Module 8 exercises: Unsupervised Learning

The data

- We will be using the well explored Wine Data from SciKit-Learn. There are 13 features and a 'real_class' column.
- Please remember that in the real world you will not have a real class column, as the goal of clustering is to find out how many distinct classes there are and then analyse the properties of each, in order to derive some value.

Exercise 1

1. Import the following packages:
    a. pandas
    b. matplotlib
    c. numpy
    d. seaborn
2. Run the following code to obtain the wine dataset from `sklearn`:

```
from sklearn.datasets import load_wine

## Import the Wine dataset from SciKit-Learn
wine_bunch = load_wine()

# Allocating this data to a Data Frame
wine = pd.DataFrame(wine_bunch.data)

# Use the feature_names attribute to give sensible column headings
# Assigning wine.columns to col names for use later
col_names = list(wine_bunch.feature_names)

# Once colum has a lenghty name, we rename this for ease of display
# od280/od315_of_diluted_wines   -> od280_od315
col_names[11] = 'od280_od315'
wine.columns = col_names

# I am adding a 1 to the 'real classes' columns to avoid having to talk
about 'cluster 0'
# We will start the count from 1
wine['real_classes'] = wine_bunch.target + 1
```

```
wine.head()
```

Exercise 2 – EDA

Familiarise yourself with the dataset. Seek to understand:

- the types of data.
- distributions of features.
- obvious patterns.
- errors, nulls, outliers.

Exercise 3 – Building your first model

After exploring the dataset, we can apply our clustering algorithm. We start by using the raw data as-is, though down the line we may make some changes.

1. Import the `KMeans` model from scikit-learn's `cluster` module.
2. Create a `KMeans` model, setting `n_clusters=2`, and calling it kmeans1. Then use the `fit` method to train the model on `wine[col_names]`.
3. Store the centroids and add a column giving each point it's label. The code below can be used:

```
## The centroids can be extracted
centroids1 = kmeans1.cluster_centers_

## This is the list of allocated classes
labels1 = kmeans1.labels_

## Appending these labels to our original dataframe
# I added 1 again to avoid talking about class 0
wine['kmeans1'] = labels1 + 1
```

4. Compute the mean value of each feature per group label given by `KMeans.` What are the key differences between the groups?
5. Use the below code to plot the clustered data:

```
import matplotlib as mpl
mpl.rcParams['axes.prop_cycle'] = mpl.cycler(color=["b", "r", "g"])

plot = pd.plotting.scatter_matrix(wine[col_names],
                                  figsize=(20,20),
                                  marker = '.',
```

```
s = 15,
alpha = 0.8,
c = wine['kmeans1'])
```

6. If you have time, try to display the count of actual label vs. predicted label for each group.

Exercise 4 – Changing clusters and features:

1. Repeat Exercise 3 but using three clusters. Compare this to the original classes as we did above
2. Repeat it again but this time using only the below five features:
    - Alcohol
    - Alcalinity_of_ash
    - Magnesium
    - Color_intensity
    - Proline
3. How does this model with five features compare to the model with all? Why do you think this is the case? What would you do to resolve the issue?
4. Re-run once more with only one feature (you should see which) * Examine the output of this

    Note: For each model create new columns in the DataFrame to store the results, e.g., kmeans2 and kmeans3.

Exercise 5 – Scaling

1. Our model was being affected by the large scale of some of our features. Apply the `MinMaxScaler` which can be imported from sklearn's preprocessing module to each feature.
2. Repeat Exercise 3, fitting `KMeans` to the scaled dataset.
3. What difference do you see?

Exercise 6 – Choosing the optimal number of clusters

1. Run the below code and examine the output. What does it show you?

```
## We use a for loop to try different numbers of clusters
## For each k, we calculate the 'inertia' and record these values
WCSS = {}
K = range(1,16)
for k in K:
    km = KMeans(n_clusters=k, random_state=42).fit(wine_std)
    WCSS[f'{k}'] = km.inertia_

## Now we can plot to help us identify the 'optimal' value for k
plt.plot(K, WCSS.values(), 'bx-')
plt.xlabel('k')
plt.ylabel('Within Sum of Squares')
plt.title('Number of clusters')
plt.show()
```

Exercise 7 – Extension

1. Examine the output of the below code which computes the silhouette coefficient and visualises it. Explore the metric and interpret the outputs.

```
from sklearn.metrics import silhouette_score, silhouette_samples

silhouettes = silhouette_samples(X=wine[col_names],
                                 labels=wine['kmeans5'])
wine["silhouette"] = silhouettes

g = sns.FacetGrid(wine, col="kmeans5")
g.map(sns.histplot, "silhouette")
```

In a new cell

```
## We use a for loop to try different numbers of clusters
silhouettes = {}
K = range(2,16)
for k in K:
    km = KMeans(n_clusters=k, random_state=42).fit(wine_std)
    silhouettes[f'{k}'] = silhouette_score(X=wine_std,
                                           labels=km.labels_)
## Now we can plot to help us identify the 'optimal' value for k
plt.plot(K, silhouettes.values(), 'bx-')
plt.xlabel('k')
plt.ylabel('Silhouette Score')
plt.title('Number of clusters')
plt.show()
```

Exercise 8 – Other clustering approaches

If you have time, explore the following other clustering approaches:

- DBSCan
- Gaussian Mixture Modelling
- WARD