PennState World Campus

DAAN862: Analytics Programming in Python

Lesson 11: Lesson 11: Unsupervised learning with Scikit-Learn: Clustering

Lesson 11: Objectives and Overview (1 of 5)

Lesson 11: Unsupervised learning with Scikit-Learn: Clustering

In SWENG 545 you learned about the clustering models. Here, we provide a quick walkthrough of how to use clustering models provided by scikit-learn package in Python. At the end of this lesson, students will be able to use scikit-learn packages to:

Build and evaluatie K-means using clustering models

Build and evaluate Hierarchical clustering models

Build and evaluate DBscan clustering models

By the end of this lesson, please complete all readings and assignments found in the Lesson 11 Course Schedule.

Lesson 11.1: K-means (2 of 5)

Lesson 11.1: K-means

(click titles to learn more)

K-Means

Kmeans (http://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html#sklearn.cluster.KMeans) is the class for k-means clustering model. The important parameter for K-means algorithm is listed in Table 11.1.1:

Table 11.1.1: Main Parameters of Kmeans

Parameters	Description
n cluster	The number of clusters to form as well as the number of centroids to generate.
init	Method for initialization, defaults to 'k-means++':
n init	Number of time the k-means algorithm will be run with different centroid seeds.

Parameters	Description
max iter	Maximum number of iterations of the k-means algorithm for a single run.
tol	Relative tolerance with regards to inertia to declare convergence

Table 11.1.2 lists the main attributes of Kmeans:

Table 11.1.2: Main Attributes of Kmeans

Attributes	Description
cluster centers	Coordinates of cluster centers.
labels	Labels of each point.
inertia	Sum of squared distances of samples to their closest cluster center.

To create Kmeans, first we will need to import the necessay packages shown in Figure 11.1:

```
In [1]: import os
   ...: import pandas as pd
   ...: import matplotlib.pyplot as plt
   ...: from sklearn.cluster import KMeans
   ...: from sklearn import metrics
   ...: from sklearn.preprocessing import MinMaxScaler
```

Fig 11.1 (click to enlarge)

For all clustering models, we will continue to use iris datasets as the example in Figure 11.2:

```
In [2]: os.chdir('E:/GoogleDrive/PSU/DAAN862/Course contents/Lesson 11')
   ...: iris = pd.read_csv('iris.csv')
   ...: iris.species = iris.species.astype("category").cat.codes
```

Fig 11.2 (click to enlarge)

It is recommended to normalize independent variables before using clustering method. In this case, each variable will be equal-weighted in distance function. Rescalling and plitting the data into X and y is shown in Figure 11.3:

```
In [3]: scaler = MinMaxScaler()
   ...: X = scaler.fit_transform(iris.iloc[:, 0:4])
   ...: y = iris.species
```

Fig 11.3 (click to enlarge)

Figure 11.4 shows how we use n clusters=3, since iris has three species:

```
In [4]: kmeans = KMeans(n_clusters = 3, random_state= 130)
   ...: y_pred = kmeans.fit_predict(X)
```

Fig 11.4 (click to enlarge)

Figure 11.5 shows how the performace can be evaluated:

```
In [5]: metrics.homogeneity_score(y, y_pred)
Out[5]: 0.736419288125285

In [6]: metrics.completeness_score(y, y_pred)
Out[6]: 0.7474865805095325

In [7]: metrics.adjusted_rand_score(y, y_pred)
Out[7]: 0.7163421126838475

In [8]: metrics.silhouette_score(X, y_pred, metric = 'euclidean')
Out[8]: 0.5043188549150884
```

Fig 11.5 (click to enlarge)

The centers of clusters are stored in *cluster centers attribute* (shown in Figure 11.6). For the plot, we only use petal length and petal with variables. Figure 11.6 shows how to select these features from the centers:

```
In [9]: centers = kmeans.cluster_centers_
   ...: centers_pl = centers[:, 2]
   ...: centers_pw = centers[:, 3]
```

Fig 11.6 (click to enlarge)

