K-Means on n-Dimensional Data

This lesson will focus on K-Means on n-dimensional data in Python.

K-means on n-dimensional data

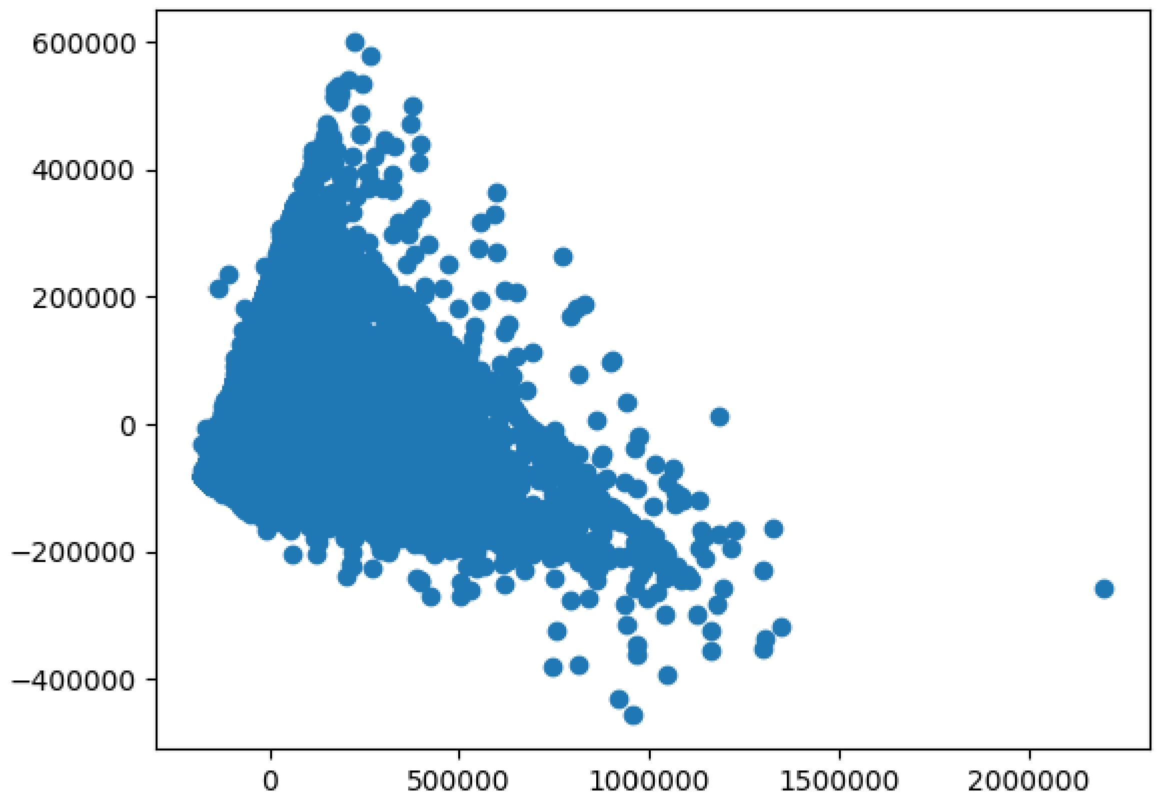
For clustering n-dimensional data, we can use the same procedure. The only difference is that we first need to reduce the number of dimensions to 2. For reducing dimensions, we will use a technique called **Principal Component Analysis (PCA)**. We do not need to go into the details of how this technique works since the details are out of the scope of this course. We can reduce data to 2 dimensions by using the `PCA` class available in `sklearn.decomposition`. We will cluster the reduced data.

We will be using the Default of Credit Card Dataset. This dataset originally had two classes. Naturally, if we do not have the class labels, we would make two clusters for this example.

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

# Make data
df = pd.read_csv('credit_card_cleaned.csv')
Y = df['default.payment.next.month']
df = df.drop(columns = ['default.payment.next.month','GENDER','MARRIAGE','ID'])
print('shape of original data set : ',df.shape)

# PCA
pca = PCA(n_components=2)
X = pca.fit_transform(df)
X = pd.DataFrame(X)
X.columns = ['feature_1','feature_2']
print('shape of transformed data set : ',X.shape)

