**Course: Introduction to Data Science (DS2006) - Laboratory 16**

Hey Everyone, just a friendly reminder that this lab does not need to be submitted. It was done to address the request of some students in class about running kmeans with python. Hope you find it useful!

**Create a file named** [**kmeans.py**](http://kmeans.py)**. After that the first thing we are going to do is create synthetic data points in order to be able to visualize them (similarly to how we did in the Lecture).**

**To make the synthetic data we will use the functionality named make\_blobs from scikit-learn. Figure 1 shows an example of how to use it.**

# Import the Make\_blobs which we will use to

# create synthetic data points:

from sklearn.datasets import make\_blobs

#Create the synthetic data:

features, true\_centers = make\_blobs(

# Define the number of instances:

n\_samples=300,

# Define the number of features

# (we will use 2 to make it easy to visualize):

n\_features=2,

# Define the number of "real" clusters:

centers=4,

# Define the standard deviation (spread) of each cluster:

cluster\_std=5.0,

# Define the random state to allow reproducibility:

random\_state=10

)

Figure 1 - Code Snippet for using make\_blobs to create synthetic data.

**Task 1:** Add the code shown in Figure 1 to your [kmeans.py](http://kmeans.py) file, then create a pandas data frame with the code shown in Figure 2, print the first 10 instances and paste the result here:

#Creating a data frame for the synthetic data:

df = pd.DataFrame(features, columns=["x1", "x2"])

Figure 2 - Code Snippet

**Task 2:** Now we are going to use the k-means algorithm to cluster the data. Figure 3 shows an example of how this can be done using the implementation available in scikit-learning.

# Import the Scikit-Learn Kmeans Algorithm Implementation:

from sklearn.cluster import KMeans

#Previous code omitted.

# Create the Object for the k-Means Clustering Algorithm:

kmeans = KMeans(n\_clusters=4, random\_state=10)

# Run the algorithm on the data passed as parameter:

kmeans.fit(features)

# Obtain the cluster predictions and their centroids:

cluster\_predictions = kmeans.labels\_

centroids = kmeans.cluster\_centers\_

Figure 3 - Code Snippet

**Task 3:** Now let us take a look at the final calculated centroid by printing the **centroids** variable. Paste the result here:

**Task 4:** You can also print the final assigned label for each instance by printing the results stored in the **cluster\_predictions** variable. Paste the result here:

**Task 5:** Since we are dealing with data with 2 dimensions, it would be nice to be able to visualize things in a 2D Plot. We can do that by using functionalities from the matplot library. Figure 4 shows how to plot the original generated synthetic data. Add this to your code and paste the generated 2D plot here:

import matplotlib.pyplot as plt

#Previous code omitted.

# Visualize the created synthetic data:

plt.figure(figsize=(7, 5))

plt.scatter(features[:, 0], features[:, 1], s=50)

plt.title("Raw Data")

plt.xlabel("Feature 1")

plt.ylabel("Feature 2")

plt.show()

Figure 4 - Code Snippet

**Task 4:** We can also use matplot lib to visualize the results from kMeans clustering. Refactor your code to also use the code from figure 5 and past the generated 2D plot here:

# Visualize the result of kmean on the created synthetic data:

plt.figure(figsize=(7, 5))

plt.scatter(features[:, 0], features[:, 1], c=cluster\_predictions, cmap='viridis', s=50)

plt.scatter(centroids[:, 0], centroids[:, 1], c='red', marker='X', s=200, label='Centroids')

plt.title("K-Means Clustering Results")

plt.xlabel("Feature 1")

plt.ylabel("Feature 2")

plt.legend()

plt.show()

Figure 5 - Code Snippet

**Task 5:** Change the number of clusters in kmeans to three different values, run the algorithms and plot the results. Paste the results here informing the value of k used: