

✓ 100 XP ▶

# Exercise - Train and evaluate advanced clustering models

12 minutes

Sandbox activated! Time remaining: **13 min**

You have used 1 of 10 sandboxes for today. More sandboxes will be available tomorrow.

✓ Runtime File Edit View

💬 Comments

▶▶ Run all ∨ 🔗 py38\_default ∨ ↻ 🔒

🗣️ We would love to hear your feedback on the notebooks experience! Please take a few minutes to [complete our survey](#).

✕

## Clustering - K-Means and Hierarchical

In the last notebook, we learned that data can be grouped into clusters, and we learned how to see whether our data may be compatible with such an analysis. In this notebook, we will perform this clustering automatically.

To get started, run the cell below to load our data.

▶ ∨

```
import pandas as pd

# load the training dataset
!wget https://raw.githubusercontent.com/MicrosoftDocs/mslearn-introduction-to-machine-learning/main/extra/seeds_csv.csv
data = pd.read_csv('seeds.csv')

# Display a random sample of 10 observations (just the features)
features = data[data.columns[0:6]]
features.sample(10)

from sklearn.preprocessing import MinMaxScaler
from sklearn.decomposition import PCA

# Normalize the numeric features so they're on the same scale
scaled_features = MinMaxScaler().fit_transform(features[data.columns[0:6]])

# Get two principal components
pca = PCA(n_components=2).fit(scaled_features)
features_2d = pca.transform(scaled_features)
```

MI 🔍 📄 🔒 ✂️ ... 🗑️

features\_2d[0:10]

[19]

✓ 1 sec

```
... --2024-02-12 23:21:33-- https://raw.githubusercontent.com/MicrosoftDocs/mslearn-
introduction-to-machine-learning/main/Data/ml-basics/seeds.csv
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133,
185.199.111.133, 185.199.110.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:443...
connected.
HTTP request sent, awaiting response... 200 OK
Length: 9386 (9.2K) [text/plain]
Saving to: 'seeds.csv.3'
```

```
seeds.csv.3          100%[=====>]    9.17K  --.-KB/s    in 0s
```

```
2024-02-12 23:21:33 (66.3 MB/s) - 'seeds.csv.3' saved [9386/9386]
```

```
array([[ 0.11883593, -0.09382469],
       [ 0.0696878 , -0.31077233],
       [-0.03499184, -0.37044705],
       [-0.06582089, -0.36365235],
       [ 0.32594892, -0.37695797],
       [-0.02455447, -0.31060184],
       [-0.00769646, -0.07594931],
       [-0.05646955, -0.26696284],
       [ 0.38196305, -0.05149471],
       [ 0.35701044, -0.17697998]])
```

## K-Means Clustering

The algorithm we'll use to create our test clusters is *K-Means*. This is a commonly used clustering algorithm that separates a dataset into  $K$  clusters of equal variance. The number of clusters,  $K$ , is user-defined. The algorithm has the following steps:

1. A set of  $K$  centroids are randomly chosen.
2. Clusters are formed by assigning the data points to their closest centroid.
3. The mean of each cluster is computed and the centroid is moved to the mean.
4. Steps 2 and 3 are repeated until a stopping criteria is met. Typically, the algorithm terminates when each new iteration results in negligible movement of centroids and the clusters become static.
5. When the clusters stop changing, the algorithm has *converged*, defining the locations of the clusters. Note that the random starting point for the centroids means that re-running the algorithm could result in slightly different clusters, so training usually involves multiple iterations, re-initializing the centroids each time, and the model with the best WCSS (*within cluster sum of squares*) is selected.

```
from sklearn.cluster import KMeans
```

```
# Create a model based on 3 centroids
model = KMeans(n_clusters=3, init='k-means++', n_init=100, max_iter=1000)
# Fit to the data and predict the cluster assignments for each data point
km_clusters = model.fit_predict(features.values)
# View the cluster assignments
```

[20]

```
km_clusters
✓ 1 sec
self._make_module_from_path(filepath)
File "/anaconda/envs/py38_default/lib/python3.8/site-packages/threadpoolctl.py", line
515, in _make_module_from_path
    module = module_class(filepath, prefix, user_api, internal_api)
File "/anaconda/envs/py38_default/lib/python3.8/site-packages/threadpoolctl.py", line
606, in __init__
    self.version = self.get_version()
File "/anaconda/envs/py38_default/lib/python3.8/site-packages/threadpoolctl.py", line
646, in get_version
    config = get_config().split()
AttributeError: 'NoneType' object has no attribute 'split'
Exception ignored on calling ctypes callback function: <function
_ThreadpoolInfo._find_modules_with_dl_iterate_phdr.<locals>.match_module_callback at
0x7fccac0be3a0>
Traceback (most recent call last):
  File "/anaconda/envs/py38_default/lib/python3.8/site-packages/threadpoolctl.py", line
400, in match_module_callback
    self._make_module_from_path(filepath)
  File "/anaconda/envs/py38_default/lib/python3.8/site-packages/threadpoolctl.py", line
515, in _make_module_from_path
    module = module_class(filepath, prefix, user_api, internal_api)
  File "/anaconda/envs/py38_default/lib/python3.8/site-packages/threadpoolctl.py", line
606, in __init__
    self.version = self.get_version()
  File "/anaconda/envs/py38_default/lib/python3.8/site-packages/threadpoolctl.py", line
646, in get_version
    config = get_config().split()
AttributeError: 'NoneType' object has no attribute 'split'
Exception ignored on calling ctypes callback function: <function
_ThreadpoolInfo._find_modules_with_dl_iterate_phdr.<locals>.match_module_callback at
0x7fccac0be3a0>
Traceback (most recent call last):
  File "/anaconda/envs/py38_default/lib/python3.8/site-packages/threadpoolctl.py", line
400, in match_module_callback
    self._make_module_from_path(filepath)
  File "/anaconda/envs/py38_default/lib/python3.8/site-packages/threadpoolctl.py", ...
```

[Show all \(87.1 kB\)](#)

```
array([2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 2, 0, 2, 2,
       2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 0, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 2, 2,
       2, 2, 2, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2, 1, 1, 1, 1, 1, 1, 1,
       2, 2, 2, 2, 1, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int32)
```

Now, let's see those cluster assignments with the two-dimensional data points.

```

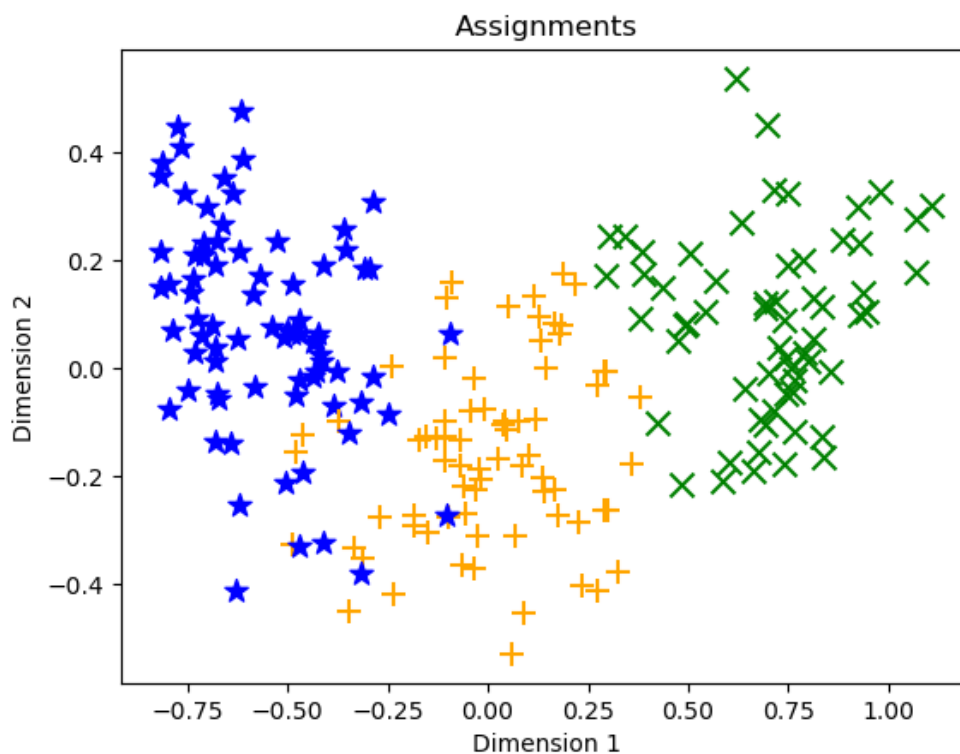
import matplotlib.pyplot as plt
%matplotlib inline

def plot_clusters(samples, clusters):
    col_dic = {0:'blue',1:'green',2:'orange'}
    mrk_dic = {0:'*',1:'x',2:'+'}
    colors = [col_dic[x] for x in clusters]
    markers = [mrk_dic[x] for x in clusters]
    for sample in range(len(clusters)):
        plt.scatter(samples[sample][0], samples[sample][1], color = colors[sample], mark
    plt.xlabel('Dimension 1')
    plt.ylabel('Dimension 2')
    plt.title('Assignments')
    plt.show()

plot_clusters(features_2d, km_clusters)

```

[21] ✓ 2 sec



The data should be separated into three distinct clusters. If not, rerun the previous two steps.

So what's the practical use of clustering? In some cases, you'll have data that you need to group into distinct clusters without knowing how many clusters there are or what they indicate. For example, a marketing organization might want to separate customers into distinct segments, and then investigate how those segments exhibit different purchasing behaviors.

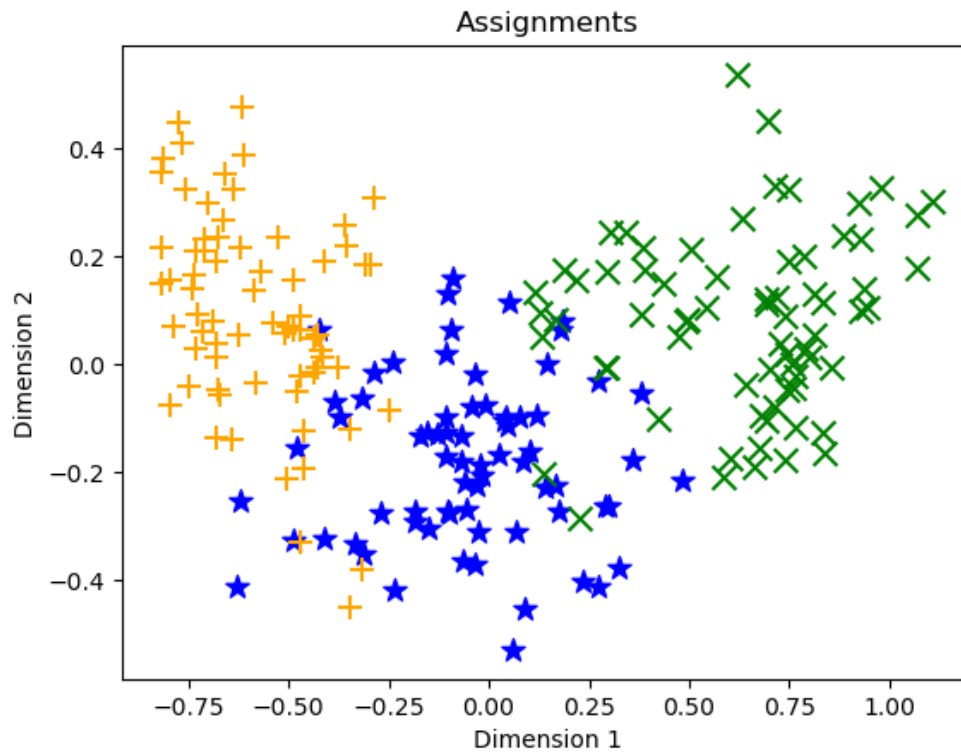
Sometimes, clustering is used as an initial step towards creating a classification model. You start by identifying distinct groups of data points, and then assign class labels to those clusters. You can then use this labelled data to train a classification model.

In the case of the seeds data, the different species of seed are already known and encoded as 0 (*Kama*), 1 (*Rosa*), or 2 (*Canadian*), so we can use these identifiers to compare the species classifications to the clusters identified by our unsupervised algorithm.

```
seed_species = data[data.columns[7]]  
plot_clusters(features_2d, seed_species.values)
```

[22]

✓ 2 sec



There may be some differences between the cluster assignments and class labels, but the K-Means model should have done a reasonable job of clustering the observations so that seeds of the same species are generally in the same cluster.

