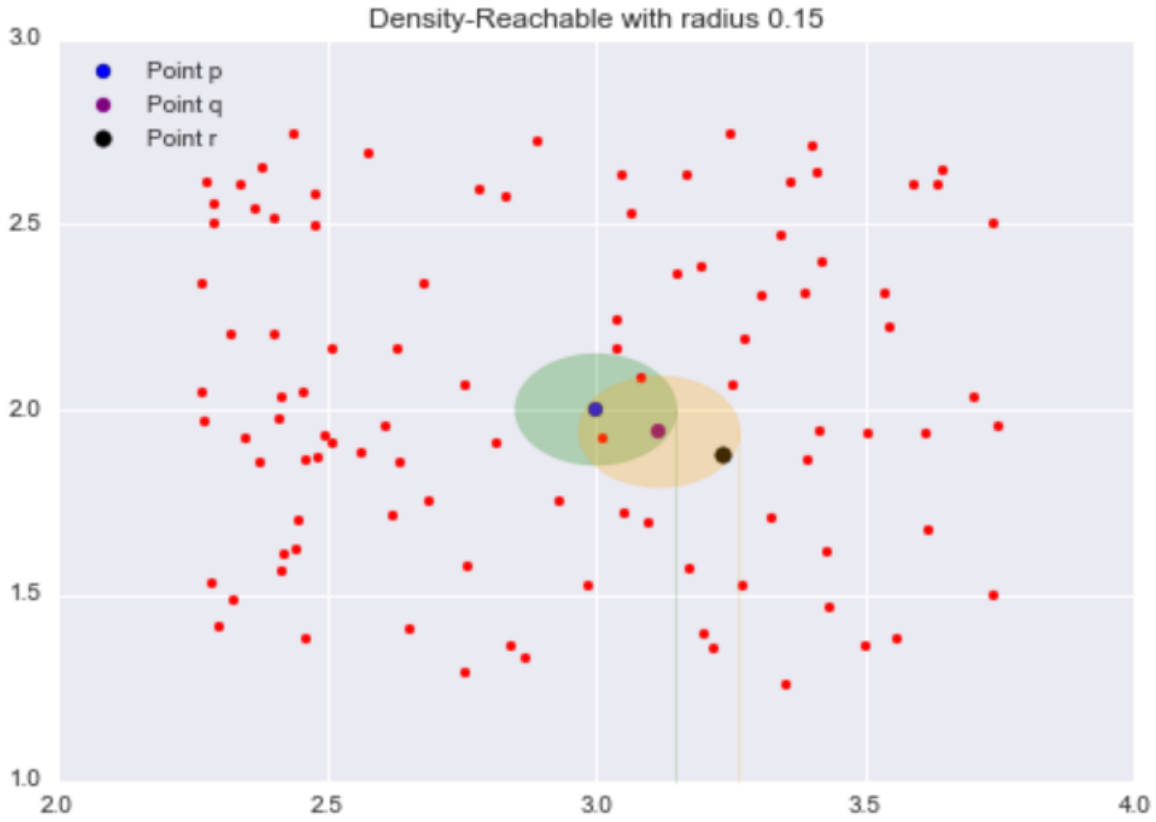
Density-Based Clustering - DBSCAN

- *Core Points*: A data point *p* is a *core point* if **Nbhd**(*p*,ε) [ε-neighborhood of*p*] contains at least *minPts* ; |**Nbhd**(*p*,ε)| >= *minPts*.
- *Border Points: A data point *q* is a *border point* if **Nbhd**(*q*, ε) contains less than *minPts* data points, but *q* is *reachable* from some *core point p*.
- *Outlier*: A data point *o* is an *outlier* if it is neither a core point nor a border point. Essentially, this is the "other" class.

