COURSEWORK REPORT

DATA70132 : STATISTICS AND MACHINE LEARNING 2

2021-2022

CONTENT

CLASSIFICATION

1. K-Means : Unsupervised clustering method

K-Means clustering is a type of unsupervised learning algorithm which is used to cluster data points into groups based on some similarity when we do not have the predefined labels in our dataset. It does not try to make any predictions but rather clubs together similar data points, and these clusters can be further used to analyse the data.

The k-means algorithm looks for a fixed number of clusters (k) within an unlabeled multidimensional dataset. The optimal value of cluster (k) is selected using the elbow method or the silhouette method. For the vertebral dataset, optimal $k = 2$, as derived by both methods shown in Figure 1.1a and 1.1b.

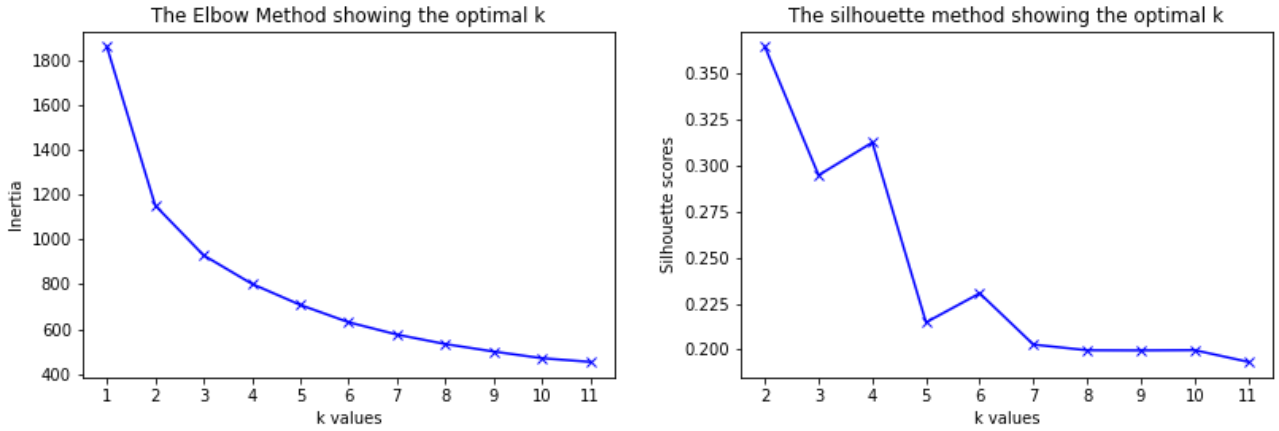

Figure 1.1 a. The elbow method plot shows the inflection point at k = 2;
b. The silhouette method plot has its global maxima at k=2

Once we have the optimal k, the algorithm takes k randomly selected centroids, which are used as the initial points for each cluster, and then performs iterative calculations to optimise the centroids.
The process stops when the centroids have stabilised or the preset number of iterations is complete.
Each point in a cluster is selected such that it is closer to its designated cluster's centre than to other clusters' centres.

2. Decision Tree Classifier : Supervised classification method

The decision tree classifier is a supervised machine learning algorithm based on a sequence of hierarchical conditions. These conditions help decide splitting the data points into the possible classes at each level until we get a homogeneous distribution in each node. This methodology is also known as a *divide and conquer algorithm*.

Decision tree classifiers are made up of three elements: the nodes, the branches, and the leaves. Each node describes a test condition on a particular feature of the dataset, the result of which branches the data out into the possible classes. This process continues till we get to the end of the decision path or the leaves. The selection of attributes to evaluate at each level in a decision tree is based on the criterion selected like gini index, entropy, etc.

For the vertebral data set we are using *Entropy* as the attribute selection criterion. Entropy is a measure of information that indicates the impurity inthe features. The feature which is chosen such that it has a lower entropy value. The final decision tree created is shown in Figure 2.1.

