K Means implementation on Iris Dataset for creating clusters

By: Sanjeet Pal Singh

Introduction

In the project, we are using the K-Means algorithm on the Iris datasets to build the clusters of the different species. In the dataset of Iris, there are total of 150 data points (rows), and 5 features (columns), which includes petal length, petal width, sepal length, and sepal width along with species column. There are three species of the Iris in dataset. In the data pre-processing part, we will ignore and remove the species part before proceeding with further steps to create clusters of species out of the given data points in order to see the real power of the clustering algorithm.

There are many different algorithms available using which we can create clusters from datasets, but K-Means is the popular and easiest to implement comparatively to other clustering algorithms. The whole project is to create clusters out of the Iris dataset without using species column has been accomplished in the python language. Now lets begin with further process of building machine learning model to create clusters.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn import metrics
from sklearn.neighbors import KNeighborsClassifier

data = pd.read_csv('Iris.csv')
data = data.drop(['Species'], axis=1)
data.shape
```

```
(150, 4)
```

Libraries being used in the project

Data Preparation

Luckily the given datasets don't have any missing values which further we need to take care of these by filling up mean, mode or median or zeroes with the help

of sklearn imputer library. So first we are importing the datasets using the pandas library and further species column is being dropped with the help of the available methods in the pandas. The datasets after importing and droping a column look like this:

```
In [93]: data.head()
```

Out[93]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
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

```
In [94]: data.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 4 columns):
SepalLengthCm    150 non-null float64
SepalWidthCm     150 non-null float64
PetalLengthCm    150 non-null float64
PetalWidthCm     150 non-null float64
dtypes: float64(4)
memory usage: 4.8 KB
```

Dataset after removing species column

The fields such as sepal length, width and petal length & width are the float variables.

For implementing K-Means we don't need to create training and test sets as the goal of clustering using this algorithm is to decrease the total separation across the k clusters that the algorithm will form. As in our case we simply just want to separate the species as cluster in our data. Also lets say if we have implemented the k means on the prior data, just after that lifecycle of the model will not end. There could be the scenarios where we have unseen points in of the identified clusters which can be read as test data. So it means that we already trained the model on the basis of some data which would then be taken as training data.

Visualization of the Scatter Plot Before K-Means

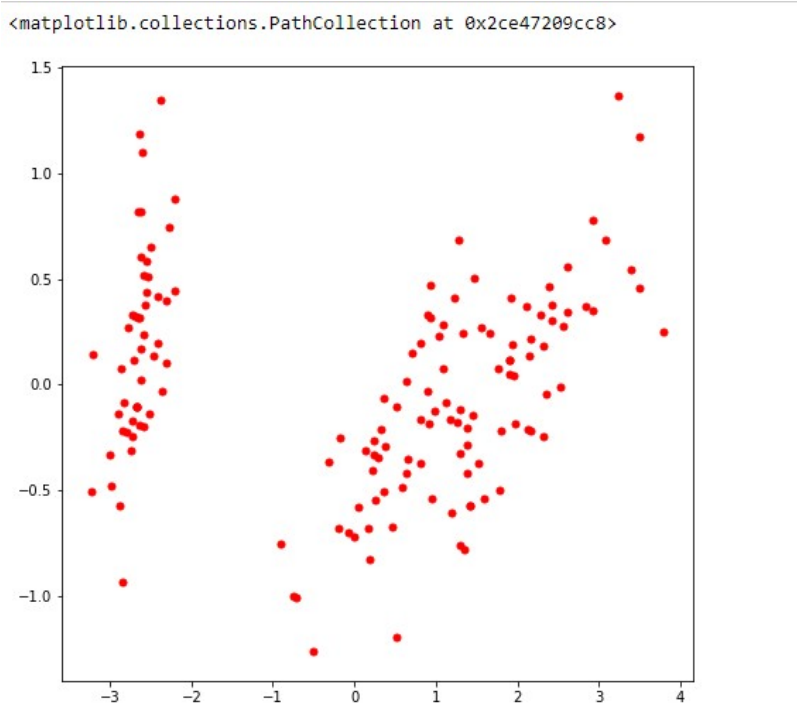
In order to see how the data is scattered over the scatter plot graph we will use the matplotlib library to plot the scatter plot. But as the number of dimensions in

this case is 4 (which is more than 2) we need to do the dimension reduction to plot the graph. In order to do the dimensional reduction we will use the PCA or Principal Component Analysis. Using sklearn library we are importing the PCA method from the it.

```
from sklearn.decomposition import PCA
pca = PCA(n_components=2).fit(data)
data = pca.transform(data)
plt.figure(figsize=(8,8))
plt.scatter(data[:,0], data[:,1], c='red', s=25)
```

Principal Component Analysis to plot the scatter plot graph

Here inside the scatter method of the matplotlib library we are passing the data points x and y axis , color as red, size of the dots as 25.



Scatter Plot before K-means

Implementing K-Means

We will use the scikit-learn library to import the k-means method to perform clustering operation. The kmeans method take following parameters as argument in order to work .

n_clusters: Number of Clusters – the by-default value is 8

init: It takes the method for initialization to select the intial clusters. The default is k-means++ or it could be random

n_init: number of time the algorithm will run, default value is 10

max_iter: number of iterations for single run, default 300

random_state: default is None, It find the random number generation to initialize the centroid.

There are other parameters too, but we are explaining which we are using in our project.

In order to implement the K-Means we need to know the value of k or number clusters we want to find out or create in the Iris dataset.

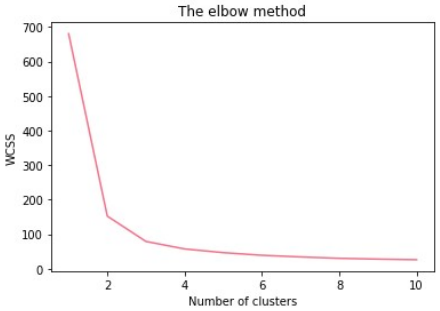
Here k is the important argument or parameter that need to be predicted correctly in order to create correct number of clusters. So there are several methods to calculate or guess the value of k using some techniques. There are two techniques: **Elbow-method and The Silhoutte Method**

In our project we are going to use the Elbow-Method

Elbow-Method

It is very popular method to determine the correct number of cluster, in this method we calculate the WCSS i.e. Within Cluster Sum of Square Errors for different values of k, normally we do the k means using by iterating the value of k within a guessed range. In this case we will iterate the value from 1 to 11. And further we will obtain the **plot between the WSS and k , the place where the elbow will form in front of values of k.** Simply we just need to look into the values of k in front of the elbow curve to get the value of k.

