CLUSTERING

Reference:
Sklearn page:
https://scikit-learn.org/stable/index.html

Clustering:

https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html#sklearn.cluster.KMeans

https://scikit-learn.org/stable/modules/generated/sklearn.cluster.AgglomerativeClustering.html#sklearn.cluster.AgglomerativeClustering

Activity

Part -1: Understanding KMEANS

1. Import Numpy Pandas and Kmeans(from Sklearn)

2. Create a Numpy Array of 6 rows and 2 columns(for replication of results please use same numbers)

```
X = np.array([[1, 2], [1, 4], [1, 0],
              [4, 2], [4, 4], [4, 0]])
print(X)
```

3. Create a Pandas dataframe from the above array.

```
Y = pd.DataFrame(X,columns=['A','B'])
print("\n")
print(Y)
```

4. Apply Kmeans clustering an above dataframe. The way we do it is by first initializing the KMEANS object and then fitting the KMEANS object to our data.

5. To get the cluster-ID assigned for each of the data points, use the ‘labels_’ attribute from the Kmeans object.

6. For each of the clusters, we can get the cluster centers with the ‘cluster_centers_’ attribute.

7. To identify how the overall clustering is performing on the data, we use the average point of each of the points from the cluster centers. Use the attribute ‘inertia_’

8. Now, if there are new data points and if we want to ‘identify’ to which cluster to belong to (or closer to), use the predict method.

```
kmeans.predict([[0, 0], [4, 4]])
```

Part-2: Applying the KMeans understanding on a dataset.

Data: Cereals.csv

Dataset Description
* name: Name of cereal
* calories: calories per serving
* protein: grams of protein
* fat: grams of fat
* sodium: milligrams of sodium
* fiber: grams of dietary fiber
* carbo: grams of complex carbohydrates
* sugars: grams of sugars
* potass: milligrams of potassium
* vitamins: vitamins and minerals - 0, 25, or 100, indicating the typical percentage of FDA recommended
* shelf: display shelf (1, 2, or 3, counting from the floor)
* weight: weight in ounces of one serving
* cups: number of cups in one serving
* rating: rating of the cereals (Possibly from Consumer Reports)

1. Import Numpy, Pandas libraries and KMeans, Scaler and Imputer objects from Sklearn.

2. Read the Dataframe using read_csv and save it as Pandas dataframe.

```
cereals = pd.read_csv("Cereals.csv")
cereals.head(10)
```

3. Understand the dimensions and observe the data using head and tail functions.

4. From Data description and from the head, and tail outputs, identify which variables are not useful and which are valid for clustering.

5. Aggregate the columns ‘name’ shelf’ and ‘rating’ into one single column and name is as ‘label’.

```
cereals['label'] = cereals['name']+ ' (' +
cereals['shelf'].astype(str) + " - " +
round(cereals['rating'],2).astype(str) + ')'
cereals.drop(['name','shelf','rating'], axis=1,
inplace=True)
```

6. Observe the datatypes of the data using dtypes attribute.

7. To ensure the newly created label field is unique across along the dataframe, use the value_counts method. (this is one way, there can be a better way).

8. Perform summary statistics on the data.

9. To continue with clustering, remove the ‘label’ column and save it separately.

```
cereal_label = cereals['label']
cereals.drop('label', axis=1, inplace=True)
```

10. Check the null values in each of the columns.

```
cereals.isnull().sum()
```

11. To ‘Impute’ the null values, initialize the ‘Imputer’ object with ‘mean’ as the imputation strategy. Then, use the Imputer object to impute the null values on the data frame.

12. Check the null values again.
13. To Standardize the data, instantiate the ‘Scaler’ object from sklearn, then fit on the data and then transform the data using the ‘transform’ operation.

14. Perform head operation and summary statistics on the transformed data.
15. Perform KMeans Clustering similar to earlier activity (apply Kmeans, get centers of clusters, get the inertia of Kmeans model and iterate this by changing the number of clusters.

Part3: Identify the best number of clusters for the data.

1. Create an empty dictionary call ‘wss’. Now, Run a for loop to change the number of clusters and apply KMeans for each of them.

```
from sklearn.metrics import silhouette_samples,
silhouette_score
wss= {}
for k in range(2, 21):
```

```
    kmeans_loop =
KMeans(n_clusters=k,n_init=30,n_jobs=2,random_state=1000,ve
rbose=0).fit(std_cereals)
    clusters = kmeans_loop.labels_
    labels = kmeans_loop.predict(std_cereals)
    print(silhouette_score(std_cereals, labels))
    wss[k] = kmeans_loop.inertia_
```

2. Plot the scree plot for the above data (it is just the line plot between 'number of clusters' and 'wss')

```
import matplotlib.pyplot as plt
plt.figure()
# %matplotlib notebook


plt.plot(list(wss.keys()),list(wss.values()) ,marker='o')
plt.grid()
plt.xlabel('Number of clusters')
plt.ylabel('Total within sum of squares')
plt.show()
```

3. From Scree plot and Silhouette scores, identify the best value of 'K'. Re-build the elbow plot with preferred value of k. This should be your final model.

Part - 4: Post-processing

1. Build the final model from earlier identified value of k

```
best_kmeans = KMeans(n_clusters=8, random_state=1240)
best_kmeans.fit(std_cereals)
best_kmeans_labels = best_kmeans.predict(std_cereals)
```

2. Earlier we have save the labels for each of the rows of data. Lets add the corresponding cluster number to labels.

```
kmeans_results =
pd.DataFrame({"label":cereal_label,"kmeans_cluster":best_km
eans_labels})
kmeans_results.head()
```

3. Now let's add the cluster number to the original data.

```
cereals = pd.read_csv("Cereals.csv")
cereals['label'] = cereals['name']+ ' (' +
cereals['shelf'].astype(str) + " - " +
round(cereals['rating'],2).astype(str) + ')'
cereals.drop(['name','shelf','rating'], axis=1,
inplace=True)
```

```
final_cluster_data = pd.merge(cereals, kmeans_results,
on='label')
final_cluster_data.head(10)
```

4. We can check the number of data points in each cluster with 'value_counts' function.
```
final_cluster_data.kmeans_cluster.value_counts().
```

5. Let us look at the centers at each of the clusters in the original scale of the data.
```
analysis =
final_cluster_data.groupby(['kmeans_cluster']).mean().reset
_index()
analysis
```

6. Can you infer anything from each of the cluster centers?? Can you give names to the clusters?

Part 5: Hierarchal Clustering

1. Import Linkage and Dendrogram from scipy library.
```
from scipy.cluster.hierarchy import linkage, dendrogram
```

2. Build the linkage matrix for the cereals data.
```
linkage_matrix = linkage(std_cereals,
method='ward',metric='euclidean')
```

3. Using the linkage matrix build the Dendrogram.
```
dendrogram(linkage_matrix,labels=cereal_label.as_matrix())
```
4. What is the ideal number of clusters from the above plot?
5. Import Heirchal clustering from Sklearn.
```
from sklearn.cluster import AgglomerativeClustering
```

6. Apply Hierarchal clustering on the Cereals data.
```
## Instantiating object
agg_clust = AgglomerativeClustering(n_clusters=6,
affinity='euclidean', linkage='ward')

## Training model and return class labels
agg_clusters = agg_clust.fit_predict(std_cereals)

## Label - Cluster
```

```
agg_result =
pd.DataFrame({"label":cereal_label,"agg_cluster":agg_cluste
rs}).sort_values('agg_cluster')
agg_result.head()
```