# Hierarchical Clustering

Hierarchical clustering methods make fewer distributional assumptions when compared to K-Means methods. However, K-Means methods are generally more scalable, sometimes very much so.

Hierarchical clustering creates clusters by using either a *divisive* method or an *agglomerative* method. The divisive method is a "top down" approach starting with the entire dataset and then finding partitions in a stepwise manner. Agglomerative clustering is a "bottom up" approach. In this lab you will work with agglomerative clustering which works as follows:

```
from sklearn.cluster import AgglomerativeClustering

agg_model = AgglomerativeClustering(n_clusters=3)
agg_clusters = agg_model.fit_predict(features.values)
agg_clusters
```

[23] ✓ <1 sec

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0,
       0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0,
       0, 0, 0, 2, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 0, 2, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

Similarity can be quite useful when working with data such as images or text documents.

## Agglomerative Clustering

So what do the agglomerative cluster assignments look like?

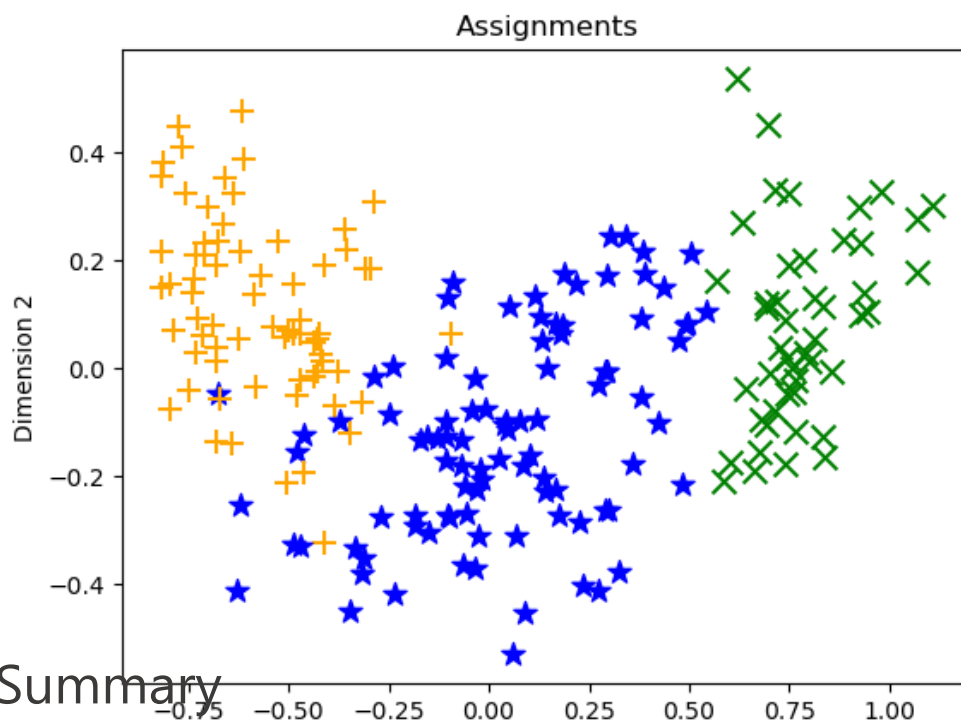
```
import matplotlib.pyplot as plt

%matplotlib inline

def plot_clusters(samples, clusters):
    col_dic = {0:'blue',1:'green',2:'orange'}
    mrk_dic = {0:'*',1:'x',2:'+'}
    colors = [col_dic[x] for x in clusters]
    markers = [mrk_dic[x] for x in clusters]
    for sample in range(len(samples)):
        plt.scatter(samples[sample][0], samples[sample][1], color = colors[sample], marker = markers[sample])
    plt.xlabel('Dimension 1')
    plt.ylabel('Dimension 2')
    plt.title('Assignments')
    plt.show()

plot_clusters(features_2d, agg_clusters)
```

[24] ✓ 1 sec



## Summary

Here we practiced using K-Means and hierarchical clustering. This unsupervised learning has the ability to take unlabelled data and identify which data points are similar to others.

## Further Reading

To learn more about clustering with scikit-learn, see the [scikit-learn documentation](#).

learn-notebooks-0f41ae00-e57d-4d65-a934-d4804373aee6 Compute connected Editing Kernel idle py38\_default

## Next unit: Knowledge check

Continue >