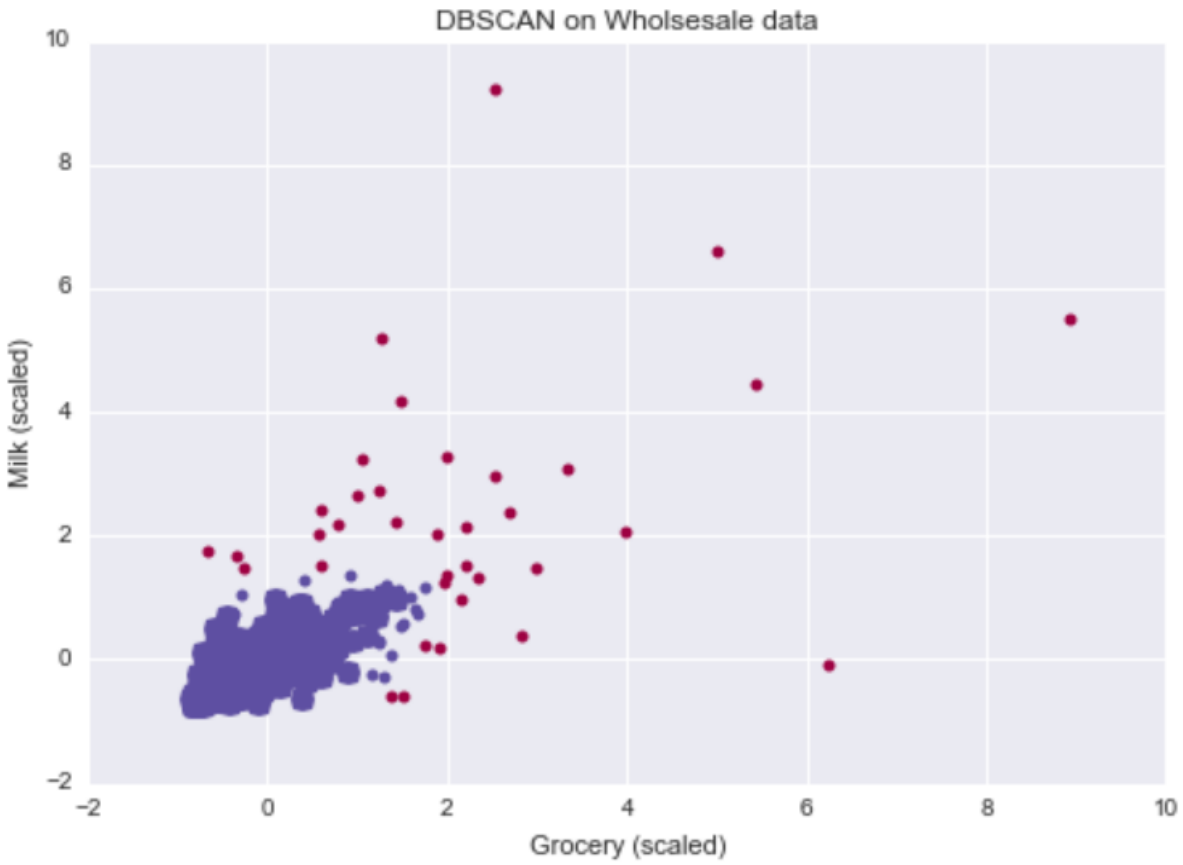
demo: http://47.74.87.250/visualization/visualizing-dbscan-clustering/demo.html

Density-Based Clustering - DBSCAN

- Assuming min_point is 3



Density-Based Clustering - DBSCAN



Density-Based Clustering - DBSCAN

MinPoints: We can obtain the minimum number of Points to be used to recognize a cluster, as follows:

- If the dataset has two dimensions, use the min sample per cluster as 4.
- If the data has more than two dimensions, the min sample per cluster should be: Min_sample(MinPoints) = 2 * Data dimension

Since our data is two-dimensional, we shall use the default value of 4 as our MinPoint parameter.

Density-Based Clustering - DBSCAN

Epsilon (Eps): To calculate the value of Eps, we shall calculate the distance between each data point to its closest neighbour using the Nearest Neighbours. After that, we sort them and finally plot them. From the plot, we identify the maximum value at the curvature of the graph. This value is our Eps.

We can employ `NearestNeighbors` in scikit-learn library to help us calculate the closest neighbour.

sklearn.neighbors.NearestNeighbors

class sklearn.neighbors.**NearestNeighbors**(*, *n_neighbors=5, radius=1.0, algorithm='auto', leaf_size=30, metric='minkowski', p=2, metric_params=None, n_jobs=None*) [source]

Density-Based Clustering - DBSCAN

sklearn.neighbors.NearestNeighbors

class sklearn.neighbors.**NearestNeighbors**(*, *n_neighbors=5, radius=1.0, algorithm='auto', leaf_size=30, metric='minkowski', p=2, metric_params=None, n_jobs=None*) [source]

Parameter name: `n_neighbors`

Meaning: Number of neighbors to use by default for `kneighbors` queries.