The iris data (shown in Figure 11.7 B) shows the original iris data, and clustering results (shown in Figure 11.7 B) shows the result from K-means clustering:

```
In [10]: plt.figure(figsize = (12, 6))
    ...:  # create a plot with two subplots and plot the first subplot
    ...: plt.subplot(121)
    ...: # Scatter plot of petal_length and petal_width,  color represents
    ...: # the original categories (c = y).
    ...: plt.scatter(X[:, 2], X[:, 3], c = y)
    ...: plt.title('iris data')
    ...: plt.xlabel('Petal length')
    ...: plt.ylabel('Petal width')
    ...: # Plot the second subplot.
    ...: plt.subplot(122)
    ...: # Scatter plot of petal_length and petal_width, the color represents
the
    ...: # predicted  categories (c = y_pred).
    ...: plt.scatter(X[:, 2], X[:, 3], c = y_pred, label = 'Predicted')
    ...: # PLot the cluster centroids with 'X' and color is red
    ...: plt.scatter(centers_pl, centers_pw, s= 100, c = 'r',
    ...:             marker = 'x', label = 'Cluster Centers')
    ...: plt.title('Clustering results')
    ...: plt.xlabel('Petal length')
    ...: plt.ylabel('Petal width')
    ...: plt.legend()
    ...: # adjust the space between two subplots
    ...: plt.subplots_adjust(wspace = 0.2)
```

Fig 11.7 (A) (click to enlarge)

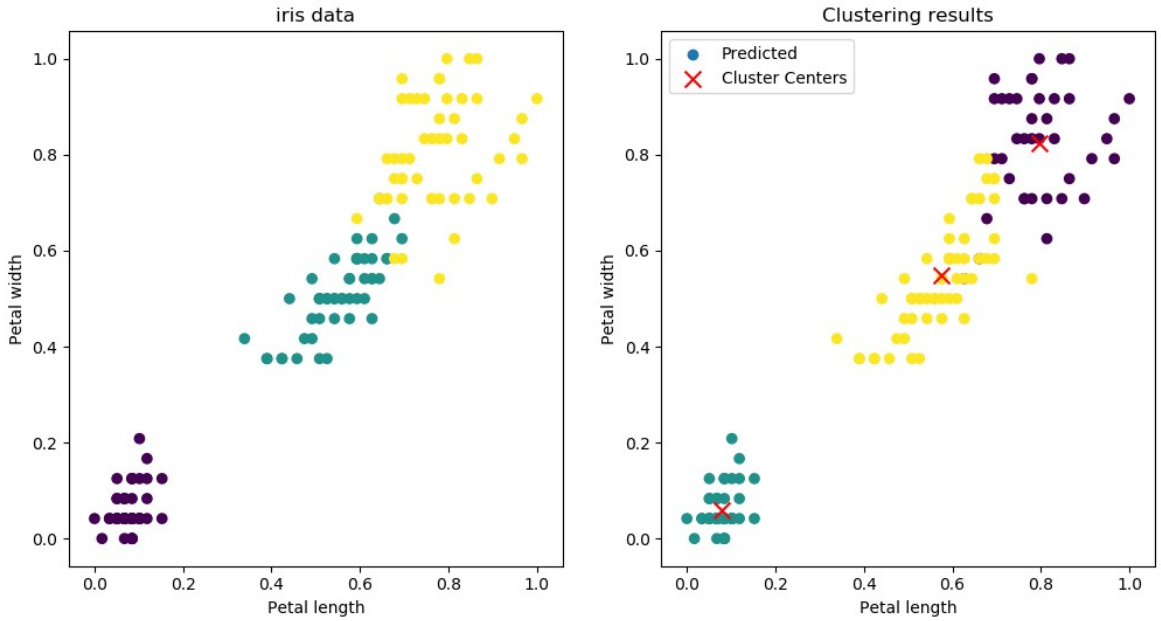


Fig 11.7 (B) (click to enlarge)

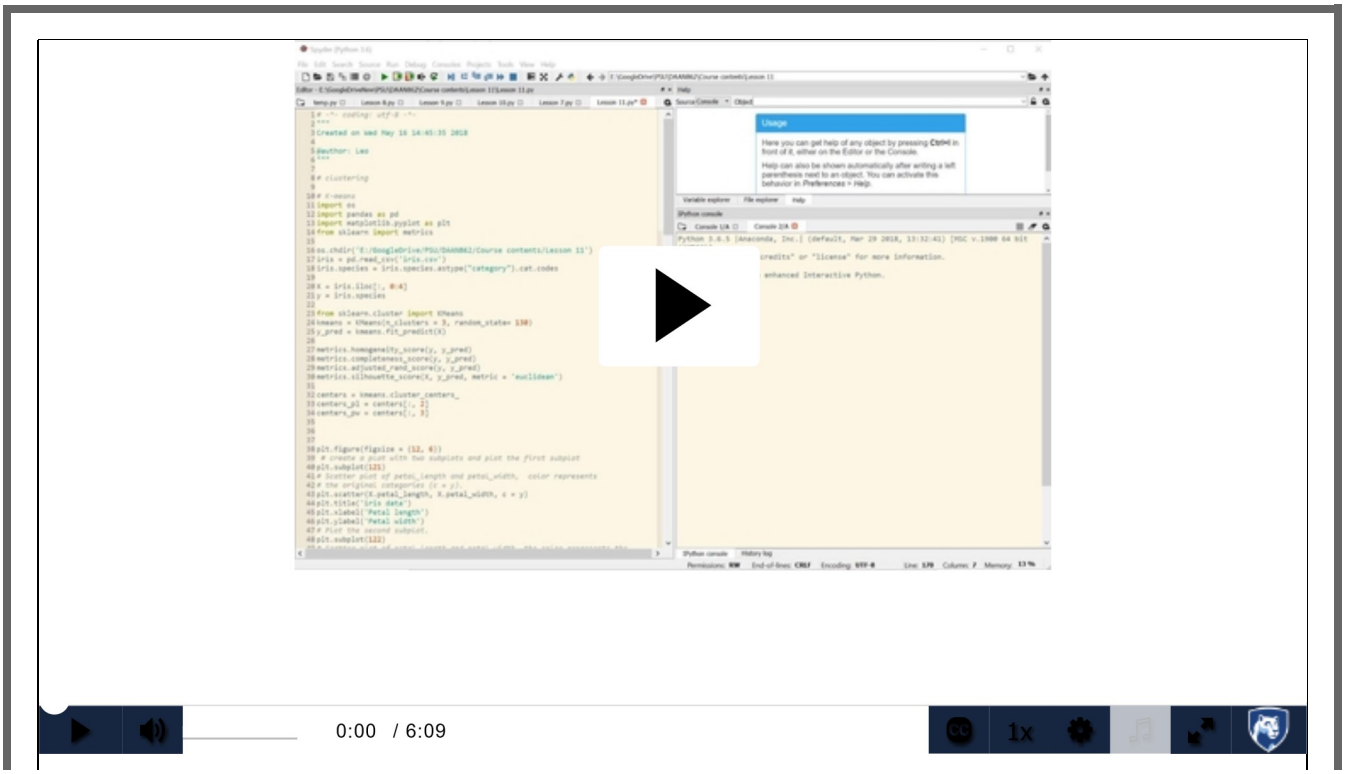
Advanced Practice

In the advanced practice stage lets assume that we didn't know how many species were in the iris data. In Figure 11.8 we use different n clusters and will find out which one gives the best Silhouette score:

```
In [11]: result = []

In [12]: nclusters = range(2, 7)

In [13]: for n in nclusters:
    ...:     y_pred_temp = KMeans(n_clusters = n).fit_predict(X)
    ...:     score = metrics.silhouette_score(X, y_pred_temp,metric =
'euclidean')
    ...:     result.append((n, score))

In [14]: result
Out[14]:
[(2, 0.6294675561906644),
 (3, 0.5043188549150884),
 (4, 0.4446273300650682),
 (5, 0.35538315267369003),
 (6, 0.3340773530959414)]
```

Fig 11.8 (click to enlarge)

It is interesting that K-means algorithm shows that iris data should be split into two groups since n_clusters = 2 gives the highest Silhouette score.



Transcript

In this video, I will introduce how to use K-means clustering with sklearn. First, let's import the basic packages: os, pandas, pyplot, metrics.

Here, we will continue using iris data. Here I convert the species to a numeric variable by converting it to category first then use the codes for it.

X are four independent variables, y is the species variable.

First, we import Kmeans object from sklearn.cluster. We create a Kmeans object with n_cluters = 3, which means the number of clusters is 3. And set the random state to reproduce the results.

