CSE 483: Artificial Intelligence Lab
Lab Activity 3: K-Means Clustering using Scikit-learn

Dr Muhammad Abul Hasan[*]

November 20, 2021

1 Objective

The k-means clustering method is an unsupervised machine learning technique used to identify clusters of data objects in a dataset. There are many different types of clustering methods, but k-means is one of the oldest and most approachable. These traits make implementing k-means clustering in Python reasonably straightforward, even for novice programmers and data scientists.

In this lab activity, you will learn:

- What k-means clustering is
- When to use k-means clustering to analyze your data
- How to implement k-means clustering in Python with scikit-learn
- How to select a meaningful number of clusters

2 What Is Clustering?

Clustering is a set of techniques used to partition data into groups, or clusters. Clusters are loosely defined as groups of data objects that are more similar to other objects in their cluster than they are to data objects in other clusters. In practice, clustering helps identify two qualities of data:

- Meaningfulness
- Usefulness

Meaningful clusters expand domain knowledge. For example, in the medical field, researchers applied clustering to gene expression experiments. The clustering results identified groups of patients who respond differently to medical treatments.

Useful clusters, on the other hand, serve as an intermediate step in a data pipeline. For example, businesses use clustering for customer segmentation. The clustering results segment customers into groups with similar purchase histories, which businesses can then use to create targeted advertising campaigns.

There are many other applications of clustering, such as document clustering and social network analysis. These applications are relevant in nearly every industry, making clustering a valuable skill for professionals working with data in any field.

[*]Assistant Professor, Deptartment of CSE, ULAB ⁘ ✉ muhammad.hasan@ulab.edu.bd

3 Overview of Clustering Techniques

You can perform clustering using many different approaches–so many, in fact, that there are entire categories of clustering algorithms. Each of these categories has its own unique strengths and weaknesses. This means that certain clustering algorithms will result in more natural cluster assignments depending on the input data.

Selecting an appropriate clustering algorithm for your dataset is often difficult due to the number of choices available. Some important factors that affect this decision include the characteristics of the clusters, the features of the dataset, the number of outliers, and the number of data objects.

You will explore how these factors help determine which approach is most appropriate by looking at three popular categories of clustering algorithms:

1. Partitional clustering
2. Hierarchical clustering
3. Density-based clustering

It is worth reviewing these categories at a high level before jumping right into k-means. You will learn the strengths and weaknesses of each category to provide context for how k-means fits into the landscape of clustering algorithms.

3.1 Partitional Clustering

Partitional clustering divides data objects into nonoverlapping groups. In other words, no object can be a member of more than one cluster, and every cluster must have at least one object.

These techniques require the user to specify the number of clusters, indicated by the variable k. Many partitional clustering algorithms work through an iterative process to assign subsets of data points into k clusters. Two examples of partitional clustering algorithms are k-means and k-medoids.

These algorithms are both nondeterministic, meaning they could produce different results from two separate runs even if the runs were based on the same input.

Partitional clustering methods have several strengths:

- They work well when clusters have a spherical shape.
- Theyre scalable with respect to algorithm complexity.

They also have several weaknesses:

- Theyre not well suited for clusters with complex shapes and different sizes.
- They break down when used with clusters of different densities.

3.2 Hierarchical Clustering

Hierarchical clustering determines cluster assignments by building a hierarchy. This is implemented by either a bottom-up or a top-down approach:

Agglomerative clustering is the bottom-up approach. It merges the two points that are the most similar until all points have been merged into a single cluster.

Divisive clustering is the top-down approach. It starts with all points as one cluster and splits the least similar clusters at each step until only single data points remain.

These methods produce a tree-based hierarchy of points called a dendrogram. Similar to partitional clustering, in hierarchical clustering the number of clusters (k) is often predetermined by the user. Clusters are assigned by cutting the dendrogram at a specified depth that results in k groups of smaller dendrograms.

Unlike many partitional clustering techniques, hierarchical clustering is a deterministic process, meaning cluster assignments wont change when you run an algorithm twice on the same input data.

The strengths of hierarchical clustering methods include the following:

- They often reveal the finer details about the relationships between data objects.
- They provide an interpretable dendrogram.

The weaknesses of hierarchical clustering methods include the following:

- Theyre computationally expensive with respect to algorithm complexity.
- Theyre sensitive to noise and outliers.

3.3 Density-Based Clustering

Density-based clustering determines cluster assignments based on the density of data points in a region. Clusters are assigned where there are high densities of data points separated by low-density regions.

Unlike the other clustering categories, this approach doesnt require the user to specify the number of clusters. Instead, there is a distance-based parameter that acts as a tunable threshold. This threshold determines how close points must be to be considered a cluster member.

Examples of density-based clustering algorithms include Density-Based Spatial Clustering of Applications with Noise, or DBSCAN, and Ordering Points To Identify the Clustering Structure, or OPTICS.

The strengths of density-based clustering methods include the following:

- hey excel at identifying clusters of nonspherical shapes.
- Theyre resistant to outliers.

The weaknesses of density-based clustering methods include the following:

- They arent well suited for clustering in high-dimensional spaces.
- They have trouble identifying clusters of varying densities.

4 Mathematics behind K-Mean Clustering algorithm

K-Means is one of the simplest unsupervised clustering algorithm which is used to cluster our data into K number of clusters. The algorithm iteratively assigns the data points to one of the K clusters based on how near the point is to the cluster centroid.

1. K number of cluster centroids
2. Data points classified into the clusters

4.1 Outline of the algorithm

Assuming we have input data points $x_1, x_2, x_3, \ldots, x_n$ and value of K (the number of clusters needed). We follow the below procedure:

1. Pick K points as the initial centroids from the dataset, either randomly or the first K.
2. Find the Euclidean distance of each point in the dataset with the identified K points (cluster centroids).
3. Assign each data point to the closest centroid using the distance found in the previous step.
4. Find the new centroid by taking the average of the points in each cluster group.
5. Repeat 2 to 4 for a fixed number of iteration or till the centroids dont change.

4.2 Euclidean Distance between two points in space

We use euclidean distance to measure the distance between two points in space which is expressed as follows:

$$d(\mathbf{p}, \mathbf{q}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2}.$$

If $\mathbf{p} = (p_1, p_2)$ and $\mathbf{q} = (q_1, q_2)$ then the distance function can be implemented in Python as follows:

```python
def euclidean_distance(pt1, pt2):
    return math.sqrt((pt1[0]-pt2[0])**2+(pt1[1]-pt2[1])**2)
```

4.3 Assigning each point to the nearest cluster

If each cluster centroid is denoted by c_i, then each data point x is assigned to a cluster based on

$$\underset{c_i \in C}{\arg\min}\, dist(c_i, x)$$

here $dist()$ is the euclidean distance. We implement this in Python as follows:

```python
#find the distance between the points and the centroids
for point in data:
    distances = []
    for index in self.centroids:
        distances.append(self.distance(point,self.centroids[index]))

        #find which cluster the datapoint belongs to by finding
        #the minimum ex: if distances are 2.03,1.04,5.6,1.05
        #then point belongs to cluster 1 (zero index)
        cluster_index = distances.index(min(distances))
        self.classes[cluster_index].append(point)
```

4.4 Finding the new centroid from the clustered group of points

$$c_i = \frac{1}{|S_i|} \sum_{x_i \in S_i} x_i.$$

S_i is the set of all points assigned to the ith cluster. The following function can be used to perform this task.

```
#find new centroid by taking the centroid of the points in the
#cluster class
for index in self.classes:
    self.centroids[index]=np.average(self.classes[index], axis = 0)
```

5 K-Means on Iris Dataset

Now K-Means algorithm will be applied on Iris dataset to classify our 3 classes of flowers, Iris setosa, Iris versicolor, Iris virginica (our classess) using the features collected from flowers sepal-length, sepal-width, petal-length and petal-width.