Type: int, default=5

Define the number of neighbours

Density-Based Clustering - DBSCAN

sklearn.neighbors.NearestNeighbors

class sklearn.neighbors.**NearestNeighbors**(*, *n_neighbors=5, radius=1.0, algorithm='auto', leaf_size=30, metric='minkowski', p=2, metric_params=None, n_jobs=None*) [source]

Parameter name: `radius`

Meaning: Range of parameter space to use by default for `radius_neighbors` queries.

Type: float, default=1.0

Density-Based Clustering - DBSCAN

sklearn.neighbors.NearestNeighbors

class sklearn.neighbors.**NearestNeighbors**(*, *n_neighbors=5, radius=1.0, algorithm='auto', leaf_size=30, metric='minkowski', p=2, metric_params=None, n_jobs=None*) [source]

And other Parameters: `algotithm, leaf_size, metric...`

We will not use these parameters specifically in this tutorial, we just need to keep them as default parameters. Of course, if you are interested, you can try to change them and see what interesting results you get!

Density-Based Clustering - DBSCAN

sklearn.neighbors.NearestNeighbors

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

DBSCAN also has some supporting methods:

Method name: `fit`

Meaning: Fit the nearest neighbors estimator from the training dataset.

Parameter name: X{array-like, sparse matrix} of shape (n_samples, n_features) or (n_samples, n_samples) if metric='precomputed'

Note: Training data.

Density-Based Clustering - DBSCAN

sklearn.neighbors.NearestNeighbors

kneighbors(*X=None, n_neighbors=None, return_distance=True*) [source]

Method name: `kneighbors`

Meaning: Find the K-neighbors of a point. Returns indices of and distances to the neighbors of each point.

Parameter name: X: array-like, shape (n_queries, n_features), or (n_queries, n_indexed) if metric == 'precomputed', default=None

Note: The query point or points. If not provided, neighbors of each indexed point are returned. In this case, the query point is not considered its own neighbor.

Density-Based Clustering - DBSCAN

The scikit-learn library has an implementation of the DBSCAN algorithm. We can use it directly.

sklearn.cluster.DBSCAN

class sklearn.cluster.**DBSCAN**(*eps=0.5, *, min_samples=5, metric='euclidean', metric_params=None, algorithm='auto', leaf_size=30, p=None, n_jobs=None*) [source]

Parameters are the values you need to define when initlise a DBSCAN instance. The primary parameters includes:

- eps
- min_samples

...

Density-Based Clustering - DBSCAN

sklearn.cluster.DBSCAN

class sklearn.cluster.**DBSCAN**(*eps=0.5, *, min_samples=5, metric='euclidean', metric_params=None, algorithm='auto', leaf_size=30, p=None, n_jobs=None*) [source]

Parameter name: `eps`

Meaning: The maximum distance between two samples for one to be considered as in the neighborhood of the other. This is not a maximum bound on the distances of points within a cluster. This is the most important DBSCAN parameter to choose appropriately for your data set and distance function.

Type: float, default=0.5

We have already got the value of eps above, so we can bring it in directly here.

Density-Based Clustering - DBSCAN

sklearn.cluster.DBSCAN

class sklearn.cluster.**DBSCAN**(*eps=0.5, *, min_samples=5, metric='euclidean', metric_params=None, algorithm='auto', leaf_size=30, p=None, n_jobs=None*) [source]

Parameter name: `min_samples`

Meaning: The number of samples (or total weight) in a neighborhood for a point to be considered as a core point. This includes the point itself.