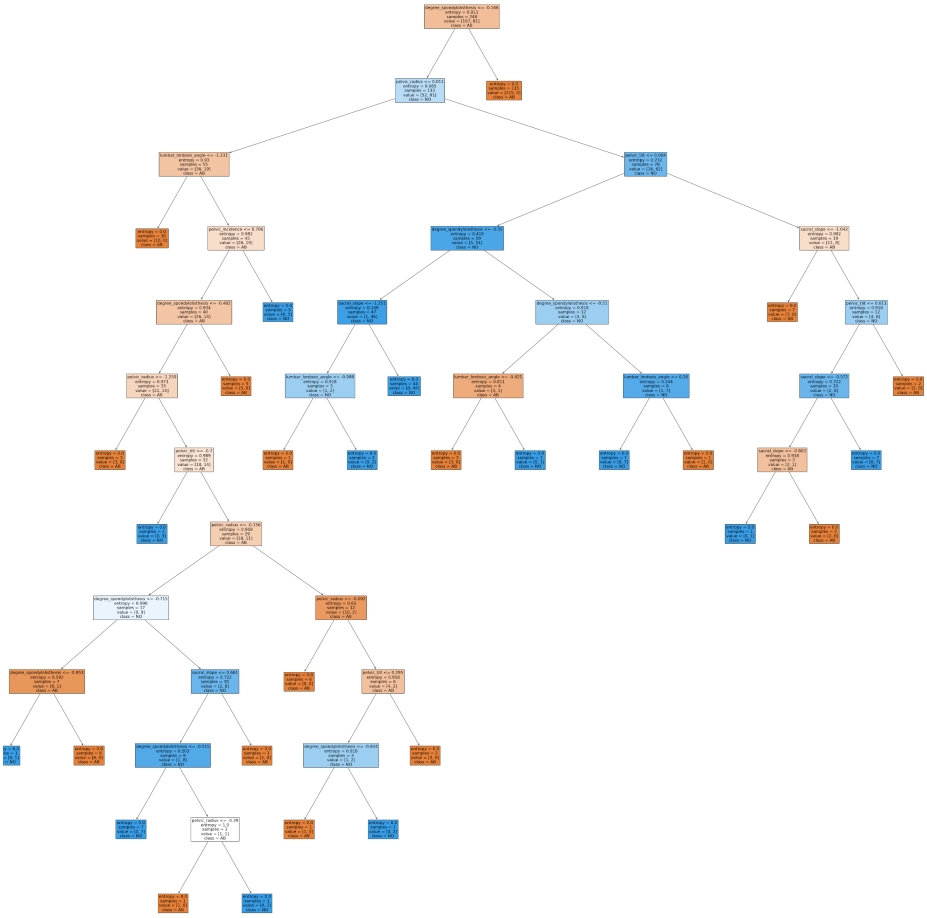

Figure 2.1: The decision tree created for Vertebral Data using Entropy as criteria

3. Data pre-processing and Exploratory Data Analysis

The first step when dealing with a data problem is to perform data pre-processing to ensure that the quality of the data is trustworthy. For this dataset, no missing values were found and the datatypes of all the features were as expected.

On performing EDA on the dataset we found that it consisted of 6 independent features of numerical type and 1 target variable of object type. There were 310 records present out of which 210 belonged to the *class AB* and 100 to *class NO* as shown in Figure 3.1.

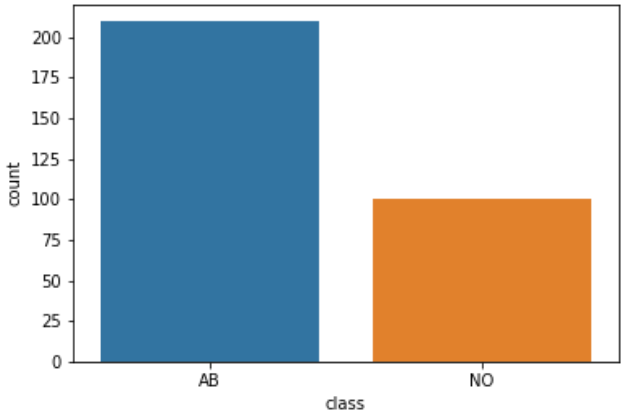

Figure 3.1: Data distribution according to output classes

Doing a bivariate analysis (Figure 3.2) we find that data points belonging to the *AB* category have higher *degree_spondylolisthesis* than those in the *NO* category (healthy patients). This means the feature *degree_spondylolisthesis* has a strong relationship to the result of the patient's condition being normal or abnormal. Also, patients with low *pelvic_radius* usually fall in the AB category (suffering from spinal problems). For the rest of the attributes, we see that people with lower values are healthy and do not suffer spinal problems.

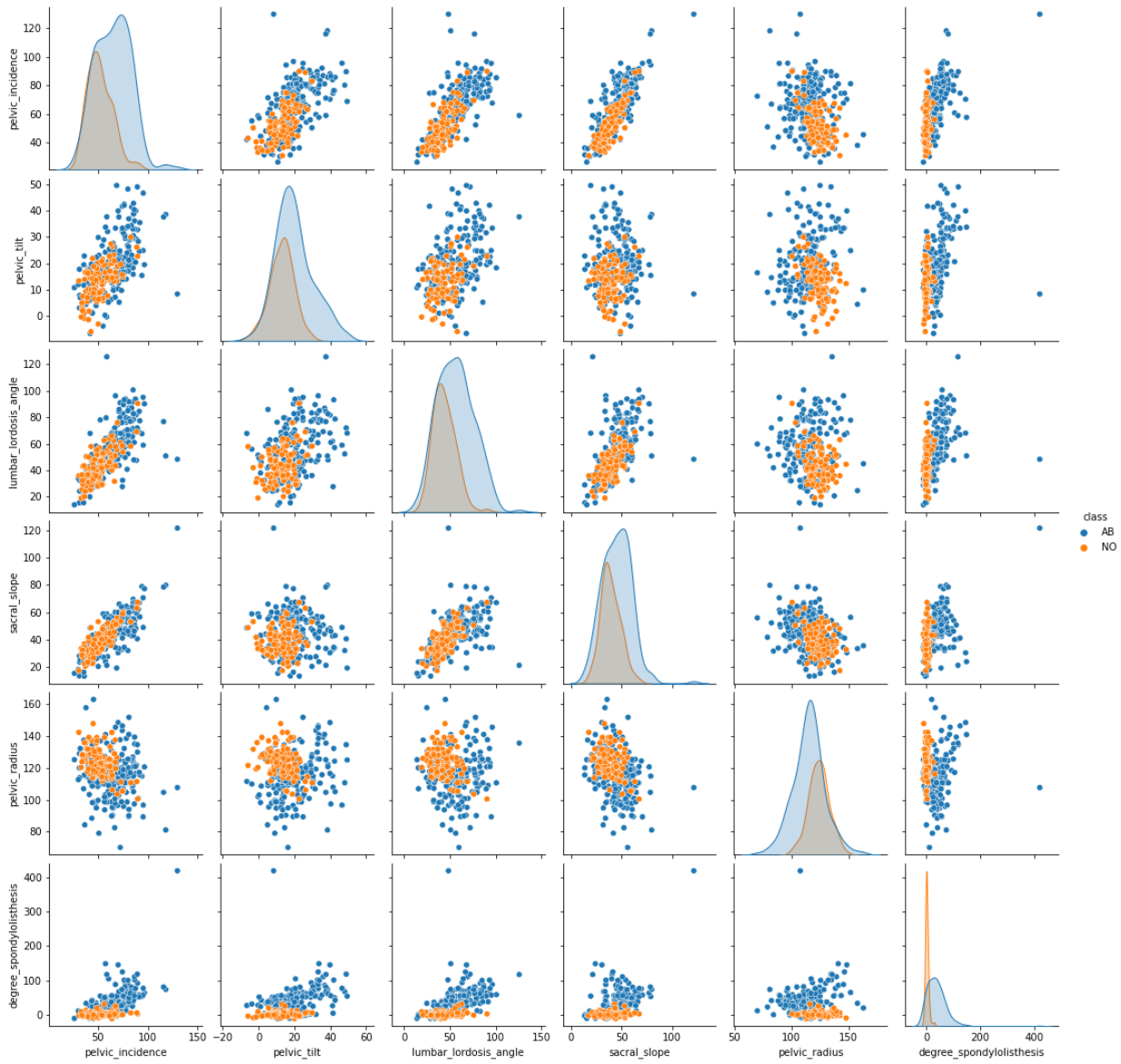

Figure 3.2: Bi-variate analysis plot for the 6 features with output class as colour mapping (Blue = AB, Orange = NO)

From the multivariate analysis we get the correlation matrix for the independent features (Figure 3.3), which shows that *sacral_slope* and *pelvic_incidence* are highly positively correlated and *sacral_slope* and *pelvic_radius* are highly negatively correlated. Since we have a hypothesis from the bivariate analysis results, that a lower *pelvic_radius* usually means spinal problems, therefore we can also hypothesise that for a healthy person the *sacral_slope* and by extension, *pelvic_incidence* should be lower.

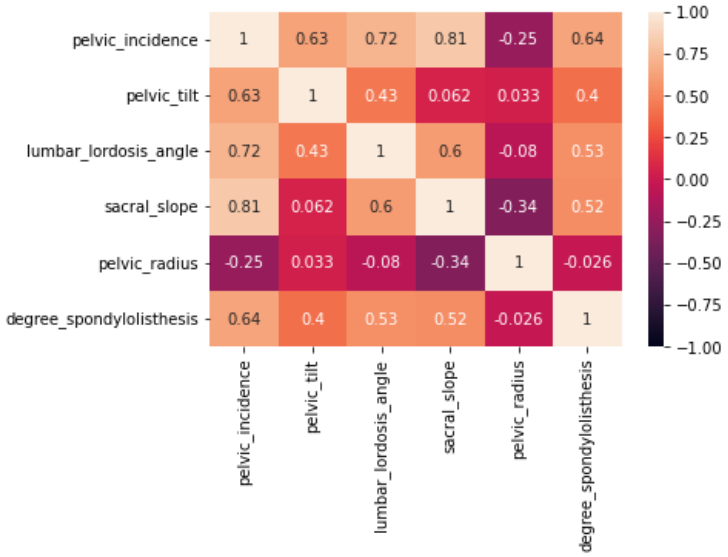

Figure 3.3: Heatmap to show correlation between the 6 independent features

After performing the EDA we split the dataset into the independent features (X) and target feature (Y). Then the features are scaled using the StandardScaler so that the data is normalised and lies in the same range. We then perform a dimensionality reduction on the dataset using PCA which helps us reduce the dimensionality from 6 to 4, since 4 principal components can effectively account for 95% of the data variability as seen in Figure 3.4.

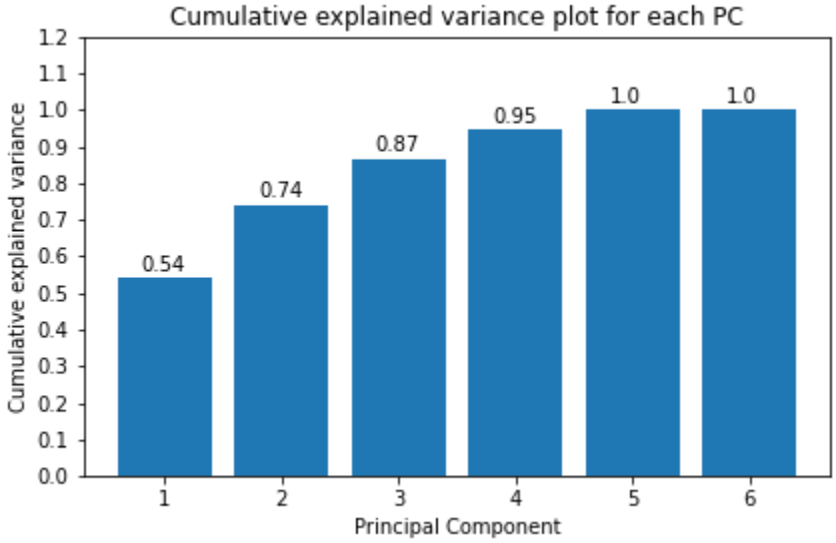

Figure 3.4: Cumulative explained variance plot for PCA

We then select the first 4 PCs and draw a scatter plot as shown in Figure 3.5, to visualise the class distribution for each component.

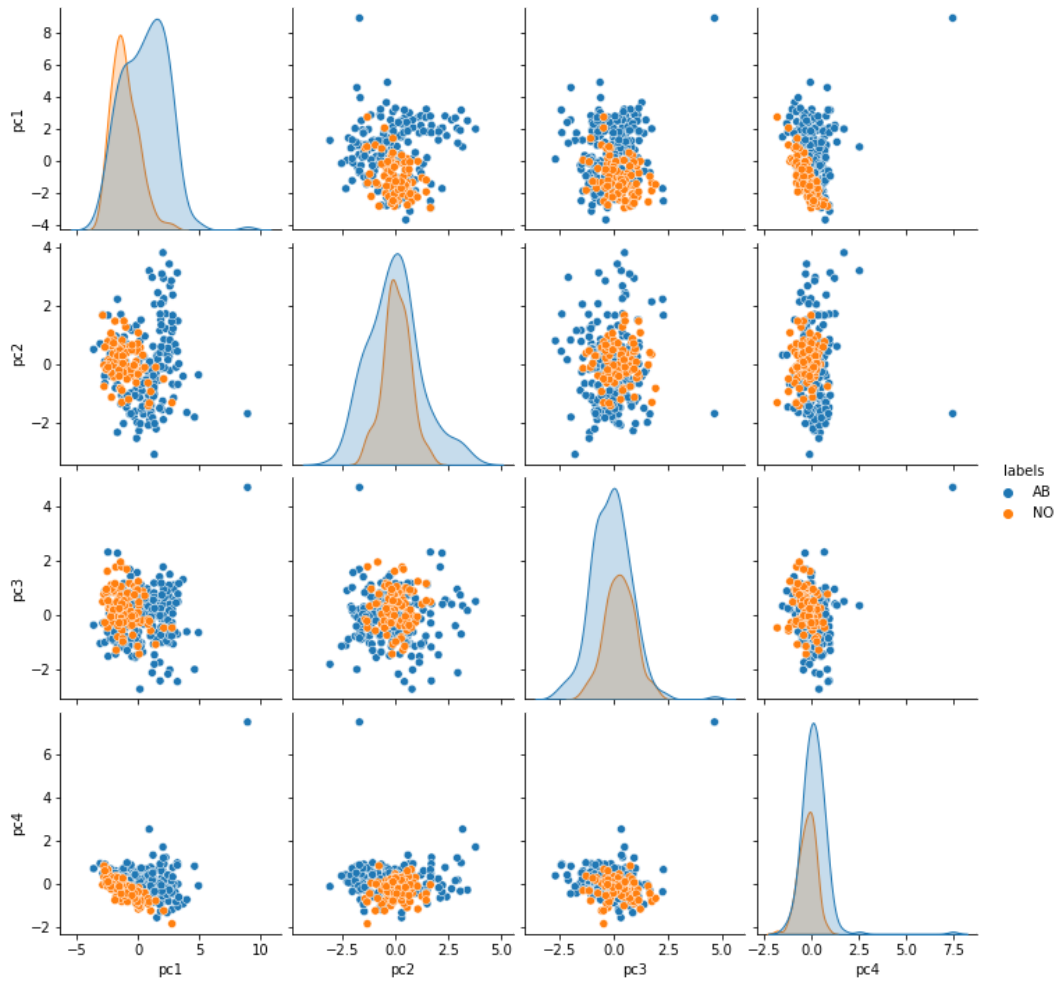

Figure 3.5: Pair-plot for the selected 4 principal components, with output class as colour mapping (Blue = AB, Orange = NO)

4. K-Means clustering results and analysis

On performing the K-Means clustering for the features with and without PCA we see that 2 clusters are formed in both cases as shown in Figure 4.1 and 4.2. The boundaries created by the K-Means cluster are clearly visible unlike the actual distribution which had no clear boundaries.

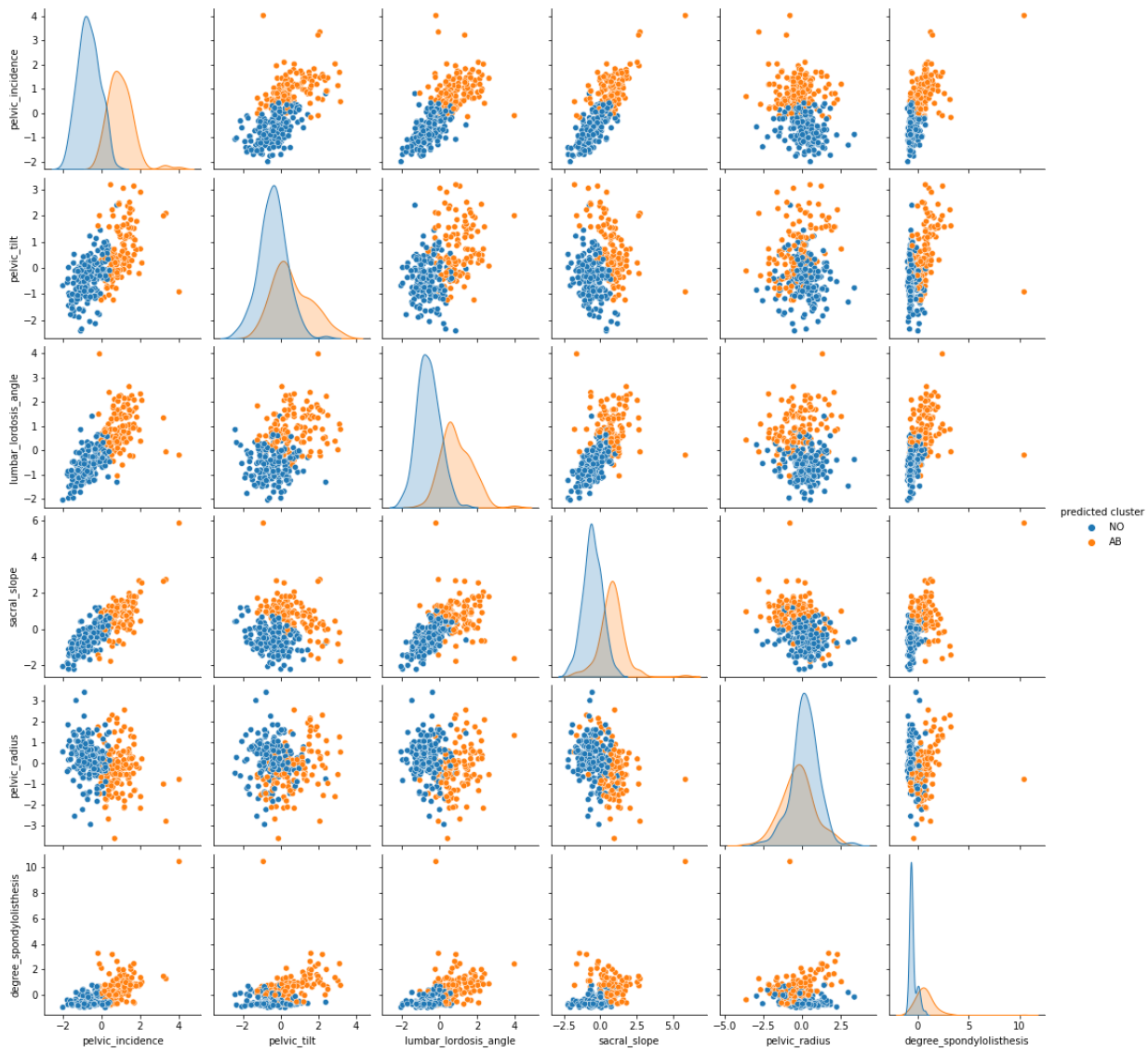

Figure 4.1: Pair-plot for the original features, with clusters assigned by K-Means used as colour mapping (Blue = NO, Orange = AB)

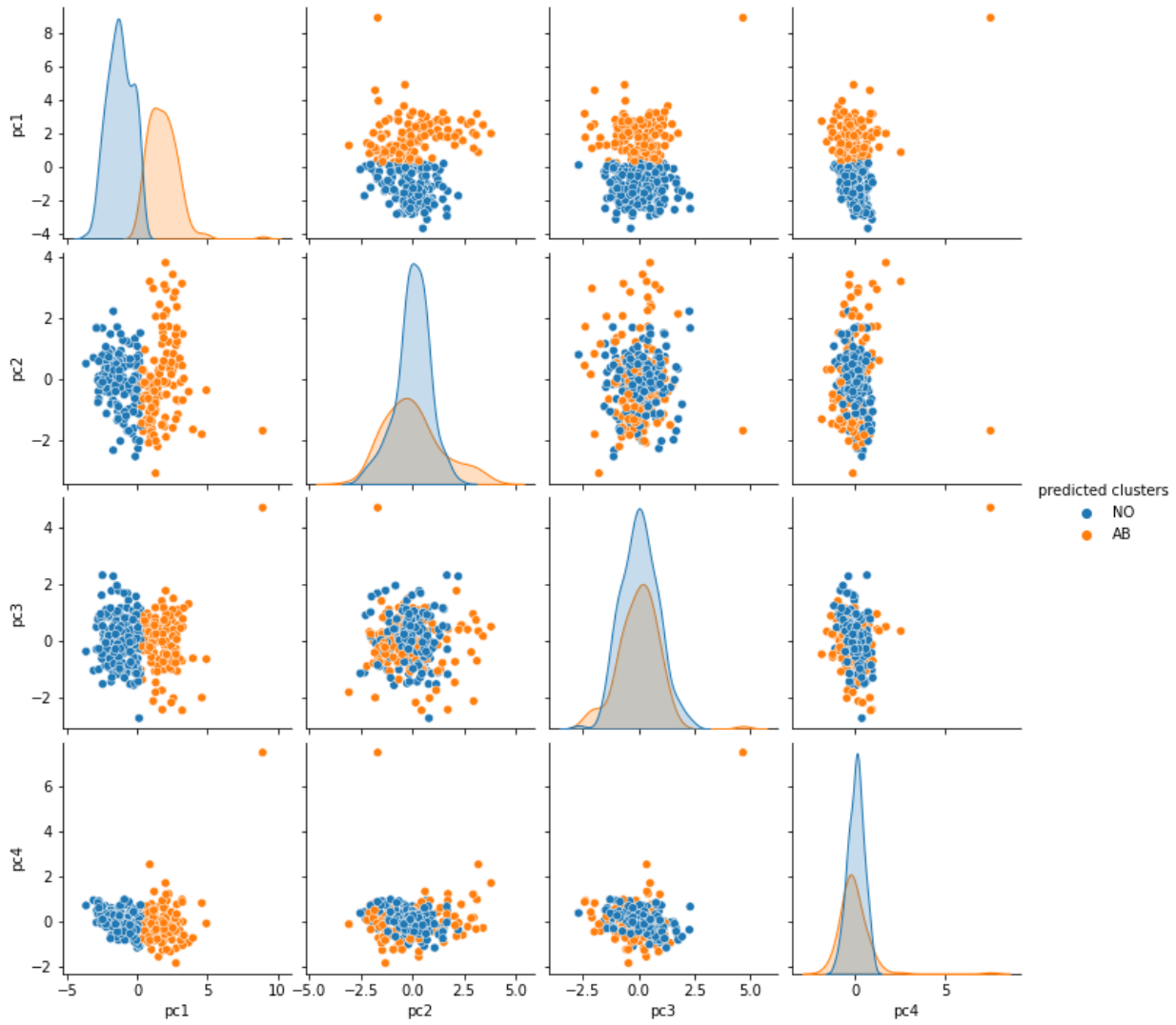

Figure 4.2: Pair-plot for the 4 selected principal components, with clusters assigned by K-Means used as colour mapping (Blue = NO, Orange = AB)

Although usually we do not have the actual labels for data being clustered, so an accuracy score is not logical. But in this case since we had the actual classes, we performed a comparison between the clusters assigned and the actual class labels. On comparison we find that the accuracy of the algorithm comes out to be around 66% in both cases (with or without PCA) successfully assigning 206 out of 310 data points in clusters that correspond to the actual classes as shown in Figure 4.3 a., and 4.3.b.

```
The kmeans cluster centers are =
 [[ 0.97012742  0.60976985  0.88853035  0.79105929 -0.27089815  0.73078755]
 [-0.66432638 -0.41755979 -0.60845013 -0.54170364  0.18550634 -0.5004306 ]]
The kmeans inertia =  1151.8966087304477
--------------------------------------------------

Result: 206 out of 310 samples were correctly labeled
Accuracy score: 0.66
```

```
The kmeans cluster centers are =
 [[ 1.82483609  0.01053572 -0.02893855 -0.0583396 ]
 [-1.24961602 -0.00721468  0.01981662  0.03994994]]
The kmeans inertia =  1051.105553459721
------------------------------------------------------------

Result: 206 out of 310 samples were correctly labeled
Accuracy score: 0.66
```

Figure 4.3 : K-Means cluster centres and inertia calculated along with the accuracy score for a) Original scaled data without PCA; b) Data after PCA

5. Decision Tree Classification results and analysis

On performing decision tree classification for the feature without PCA (Figure 5.1), we see that the accuracy comes out to be 80%. We also see that 35 out of 43 data points of patients with spinal problems were correctly classified and 8 were incorrectly classified as healthy. 4 out of 19 healthy people were incorrectly classified as patients of spinal problems, and 15 were correctly classified as healthy.

Since this is medical data therefore recall is more important, since we do not want an unhealthy person to be incorrectly labelled as healthy.

```
The classification report for Decision Tree :
              precision    recall  f1-score   support

          AB       0.90      0.81      0.85        43
          NO       0.65      0.79      0.71        19

    accuracy                           0.81        62
   macro avg       0.77      0.80      0.78        62
weighted avg       0.82      0.81      0.81        62

-----------------------------------------------------

The confusion matrix for Decision Tree :
[[35  8]
 [ 4 15]]
-----------------------------------------------------

The accuracy for  Decision Tree is 0.8064516129032258
```

Figure 5.1: Decision Tree result for data without PCA

On performing decision tree classification for the feature with PCA (Figure 5.2), we see that the accuracy comes out to be 75%, which is less than what we got for data without PCA. This means that our assumption that the classifier would perform better after dimensionality reduction was found to be incorrect. This resonates with the findings of the research paper done to review the effects of PCA on classification techniques when used with medical data. The authors stated that although PCA proves quite useful in case of unsupervised learning, there is no guarantee that the components derived help in a supervised classification problem [1].

We also see that 35 out of 43 data points of patients with spinal problems were correctly classified and 8 were incorrectly classified as healthy. 7 out of 19 healthy people were incorrectly classified as patients of spinal problems, and 12 were correctly classified as healthy.

Due to this, recall is also lower in case of data with PCA than data without PCA.

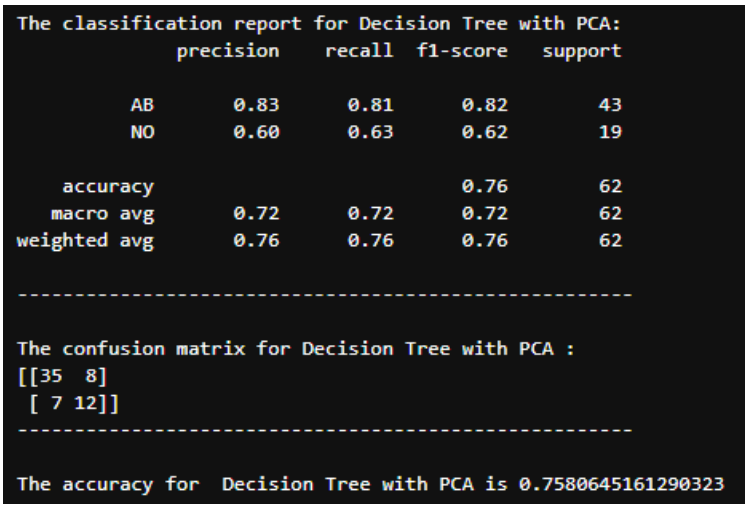

Figure 5.2: Decision Tree result for data with PCA

6. Decision Tree Classification vs K-Means Clustering

On comparing the two algorithms we see that classification gives a better accuracy which can be expected since classification being a supervised learning algorithm, had access to the true labels while learning whereas clustering merely clubbed the data points together based on their similarities.

The results from the K-means clustering can further be used to improve the classification accuracy when the two techniques are used together in an ensemble model [3]. This can be done in two ways [2]:

- Taking the clusters as the new features
- Segregating the dataset into multiple parts based on the clusters assigned and training different classifiers on them

APPENDIX

1. Code for Data Pre-processing and EDA

```
# importing necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
# for statistical data visualisation
import seaborn as sns

# getting column names from the dataset source
columns = ["pelvic_incidence", "pelvic_tilt", "lumbar_lordosis_angle",
"sacral_slope", "pelvic_radius", "degree_spondylolisthesis", "class"]

# read the data from the file
vertebral_data = pd.read_csv("./ClassificationLab/Data/vertebral_column_data.txt",
names = columns, sep = " ")
vertebral_data.head()

# EDA
###################################
# Numerical analysis of the dataset
print("Shape of the dataset : ", vertebral_data.shape)
print("-------------------------------------------------------------------------")
print("Count of null values in each column :\n")
print(vertebral_data.isna().sum())
print("-------------------------------------------------------------------------")
print("Column data types for dataset: \n")
print(vertebral_data.dtypes)
print("-------------------------------------------------------------------------")
print("Summary statistics for dataset: \n")
vertebral_data.describe()

# 1) Univariate Analysis
# histogram plot for feature data
print("\nHistogram plots for the features:\n")
ax = vertebral_data.hist(bins = 25, ec="white")
plt.tight_layout()
plt.savefig('./FiguresCW2/Histogram plot.png',format='png')
plt.show()

# 2) Bivariate Analysis
# pair plot for the features, to see if they have correlations and how they affect
each other
```

```
print("\nPair plot for the features:\n")
sns.pairplot(vertebral_data, hue="class")
plt.savefig('./FiguresCW2/Pair plot 1.png',format='png')
plt.show()

# 3) Multivariate Analysis
# heat map to visualise correlation matrix for the features
print("\nHeat map for correlation between the features:\n")
sns.heatmap(vertebral_data.corr(), vmax=1, vmin=-1, annot=True)
plt.savefig('./FiguresCW2/Heat map.png',format='png')
plt.show()

# count of data belonging to both the classes
vertebral_data.loc[:,'class'].value_counts()
sns.countplot(x="class", data=vertebral_data)
plt.savefig('./FiguresCW2/Class distribution.png',format='png')
plt.show()

# separating the labels from the features
class_labels = vertebral_data["class"]
vertebral_data.drop("class", inplace = True, axis = 1)

# import scaler from sklearn
from sklearn.preprocessing import StandardScaler
# Feature scaling
# normalise the dependent variables, so that they are in a particular range using
standard scaler
stdScaler = StandardScaler()
vertebral_data_scaled = stdScaler.fit_transform(vertebral_data)

# creating a dataframe for the scaled features
scaled_features_df = pd.DataFrame(vertebral_data_scaled,
index=vertebral_data.index, columns=vertebral_data.columns)
```

2. Code for PCA

```
# PRINCIPAL COMPONENT ANALYSIS (PCA)
# import PCA
from sklearn.decomposition import PCA

# create PCA instance, fit and transform the standardised data with pca
pca = PCA()
pc = pca.fit_transform(scaled_features_df)
```

```
# calculate the cumulative sum of the explained variance ratio for each principal
component
cum_var_exp = pca.explained_variance_ratio_.cumsum()

# plot the bar graph for cumulative explained variance for each principal component
PC_values = np.arange(pca.n_components_) + 1
bars = plt.bar(PC_values, cum_var_exp)
# access the bar attributes to place the text values on top of each bar
for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + .2, yval + .02, round(yval,2))

plt.title('Cumulative explained variance plot for each PC')
plt.ylabel('Cumulative explained variance')
plt.xlabel('Principal Component')
plt.yticks(np.arange(0, 1.3, step=0.1))
plt.tight_layout()
plt.savefig('./FiguresCW2/PCA variance plot.png',format='png')
plt.show()

# change to data frames first
vertebral_data_scaled_pca = pd.DataFrame(pc, columns = ['pc1','pc2','pc3', 'pc4',
'pc5', 'pc6'])

X_pca = vertebral_data_scaled_pca[['pc1','pc2','pc3', 'pc4']]
X_pca["labels"] = class_labels

# Scatter plot for the 3 top Principal components with actual labels
sns.pairplot(X_pca, hue="labels")
plt.savefig('./FiguresCW2/Pair plot PCA.png',format='png')
plt.tight_layout()
plt.show()
```

3. Code for K-Means Clustering

```
# Unsupervised clustering - K-Means
##########################################

# finding optimal k value using Elbow method
from sklearn.cluster import KMeans

inertias = []
K = range(1,12)
for k in K:
```

```
    kmean = KMeans(n_clusters=k)
    kmean.fit(scaled_features_df)
    inertias.append(kmean.inertia_)

plt.plot(K, inertias, 'bx-')
plt.xlabel('k values')
plt.ylabel('Inertia')
plt.title('The Elbow Method showing the optimal k')
plt.xticks(np.arange(1,12))
plt.savefig('./FiguresCW2/Elbow method.png',format='png')
plt.show()

# finding optimal k value using Silhouette method
from sklearn.metrics import silhouette_samples, silhouette_score

# The Silhouette Score reaches its global maximum at the optimal k
# Therefore a high Silhouette Score is desirable
silhouettes = []
K = range(2,12)
for k in K:
    kmean = KMeans(n_clusters=k)
    kmean.fit(scaled_features_df)
    silhouettes.append(silhouette_score(scaled_features_df, kmean.labels_, metric =
'euclidean'))

plt.plot(K, silhouettes, 'bx-')
plt.xlabel('k values')
plt.ylabel('Silhouette scores')
plt.title('The silhouette method showing the optimal k')
plt.xticks(np.arange(2,12))
plt.savefig('./FiguresCW2/Silhouette Method.png',format='png')
plt.show()
# selected optimal k value as 2, because that is the global maxima in the graph

# create the k means clustering model
kmeans = KMeans(n_clusters = 2, random_state = 0)
# fit the data to the model
y_kmeans = kmeans.fit_predict(scaled_features_df)
# calculate the cluster centres for the data
clusters = kmeans.cluster_centers_
# calculate the inertia for the data
inertia = kmeans.inertia_
print("The k means cluster centres are = \n", clusters)
print("The kmeans inertia = ", inertia)

# evaluate and store the labels generated by the model
```

```
labels = kmeans.labels_
# changing predicted labels to the actual classes
mapping = {0:'AB', 1:'NO'}
y_pred = [mapping[i] for i in labels]
# calculating number of cluster labels that match the actual class labels
correct_labels = sum(class_labels == y_pred)
print("-----------------------------------------------------")
print("\nResult: %d out of %d samples were correctly labelled" % (correct_labels,
class_labels.size))
print('Accuracy score: {0:0.2f}'. format(correct_labels/float(class_labels.size)))

# creating a new dataframe with the scaled features and the predicted clusters
X_Kmeans = scaled_features_df.copy()
X_Kmeans["predicted cluster"] = y_pred

# plotting the scatter plot for the predicted clusters
sns.pairplot(X_Kmeans, hue="predicted cluster")
plt.savefig('./FiguresCW2/Pair plot Kmeans.png',format='png')
plt.show()

# importing the sklearn libraries for creating the confusion matrix and
classification report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

# Confusion matrix for K-Means clustering
kmeans_conf_matrix = confusion_matrix(class_labels, y_pred)
kmeans_report = classification_report(class_labels, y_pred)
print("Confusion matrix for K-Means :")
print(kmeans_conf_matrix)
print("-----------------------------------------------------")
print("\nClassification Report for K-Means:")
print(kmeans_report)
```

4. Code for K-Means Clustering with PCA

```
# K Means using the data from PCA
# fit the data to the model
y_kmeans = kmeans.fit_predict(X_pca.drop("labels", axis=1))
# calculate the cluster centres for the data
clusters = kmeans.cluster_centers_
# calculate the inertia for the data
inertia = kmeans.inertia_
print("The k means cluster centres are = \n", clusters)
```

```
print("The kmeans inertia = ", inertia)

# evaluate and store the labels generated by the model
labels = kmeans.labels_
mapping = {0:'AB', 1:'NO'}
y_pred = [mapping[i] for i in labels]
# checking the number of cluster labels predicted that match the actual classes
correct_labels = sum(class_labels == y_pred)
print("-----------------------------------------------------")
print("\nResult: %d out of %d samples were correctly labelled" % (correct_labels,
class_labels.size))
print('Accuracy score: {0:0.2f}'. format(correct_labels/float(class_labels.size)))

# plotting the scatter plots for the predicted clusters after PCA/K-Means
X_Kmeans_pca = vertebral_data_scaled_pca[ ['pc1','pc2','pc3', 'pc4']]
X_Kmeans_pca["predicted clusters"] = y_pred

sns.pairplot(X_Kmeans_pca, hue="predicted clusters")
plt.savefig('./FiguresCW2/Pair plot KMeans PCA.png',format='png')
plt.show()
```

5. Code for Decision Tree Classification

```
# Supervised Clustering - Decision Tree
#############################################################

from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split

# building a decision tree classifier model
classifier = DecisionTreeClassifier(random_state=0, criterion="gini")
# create a 80/20 train test split
x_train, x_test, y_train, y_test = train_test_split(scaled_features_df,
class_labels, test_size = 0.20, random_state = 0)
# training the classifier
classifier.fit(x_train, y_train)
# making predictions
y_pred = classifier.predict(x_test)

# Summary of the predictions made by the classifier
print("The classification report for Decision Tree :")
print(classification_report(y_test, y_pred))
print("-----------------------------------------------------")
# evaluate the quality of the predictions
```

```
print("\nThe confusion matrix for Decision Tree :")
print(confusion_matrix(y_test, y_pred))
print("-------------------------------------------------------")
# Accuracy score for the classifier model
print('\nThe accuracy for  Decision Tree is', accuracy_score(y_pred, y_test))

# plot the decision tree for visualisation
plt.figure(figsize = (70,70))
plot_tree(classifier, filled = True, fontsize = 16, feature_names =
vertebral_data.columns, class_names = ["AB", "NO"])
plt.savefig('./FiguresCW2/Decision Tree.png',format='png')
plt.show()
```

6. Code for Decision Tree Classification with PCA

```
# Decision Tree classification using data from PCA
# create a 80/20 train test split
x_train_pca, x_test_pca, y_train_pca, y_test_pca =
train_test_split(vertebral_data_scaled_pca, class_labels, test_size = 0.20,
random_state = 0)
# training the classifier
classifier.fit(x_train_pca, y_train_pca)
# making predictions
y_pred_pca = classifier.predict(x_test_pca)

# Summary of the predictions made by the classifier
print("The classification report for Decision Tree with PCA:")
print(classification_report(y_test_pca, y_pred_pca))
print("-------------------------------------------------------")
# evaluate the quality of the predictions
print("\nThe confusion matrix for Decision Tree with PCA :")
print(confusion_matrix(y_test_pca, y_pred_pca))
print("-------------------------------------------------------")
# Accuracy score for the classifier model
print('\nThe accuracy for  Decision Tree with PCA is', accuracy_score(y_pred_pca,
y_test_pca))
```

REFERENCES

1. Pechenizkiy, M., Tsymbal, A. and Puuronen, S. (2004). *PCA-based Feature Transformation for Classification: Issues in Medical Diagnostics*. Proceedings of the 17th IEEE Symposium on

Computer-Based Medical Systems (CBMS'04). [Online]. available at: https://www.win.tue.nl/~mpechen/publications/pubs/PechenizkiyCBMS04.pdf (Accessed: 14 March 2022).
2. Brendel, C. (2020). *Cluster-then-predict for classification tasks*, *Medium*. [Online]. available at: https://towardsdatascience.com/cluster-then-predict-for-classification-tasks-142fdfdc87d6 (Accessed: 16 March 2022).
3. Chakraborty, T. (2022). *EC3: Combining Clustering and Classification for Ensemble Learning*, *arXiv.org*. [Online]. available at: https://doi.org/10.48550/arXiv.1708.08591 (Accessed: 16 March 2022).