COMP809 – Data Mining and Machine Learning

Lab 7 – Clustering with Kmeans

This lab will cover Kmeans clustering.

Study the code provided below and run it in Python. The code uses SSE (it is referred to as distortion in the code) and the cluster silhouette measure for evaluation.

The code also compares two different cluster configurations, one with K=3 and K=2. You will observe that K=3 produces a better cluster configuration.

Extend the code to use the Iris dataset. Once it is working use PCA as a pre processing step prior to applying Kmeans.

```
# KMeans module from sklean is used to perform the clustering.
#Go to https://scikit-
learn.org/stable/modules/generated/sklearn.cluster.KMeans.html
for more details.

from IPython.display import Image
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
import numpy as np
from matplotlib import cm
from sklearn.metrics import silhouette_samples

#Create a sample data set using make_blobs. This particular
dataset has 2 features and 3 clusters.

X, y = make_blobs(n_samples=150,
                  n_features=2,
                  centers=3,
                  cluster_std=0.5,
                  shuffle=True,
                  random_state=0)

#Display the datset
plt.scatter(X[:, 0], X[:, 1], c='white', marker='o',
edgecolor='black', s=50)
plt.grid()
plt.tight_layout()

#Customize the path to save images
plt.savefig('P:/COMP809/11_01.png', dpi=300)
plt.show()

#Apply k-means clustering with 3 centroids
km = KMeans(n_clusters=3,
            init='random',
            n_init=10,
            max_iter=300,
            tol=1e-04,
            random_state=0)

#We set n_init=10 to run the k-means clustering algorithms 10
times independently with different random centroids to choose
the final model as the one with the lowest SSE.
```

```
#Predicting cluster labels
y_km = km.fit_predict(X)

#Visualize the clusters identified(using y_km)together with
cluster labels.
plt.scatter(X[y_km == 0, 0],
            X[y_km == 0, 1],
            s=50, c='lightgreen',
            marker='s', edgecolor='black',
            label='cluster 1')

plt.scatter(X[y_km == 1, 0],
            X[y_km == 1, 1],
            s=50, c='orange',
            marker='o', edgecolor='black',
            label='cluster 2')

plt.scatter(X[y_km == 2, 0],
            X[y_km == 2, 1],
            s=50, c='lightblue',
            marker='v', edgecolor='black',
            label='cluster 3')

plt.scatter(km.cluster_centers_[:, 0],
            km.cluster_centers_[:, 1],
            s=250, marker='*',
            c='red', edgecolor='black',
            label='centroids')

plt.legend(scatterpoints=1)
plt.grid()
plt.tight_layout()
plt.savefig('P:/COMP809/11_02.png', dpi=300)
plt.show()

#Calculating Distortion
print('Distortion: %.2f' % km.inertia_)
distortions = []
```

```
#Observing the behaviour of the distortion with the number of
clusters.
#Using elbow method to find optimum number of clusters
for i in range(1, 11):
    km = KMeans(n_clusters=i,
                init='k-means++',
                n_init=10,
                max_iter=300,
                random_state=0)
    km.fit(X)
    distortions.append(km.inertia_)

plt.plot(range(1, 11), distortions, marker='o')
plt.xlabel('Number of clusters')
plt.ylabel('Distortion')
plt.tight_layout()
plt.savefig('P:/COMP809/11_03.png', dpi=300)
plt.show()


#Quantifying the quality of clustering  via silhouette plots.
#k-means++ gives better clustering/performance than classic
approach(init='random'). Always recommended to use k-means++

km = KMeans(n_clusters=3,
            init='k-means++',
            n_init=10,
            max_iter=300,
            tol=1e-04,
            random_state=0)
y_km = km.fit_predict(X)

#Silhouette Measure/plots for cluster evaluation. Higher the
score better the clustering.

cluster_labels = np.unique(y_km)
n_clusters = cluster_labels.shape[0]
silhouette_vals = silhouette_samples(X, y_km,
metric='euclidean')
y_ax_lower, y_ax_upper = 0, 0
yticks = []
for i, c in enumerate(cluster_labels):
    c_silhouette_vals = silhouette_vals[y_km == c]
    c_silhouette_vals.sort()
    y_ax_upper += len(c_silhouette_vals)
    color = cm.jet(float(i) / n_clusters)
    plt.barh(range(y_ax_lower, y_ax_upper), c_silhouette_vals,
height=1.0,edgecolor='none', color=color)

    yticks.append((y_ax_lower + y_ax_upper) / 2.)
    y_ax_lower += len(c_silhouette_vals)

silhouette_avg = np.mean(silhouette_vals)
plt.axvline(silhouette_avg, color="red", linestyle="--")

plt.yticks(yticks, cluster_labels + 1)
plt.ylabel('Cluster')
plt.xlabel('Silhouette coefficient')

plt.tight_layout()
plt.savefig('P:/COMP809/11_04.png', dpi=300)
plt.show()

# Comparison to "bad" clustering:Same data samples are
clustered using 2 cluters.

km = KMeans(n_clusters=2,
            init='k-means++',
            n_init=10,
            max_iter=300,
            tol=1e-04,
            random_state=0)
y_km = km.fit_predict(X)

plt.scatter(X[y_km == 0, 0],
            X[y_km == 0, 1],
            s=50,
            c='lightgreen',
            edgecolor='black',
            marker='s',
            label='cluster 1')

plt.scatter(X[y_km == 1, 0],
            X[y_km == 1, 1],
            s=50,
            c='orange',
            edgecolor='black',
            marker='o',
            label='cluster 2')

plt.scatter(km.cluster_centers_[:, 0], km.cluster_centers_[:,
1], s=250, marker='*', c='red', label='centroids')
plt.legend()
plt.grid()
plt.tight_layout()
plt.savefig('P:/COMP809/11_05.png', dpi=300)
plt.show()

cluster_labels = np.unique(y_km)
n_clusters = cluster_labels.shape[0]
silhouette_vals = silhouette_samples(X, y_km,
metric='euclidean')
y_ax_lower, y_ax_upper = 0, 0
yticks = []
for i, c in enumerate(cluster_labels):
    c_silhouette_vals = silhouette_vals[y_km == c]
    c_silhouette_vals.sort()
    y_ax_upper += len(c_silhouette_vals)
    color = cm.jet(float(i) / n_clusters)
    plt.barh(range(y_ax_lower, y_ax_upper), c_silhouette_vals,
height=1.0,
             edgecolor='none', color=color)

    yticks.append((y_ax_lower + y_ax_upper) / 2.)
    y_ax_lower += len(c_silhouette_vals)

silhouette_avg = np.mean(silhouette_vals)
plt.axvline(silhouette_avg, color="red", linestyle="--")

plt.yticks(yticks, cluster_labels + 1)
plt.ylabel('Cluster')
plt.xlabel('Silhouette coefficient')

plt.tight_layout()
plt.savefig('P:/COMP809/11_06.png', dpi=300)
plt.show()
```