Importing libraries:

```
import pandas as pd
import numpy as np
from sklearn import datasets
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
import sklearn.metrics as sm
%matplotlib inline
```

Loading dataset:

```
iris = datasets.load_iris()
```

Prining the first 5 data:

```
print (iris.data[0:5])
```

Output:

```
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]]
```

Prining traget names:

```
print (iris.target_names)
```

Output:

```
['setosa' 'versicolor' 'virginica']
```

Prining data labels:

```
print (iris.target)
```

Output:

```
[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 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 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 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

Pandas is a popular python library useful for data scientiest. Pandas data structure *DataFrame* is a way to represent and work with tabular data. It can be seen as a table that organizes data into rows and columns, making it a two-dimensional data structure. A DataFrame can be either created from scratch or you can use other data structures like **Numpy** arrays.

Converting iris data into Pandas DataFrames:

```
# assigning column labels SL, SW, PL, PW for SL: Sepal Length,
# SW: Sepal Width, PL: Petal Length and PW: Petal Width.
x = pd.DataFrame(iris.data, columns=['SL','SW','PL','PW'])
y = pd.DataFrame(iris.target, columns=['Target'])
```

Prining Data in data frames:

```
x.head()
```

Output:

	SL	SW	PL	PW
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

Prining target:

```
y.head()
```

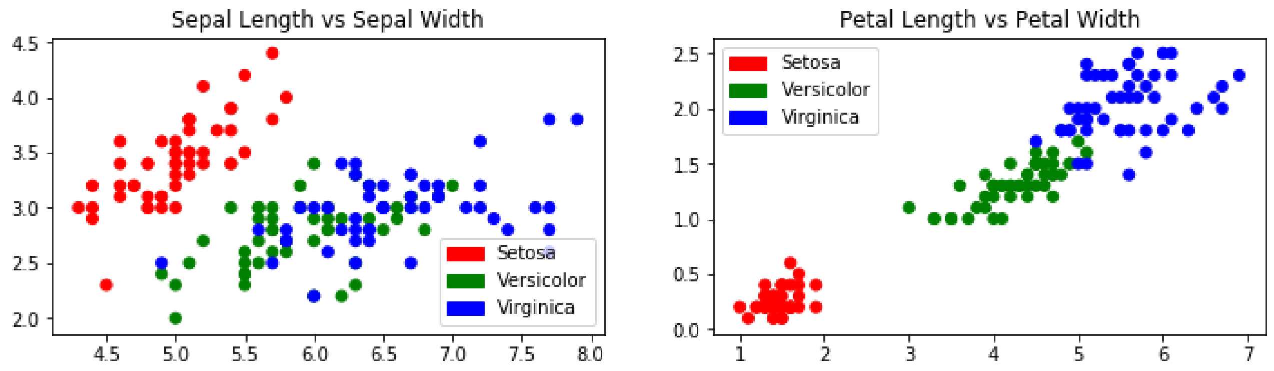
Output:

	Target
0	0
1	0
2	0
3	0
4	0

Now, Lets visualize the data using scatter plots. It would be the best if we could generate a 4D scatter plot. Since it is not possible, we can visualize the relationship between any two features using a 2D scatter plot. In the following, we generated two diagrams representing Sepal Length vs Sepal Width, and Petal Length vs Petal Width respectively.

```
plt.figure(figsize=(12,3))
colors = np.array(['red', 'green', 'blue'])
iris_targets_legend = np.array(iris.target_names)
red_patch = mpatches.Patch(color='red', label='Setosa')
green_patch = mpatches.Patch(color='green', label='Versicolor')
blue_patch = mpatches.Patch(color='blue', label='Virginica')

plt.subplot(1, 2, 1)
plt.scatter(x['SL'], x['SW'], c=colors[y['Target']])
plt.title('Sepal␣Length␣vs␣Sepal␣Width')
plt.legend(handles=[red_patch, green_patch, blue_patch])

plt.subplot(1,2,2)
plt.scatter(x['PL'], x['PW'], c= colors[y['Target']])
plt.title('Petal␣Length␣vs␣Petal␣Width')
plt.legend(handles=[red_patch, green_patch, blue_patch])
```

Output:



Next, let us create a KMeans model to cluster the data. We use the entire dataset for training. Please note that, we have not split the dataset into training and test set here. However, if we want to do that, how it can be done?