Type: int, default=5

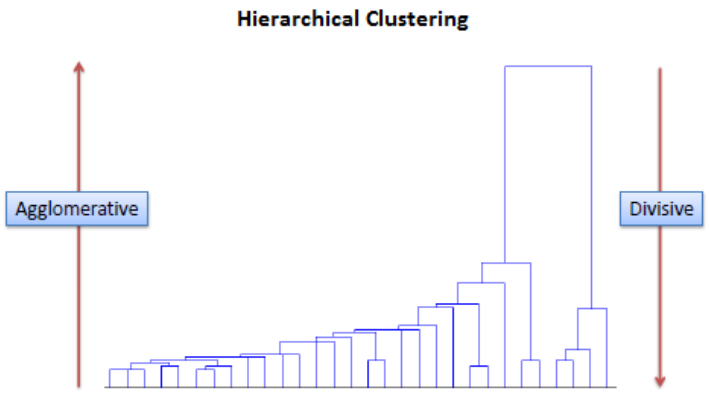
Since our data is two-dimensional, we shall use the default value of 4 as our `min_samples` value.

Hierarchical-Based Clustering

Hierarchical-Based Clustering

Hierarchical clustering algorithms group similar objects into groups called clusters. There are two types of hierarchical clustering algorithms:

- Agglomerative — Bottom up approach. Start with many small clusters and merge them together to create bigger clusters.
- Divisive — Top down approach. Start with a single cluster than break it up into smaller clusters.
- In this tutorial, we will be learning to implement the agglomeration clustering.



Hierarchical-Based Clustering

The scikit-learn library has an implementation of the Agglomerative clustering . We can use it directly.

sklearn.cluster.AgglomerativeClustering

class sklearn.cluster.**AgglomerativeClustering**(*n_clusters=2, *, affinity='euclidean', memory=None, connectivity=None, compute_full_tree='auto', linkage='ward', distance_threshold=None, compute_distances=False*) [source]

Parameters are the values you need to define when initlise an agglomerative clustering instance. The primary parameters includes:

- n_clusters
- linkage

...

Hierarchical-Based Clustering

sklearn.cluster.AgglomerativeClustering

class sklearn.cluster.**AgglomerativeClustering**(*n_clusters=2, *, affinity='euclidean', memory=None, connectivity=None, compute_full_tree='auto', linkage='ward', distance_threshold=None, compute_distances=False*) [source]

Parameter name: `n_clusters`

Meaning: The number of clusters to find. It must be `None` if `distance_threshold` is not `None`.

Type: int or None, default=2

Hierarchical-Based Clustering

sklearn.cluster.AgglomerativeClustering

class sklearn. cluster. **AgglomerativeClustering**(*n_clusters=2, *, affinity='euclidean', memory=None, connectivity=None, compute_full_tree='auto', linkage='ward', distance_threshold=None, compute_distances=False*) [source]

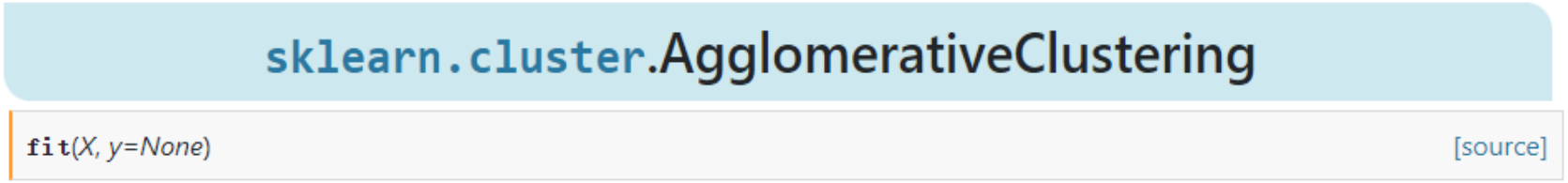
Parameter name: `linkage`

Meaning: Which linkage criterion to use. The linkage criterion determines which distance to use between sets of observation. The algorithm will merge the pairs of cluster that minimize this criterion.

Type: linkage{'ward', 'complete', 'average', 'single'}, default='ward'

- 'average' uses the average of the distances of each observation of the two sets
- 'complete' or 'maximum' linkage uses the maximum distances between all observations of the two sets.
- 'ward' uses the sum of squared estimate of errors (SSE) of the two sets to calculate the distances. (SSE_{AB}-SSE_A-SSE_B)

Hierarchical-Based Clustering



Agglomerative clustering also has some supporting methods:

Method name: `fit`

Meaning: Fit the hierarchical clustering from features, or distance matrix.