# Plot
plt.scatter(x= X['feature_1'],y = X['feature_2'])
plt.scatter(x = X['feature_1'], y = X['feature_2'])
```

We import the `PCA` class from `sklearn.decomposition` in **line 3**. After reading the data in **line 6**, we separate the labels in `Y` in **line 7**. We drop the categorical columns in the next line. We initialize the `PCA` class in **line 12** with `n_components=2`, which means we need to reduce dimensions to 2. The `PCA` class has the function `fit_transform`, which gives us the transformed data. We use `fit_transform` in **line 13** and store the reduced data in `X`. At this time, the returned data is not a dataframe, rather it is a 2-dimensional list. So we convert it into a dataframe in **line 14** by giving the list to `pd.DataFrame`. We name the columns `feature_1` and `feature_2` in **line 15**, so they are easy to access later. From the outputs of **lines 9 and 16**, we can see that the data has been reduced to 2 columns.

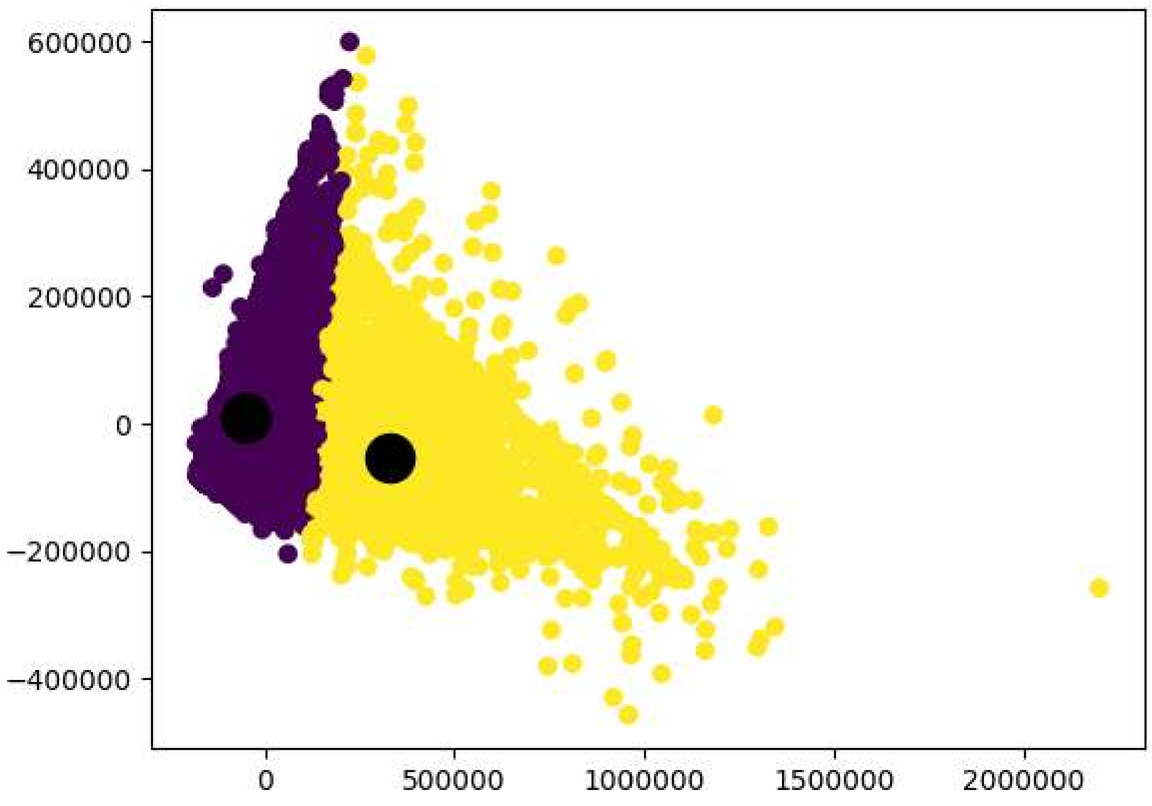
In the end, we just plot the reduced data as we did for the above simple example.



Now we have this data that we need to cluster. Just by looking at the data, we cannot easily separate the data into two clusters. Let's see how K-means clustering does that for us below.

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.metrics import accuracy_score

# Make Data
df = pd.read_csv('credit_card_cleaned.csv')
Y = df['default.payment.next.month']
df = df.drop(columns = ['default.payment.next.month','GENDER','MARRIAGE','ID'])
print('shape of original data set : ',df.shape)

# PCA
pca = PCA(n_components=2)
X = pca.fit_transform(df)
X = pd.DataFrame(X)
X.columns = ['feature_1','feature_2']
print('shape of transformed data set : ',X.shape)

# kmeans clustering on reduced data
km = KMeans(n_clusters = 2,random_state=2)
km.fit(X)

# predictions
preds = km.predict(X)
centers = km.cluster_centers_

# plots
plt.scatter(x = X['feature_1'],y = X['feature_2'],c = preds)
plt.scatter(x = centers[:,0],y = centers[:,1],c = 'k',s = 300)

# Set classes to 0 and 1 to compare results
Y.loc[Y=='no'] = 0
Y.loc[Y=='yes'] = 1
print(accuracy_score(y_true = Y,y_pred = preds))
```

In **line 21**, we initialize the `KMeans` class object with `n_clusters = 2`, which means that we want 2 clusters. Then we use the function `fit` and give it the reduced data `X`. We obtain the predicted classes by using the `predict` function in **line 25**. In **line 26**, we retrieve the cluster centers. We plot the data and color the data by the predicted classes in **line 29**. We plot the cluster centers in **line 30**.



From the produced plot, we can see that the data has been divided into two clusters. We can see a boundary between the two clusters.

To compare the results of unsupervised learning to supervised learning, we can compare the predicted clusters with the actual labels that we have. For that, we need to set the class label `no` to 0 and `yes` to 1 since clustering predicted classes as `0` and `1`. We do that in **lines 33 and 34**. Then we print the accuracy with the predicted labels and actual labels. The accuracy comes out to be approximately 70%, which is decent because we got approximately 75-80% accuracy with supervised learning models on this dataset.

This concludes the lesson on unsupervised learning. In the next lesson, you will be tested on the concepts that you have learned in this lesson.