Since here we don't have a test data, I will use fit_predcit method since it will fit the data with Kmeans and predict the clusters numbers.

Now you the predicted value for y, you can use cluster metrics to evaluate it.

In order to view the cluster assignment, we will use petal length and petal width as x, y. First, we get the cluster centers from cluster_Centers attribute, petal length and petal width are column 2 and 3, respectively.

Set the figzie = 12 times 6.

For the first subplot 121, we plot the original data, the color is y

For the send subplot 122, we plot the results from Kmeans. C equals the y_pred.

Next we will add the cluster centers in the second plot, c equals to red. We use marker X.

If you don't know the number of clusters you should use for a data, you need to try a few numbers and select the one which gives the best results. Here we use silhouette score to select best n.

Results are the output and it's an empty list.

We want to try n_cluters from 2 to 7.

We build the Kmeans model with different c_clusters and fit_predict X.

From the results, we can see that based on silhouette score, n = 2 gives the best results. Since there isn't a clear separation between the values for Versicolor and Virginia.

Lesson 11.2: Hierarchical Clustering

The *agglomerativeClustering (http://scikit-learn.org/stable/modules/generated/sklearn.cluster.AgglomerativeClustering.html#sklearn.cluster.AgglomerativeClustering)* object is for a hierarchical clustering with a bottom-up approach. Moreover, the second clustering methods we introduce here is Hierarchical clustering.The important parameter for Hierarchicl clustering methods are listed in Table 11.1.2:

Table 11.2.1: Main Parameters of agglomerativeClustering

Parameters	Description
n cluster	The number of clusters to form as well as the number of centroids to generate
affinity	Metric used to compute the linkage. Can be “euclidean”, “l1”, “l2”, “manhattan”, “cosine”, or ‘precomputed’. If linkage is “ward”, only “euclidean” is accepted.
linkage	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. Ward minimizes the variance of the clusters being merged. 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.

The attributes of Table 11.2.1 are listed in Table 11.2.2:

Table 11.2.2: Main Attributes of agglomerativeClustering

Attributes	Description
labels_	Labels of each point
n leaves_	Number of leaves in the hierarchical tree.

Attributes	Description
n components_	The estimated number of connected components in the graph.
children_	The children of each non-leaf node.

You can import agglomerativeclustering object from the sklearn cluster: (shown in Figure 11.9):

```
In [15]: from sklearn.cluster import AgglomerativeClustering
```

Fig 11.9 (click to enlarge)

(click tabs to learn more about hierarchical clustering)

Linkage Type= Average:

For Hierarchical clustering, we tried a diferent linkage type and plot clusters according to thier predication:

Fig 11.10 (click to enlarge)

Unfortunately, the model didn't provide tools to plot the dendrogram. We have to use Scipy package for the plot. First, we import *dengrogram* and *linkage* functions into Python. The *linkage* (https://docs.scipy.org/doc/scipy/reference/generated/scipy.cluster.hierarchy.linkage.html) function is used to calculate the hierarchical clustering encoded as a linage matrix, and the *dendrogram* plots the hierarchical clustering as a dendrogram (https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.cluster.hierarchy.dendrogram.html) shown in Figure 11.11:

```
In [21]: from scipy.cluster.hierarchy import dendrogram, linkage

In [22]: Z_avg = linkage(X, method = 'average')

In [23]: plt.figure()
    ...: den = dendrogram(Z_avg, leaf_font_size = 8)
```

Fig 11.11 (click to enlarge)

Figure 11.12 shows the dendrogram for the average linkage method average:

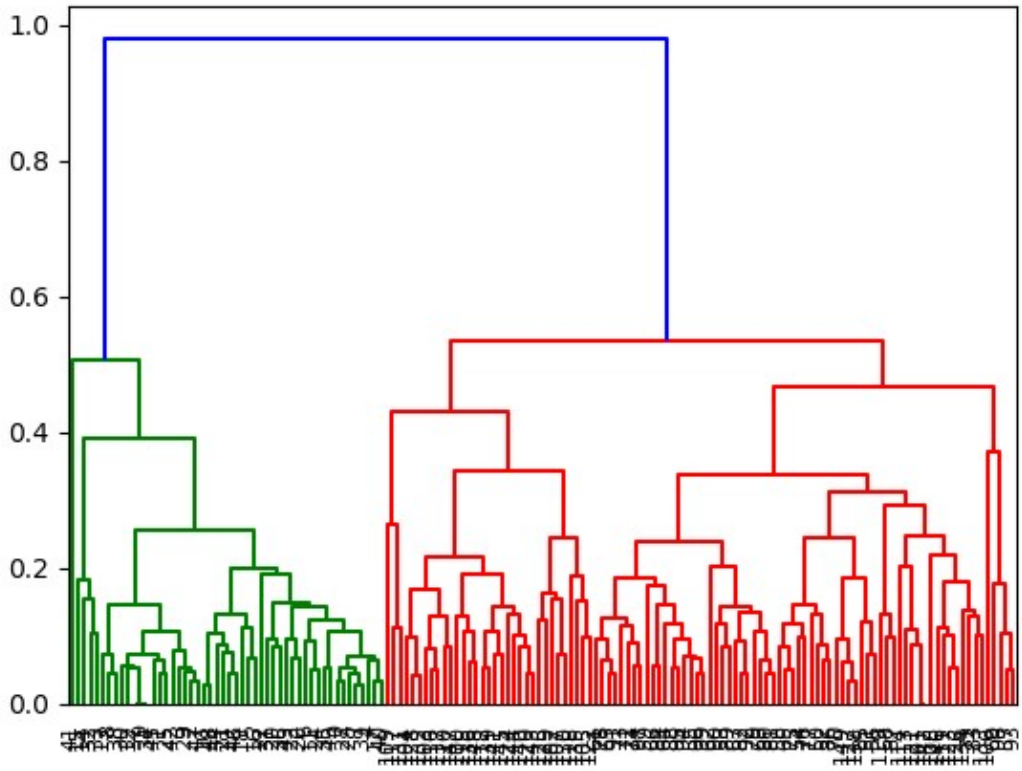


Fig 11.12 (click to enlarge)

Linkage Type= Complete:

Figure 11.13 shows the clustering results when you use linkage= 'complete' in the AgglomerativeClustering object:

```
In [24]: Hier_y_pred2 = AgglomerativeClustering(
    ...:         n_clusters = 3,
    ...:         affinity = 'euclidean',
    ...:         linkage = 'complete').fit_predict(X)

In [25]: metrics.homogeneity_score(y, Hier_y_pred2)
Out[25]: 0.7441323612776896

In [26]: metrics.completeness_score(y, Hier_y_pred2)
Out[26]: 0.7684322991258716

In [27]: metrics.adjusted_rand_score(y, Hier_y_pred2)
Out[27]: 0.7060059674676763

In [28]: metrics.silhouette_score(X, Hier_y_pred2, metric = 'euclidean')
Out[28]: 0.5030674466509139
```

Caption 11.13 (click to enlarge)

Linkage Type= Ward:

Figure 11.14 shows the clustering results when you use linkage= 'ward' in AgglomerativeClustering object:

```
In [29]: Hier_y_pred3 = AgglomerativeClustering(
    ...:         n_clusters = 3,
    ...:         affinity = 'euclidean',
    ...:         linkage = 'ward').fit_predict(X)

In [30]: metrics.homogeneity_score(y, Hier_y_pred3)
Out[30]: 0.7697177990193531

In [31]: metrics.completeness_score(y, Hier_y_pred3)
Out[31]: 0.7982832930540932

In [32]: metrics.adjusted_rand_score(y, Hier_y_pred3)
Out[32]: 0.7195837484778037

In [33]: metrics.silhouette_score(X, Hier_y_pred2, metric = 'euclidean')
Out[33]: 0.5030674466509139
```

Fig 11.14 ((click to enlarge)

Figures 11.15 (A&B) shows how to plot iris data and the results from hierarchical clustering models with different linkage types:

Click dropdown to view examples:

```
In [34]: plt.figure(figsize = (12, 12))
    ...: plt.subplot(221)
    ...: plt.scatter(X[:, 2], X[:, 3], c = y)
    ...: plt.title('iris data')
    ...: plt.xlabel('Petal length')
    ...: plt.ylabel('Petal width')
    ...:
    ...: # Plot the iris data colored by the average linkage prediction
    ...: plt.subplot(222)
    ...: plt.scatter(X[:, 2], X[:, 3], c = Hier_y_pred1)
    ...: plt.title('Average linkage')
    ...: plt.xlabel('Petal length')
    ...: plt.ylabel('Petal width')
    ...:
    ...: # Plot the iris data colored by the complete linkage predictio
    ...: plt.subplot(223)
    ...: plt.scatter(X[:, 2], X[:, 3], c = Hier_y_pred2)
    ...: plt.title('Complete linkage')
    ...: plt.xlabel('Petal length')
    ...: plt.ylabel('Petal width')
    ...:
    ...: # Plot the iris data colored by the ward linkage prediction.
    ...: plt.subplot(224)
    ...: plt.scatter(X[:, 2], X[:, 3], c = Hier_y_pred3)
    ...: plt.title('Ward linkage')
    ...: plt.xlabel('Petal length')
    ...: plt.ylabel('Petal width')
    ...:
    ...: plt.subplots_adjust(wspace = 0.2)
```

Fig 11.15 (A) (click to enlarge)

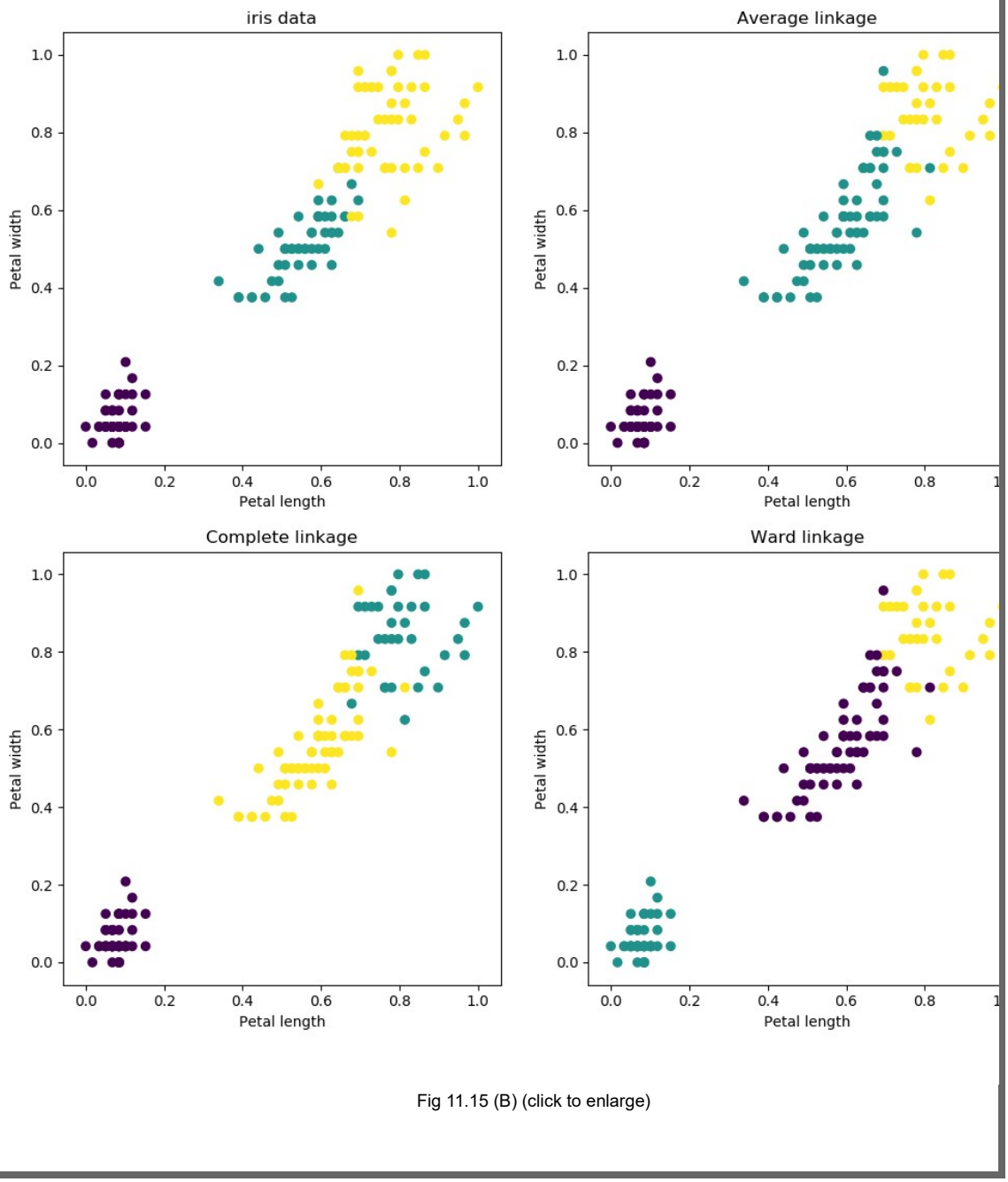
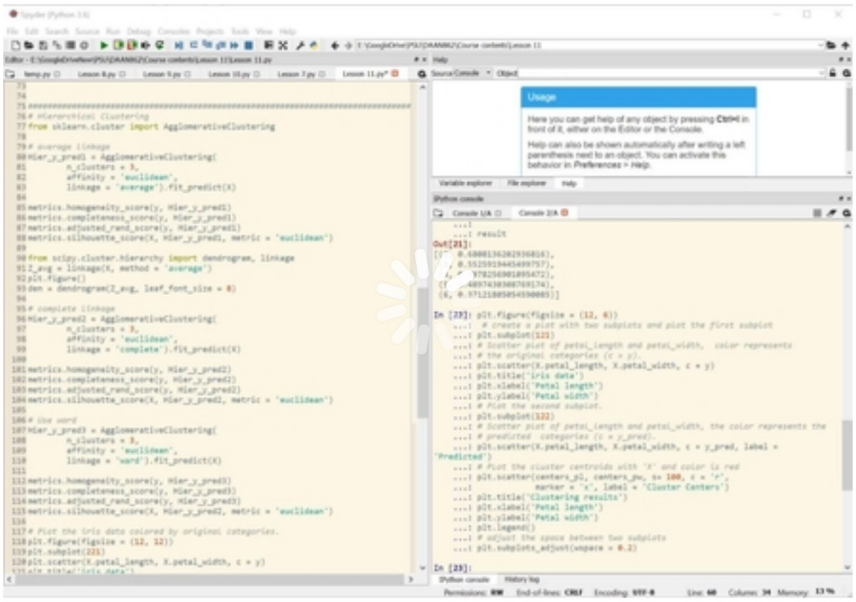


Fig 11.15 (B) (click to enlarge)



Transcript

In this video, I will introduce how to use Hierarchical clustering. Let's import Agglomerativeclustering from sklearn.clusters. For Hierarchical clustering, you can choose distance function type and linkage type. Here, we use n_clusters = 3, affinity = "Euclidean" and linkage = "average").

Hier_y_pred1 is the predicted y value for this model, you can evaluate with homogeneity score, completeness score, adjusted rand score, and silhouette score.

For plotting the dendrogram, you need to import dendrogram and linkage object form scipy.cluster.hierachy.

Z_avg is the linkage(X, method = average) will build hierarchical clustering based on this linkage type.

Dendrogram function will plot the dendrogram. You can enlarge it to see the x labels.

Next, we tried complete and Ward linkage. You can plot them similarly. Next, we plot the original iris data and the hierarchical clustering results with different linkage.

Three different linkages can predict the data for setosa clearly. The difference how accurate they can predict this part of the data.

Lesson 11.3: DB-Scan

DBSCAN (http://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html#sklearn.cluster.DBSCAN) is the model for DB-scan clusterings.

The important parameter for Hierarchical clustering methods are listed in Table 11.3.1:

Table 11.3.1: Main Parameters of DBSCAN

Parameters	Description
eps	The maximum distance between two samples for them to be considered as in the same neighborhood.
min samples	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.
metric	The metric to use when calculating distance between instances in a feature array.

The attributes for DBSCAN are listed in Table 11.3.2:

Table 11.3.2: Main Attributes of DBSCAN

Attributes	Description
Icore sample indicies	Indices of core samples.
components_	Copy of each core sample found by training.
labels	Cluster labels for each point in the dataset given to fit().

Let's follow the standard process to import DBSCAN object, which is trained with iris data and evaluated by the model performance. In Figure 11.16 we randomly pick up values for eps an min samples. Later, we will use grid search to find the best parameters:

```
In [38]: from sklearn.cluster import DBSCAN

In [39]: db_y_pred = DBSCAN(eps = 0.2, min_samples =5).fit_predict(X)

In [40]: metrics.homogeneity_score(y, db_y_pred)
Out[40]: 0.5763288083839111

In [41]: metrics.completeness_score(y, db_y_pred)
Out[41]: 0.8771705982981065

In [42]: metrics.adjusted_rand_score(y, db_y_pred)
Out[42]: 0.5535820784912957

In [43]: metrics.silhouette_score(X, db_y_pred, metric = 'euclidean')
Out[43]: 0.5552633662863133
```

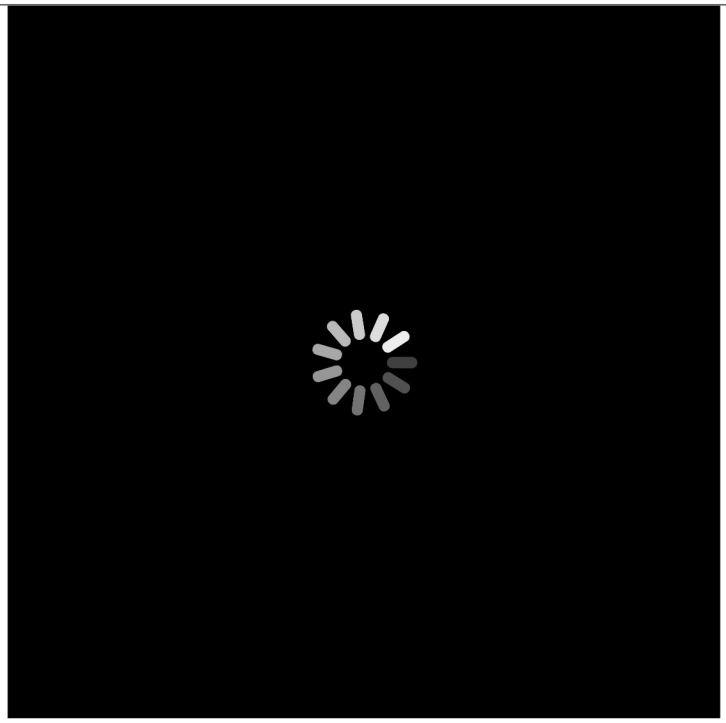
Fig 11.16 (click to enlarge)

Model Optimizaton:

The *eps and min_samples* are the most important parameters for DB-scan method. Figure 11.17 shows how to manually grid search the best parameters to separate the three species in iris data:

Fig 11.17

From the result, we can see that the highest Silhouette score can be achieved with eps = 0.2 and min_samples = 5 if we choose the number of clusters equals to 3. The fine search should be continued around eps = 0.2 and min_samples = 5 in order to maximize the results.



Transcript

In this video, I will introduce how to use DBSCAN clustering. You can import DBSCAN object from sklearn.cluster.

This is to initicate the DBSCAN model with eps or radius = 0.5, min_samples with the radium is 5. Db_y_pred is the prediction from this model.

You can evaluate it with clustering metrics.

Here we random choose values for eps and min_samples. These value should be optimized. Here we try few values.

Epses is 0.4, 0.6 and 0.8. min_samples is 5, 10, 15.

You can use grid search for it. Here we try to manually optimize it with nested for loop.

For v in epses for n in min_samples, you will build and evaluate the silhouette score with DBSCAN models with eps = v, and min_samples = n. And append (V, n, score) in to results.

From the results, we can see that 0.4, 15 gives the highest silhouette score.

Lesson 11 References (5 of 5)

Lesson 11 References

- http://scikit-learn.org/stable/index.html (http://scikit-learn.org/stable/index.html)
- Python Data Science Handbook, Jake VanderPlas
- https://jakevdp.github.io/PythonDataScienceHandbook/index.html (https://jakevdp.github.io/PythonDataScienceHandbook/index.html)