Parameter name: X: array-like, shape (n_samples, n_features) or (n_samples, n_samples)

Note: Training instances to cluster, or distances between instances.

Hierarchical-Based Clustering

sklearn.preprocessing.MinMaxScaler

class sklearn.preprocessing.**MinMaxScaler**(*feature_range=(0, 1), *, copy=True, clip=False*) [source]

Meaning: Transform features by scaling each feature to a given range. This estimator scales and translates each feature individually such that it is in the given range on the training set, e.g. between zero and one.

fit_transform(*X, y=None, **fit_params*) [source]

Meaning: Fit to data, then transform it. Fits transformer to `X` and `y` with optional parameters `fit_params` and returns a transformed version of `X`.

Parameter name: Xarray-like of shape (n_samples, n_features)

Input samples.

Hierarchical-Based Clustering

In order to present Agglomerative clustering as a dendrogram, here we need to use an additional new package.

In this section we will be using the **`scipy`** package, which is similar to **`sklearn`** that contains many implementations of algorithm functions.


```
from scipy.spatial import distance_matrix

from scipy.cluster import hierarchy
```

Hierarchical-Based Clustering

- A distance matrix contains the distance from each point to every other point of a dataset .
- Use the function `distance_matrix`, which requires two inputs.
- We use the Feature Matrix, `X1` as both inputs and save the distance matrix to a variable called `dist_matrix`
- Using the `linkage` class from `hierarchy`, pass in the parameters:
  - The distance matrix `dist_matrix`
  - `'complete'` for complete linkage

Density-Based Clustering - DBSCAN

scipy.spatial.distance_matrix

```
scipy.spatial.distance_matrix(x, y, p=2, threshold=1000000)
```

Method name: `distance_matrix`

Meaning: Compute the distance matrix. Returns the matrix of all pair-wise distances.

Parameter name:

- x(M, K) array_like: Matrix of M vectors in K dimensions.
- y(N, K) array_like: Matrix of N vectors in K dimensions.

Density-Based Clustering - DBSCAN

scipy.cluster.hierarchy.linkage

```
scipy.cluster.hierarchy.linkage(y, method='single', metric='euclidean',
optimal_ordering=False)
```

Method name: `linkage`

Meaning: Perform hierarchical/agglomerative clustering.

Parameter name:

- y: The input y may be either a 1-D condensed distance matrix or a 2-D array of observation vectors.

- method: Linkage methods are used to compute the distance d(s,t) between two clusters s and t. The algorithm begins with a forest of clusters that have yet to be used in the hierarchy being formed.

Hierarchical-Based Clustering

- A Hierarchical clustering is typically visualized as a dendrogram. Each merge is represented by a horizontal line. The y-coordinate of the horizontal line is the similarity of the two clusters that were merged, where cities are viewed as singleton clusters. By moving up from the bottom layer to the top node, a dendrogram allows us to reconstruct the history of merges that resulted in the depicted clustering.

- Next, save the dendrogram to a variable called `dendro`. In doing this, the dendrogram will also be displayed.

Density-Based Clustering - DBSCAN

scipy.cluster.hierarchy.dendrogram

```
scipy.cluster.hierarchy.dendrogram(Z, p=30, truncate_mode=None, color_threshold=None,
get_leaves=True, orientation='top', labels=None, count_sort=False, distance_sort=False,
show_leaf_counts=True, no_plot=False, no_labels=False, leaf_font_size=None,
leaf_rotation=None, leaf_label_func=None, show_contracted=False, link_color_func=None,
ax=None, above_threshold_color='C0')                                          [source]
```

Method name: `dendrogram`

Meaning: Plot the hierarchical clustering as a dendrogram.

Parameter name:

Z:ndarray

The linkage matrix encoding the hierarchical clustering to render as a dendrogram.

Hierarchical-Based Clustering

- Now, we can use the '`AgglomerativeClustering`' function from `scikit-learn` library to cluster the dataset. The `AgglomerativeClustering` performs a hierarchical clustering using a bottom up approach. The linkage criteria determines the metric used for the merge strategy:
  - Maximum or complete linkage minimizes the maximum distance between observations of pairs of clusters.
  - Average linkage minimizes the average of the distances between all observations of pairs of clusters.