```
iris_k_mean_model = KMeans(n_clusters=3)  # we set K = 3
iris_k_mean_model.fit(x) # trained using the entire dataset
```

Output:

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=3, n_init=10, n_jobs=None, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=0)
```

We test the model performance using the entire dataset that was used for training the model. Although, this is not an standard parctice, we are doing it to gain an insight about the KMeans algorithm.

```python
print(iris_k_mean_model.predict(x))
```

Output:

```
[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 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 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 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 2 2 2 2 0 2 2 2 2
 2 2 0 0 2 2 2 2 0 2 0 2 0 2 2 0 0 2 2 2 2 2 0 2 2 2 2 0 2 2 2 0 2 2 2 0 2
 2 0]
```

The computed cluster centers can be examined using the following piece of code. Please have a careful look on the data and identify which cluster center belongs which class of iris flower. UNDERSTANDING THIS IS IMPORTANT.

```python
print (iris_k_mean_model.cluster_centers_)
```

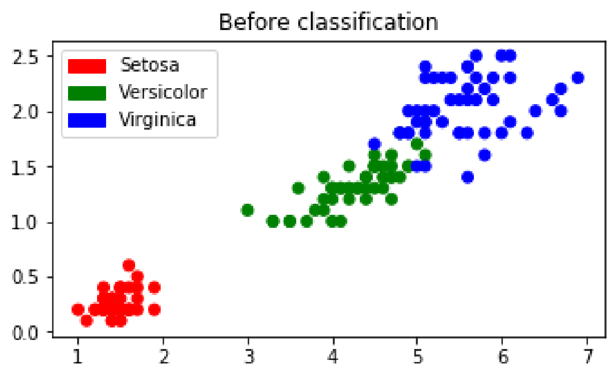
Output:

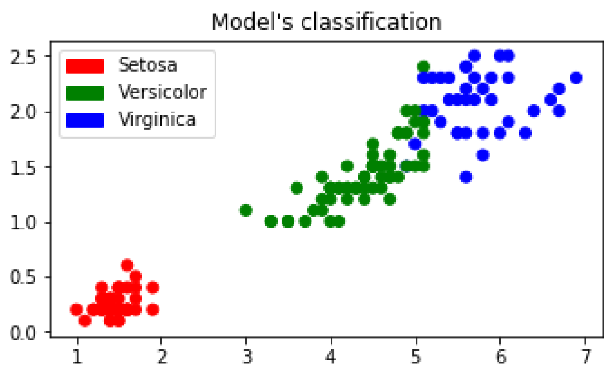
```
[[5.9016129  2.7483871  4.39354839 1.43387097]
 [5.006      3.428      1.462      0.246     ]
 [6.85       3.07368421 5.74210526 2.07105263]]
```

Now, we are interested to check the classification performance of the trained model. The following piece of code visualizes the data before classification and the classification made by the model.

```python
plt.figure(figsize=(12,3))

colors = np.array(['red', 'green', 'blue'])

predictedY=np.choose(iris_k_mean_model.labels_,[1,0,2]).astype(np.int64)

plt.subplot(1, 2, 1)
plt.scatter(x['PL'], x['PW'], c=colors[y['Target']])
plt.title('Before classification')
plt.legend(handles=[red_patch, green_patch, blue_patch])

plt.subplot(1, 2, 2)
plt.scatter(x['PL'], x['PW'], c=colors[predictedY])
plt.title("Model's classification")
plt.legend(handles=[red_patch, green_patch, blue_patch])
```

Output:





At this stage, we would like to know the accuracy of the trained model. We do it using the following piece of code.

```
sm.accuracy_score(predictedY, y['Target'])
```

Output:

```
0.8933333333333333
```

And finally, we generate the confusion matrix using the following piece of code. As it can be seen, 50 out of 50 setosa has been classified successfully. However, 2 versicolor and 14 virginica flowers have be failed to detect correctly by the trained model.

```
sm.confusion_matrix(predictedY, y['Target'])
```

Output:

```
array([[50,  0,  0],
       [ 0, 48, 14],
       [ 0,  2, 36]])
```

6 Lab Exercises

Please code yourself and write a report based on your findings:

1. Split the dataset into training and testing set and evaluate their performance.
2. Evaluate the model performance using Sepal Length vs Petal Width.

My Notes Date:________________