```
: from sklearn.cluster import KMeans
wcss = []

for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', max_iter = 300, n_init = 10, random_state = 0)
    kmeans.fit(data)
    wcss.append(kmeans.inertia_)

#Plotting the results onto a line graph, allowing us to observe 'The elbow'
plt.plot(range(1, 11), wcss)
plt.title('The elbow method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS') #within cluster sum of squares
plt.show()
```



Elbow method implementation

We can observe in the above graph that the curve is from k=2 to 4, so the value is in between i.e. k=3 which is correct and also we know this number in the beginning that number of species were three.

So now lets implement the K-Means using the correct value of k in order to get our scatter plot graph to see clusters distributed over the graph.

```
kmeans = KMeans(n_clusters = 3, init = 'k-means++', max_iter = 300, n_init = 10, random_state = 0)
y_kmeans = kmeans.fit_predict(data)
print(y_kmeans)
```

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

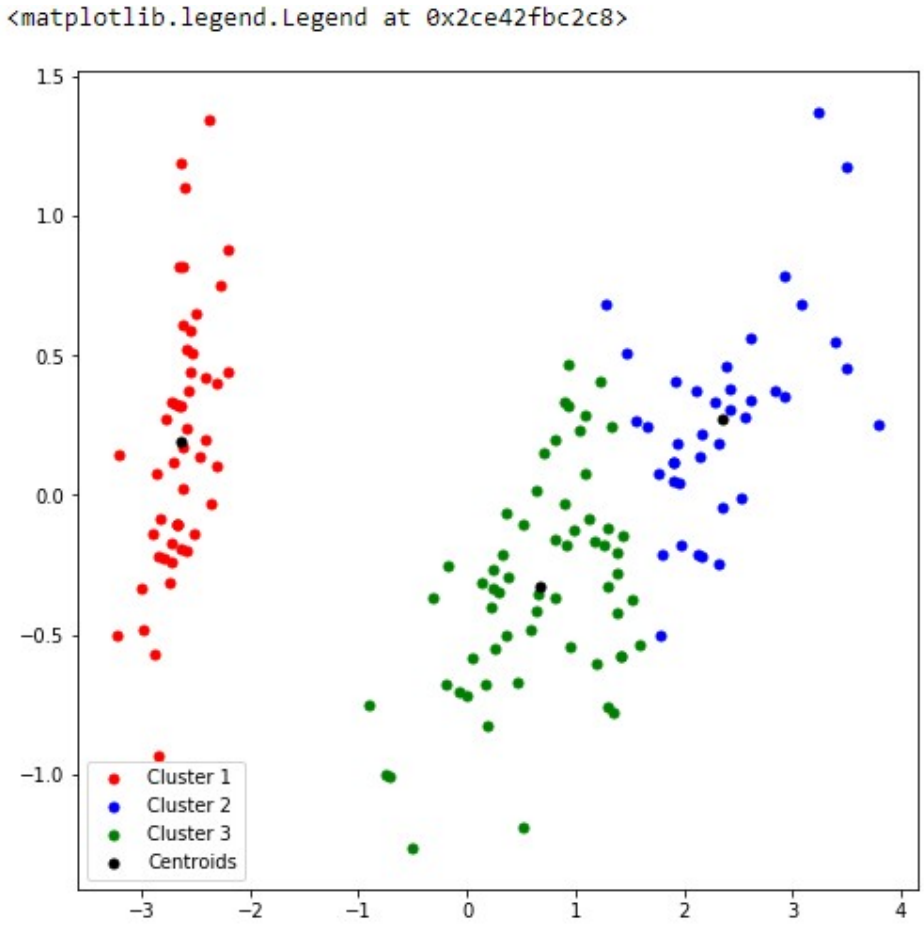
Applying the k-means

```
x = data
plt.figure(figsize=(8,8))
plt.scatter(x[y_kmeans == 0, 0], x[y_kmeans == 0, 1], s = 25, c = 'red', label = 'Cluster 1')
plt.scatter(x[y_kmeans == 1, 0], x[y_kmeans == 1, 1], s = 25, c = 'blue', label = 'Cluster 2')
plt.scatter(x[y_kmeans == 2, 0], x[y_kmeans == 2, 1], s = 25, c = 'green', label = 'Cluster 3')

#Plotting the centroids of the clusters
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:,1], s = 25, c = 'black', label = 'Centroids')

plt.legend()
```

Visualizing the clusters

Results

Our correctly plotted scatter plot graph showing the clusters here each cluster represent the one of the species. Also we are able to see this graph because we are passing the values after PCA which we did earlier before k-means.



Scatter Plot Graph after applying K-means algorithm to create clusters