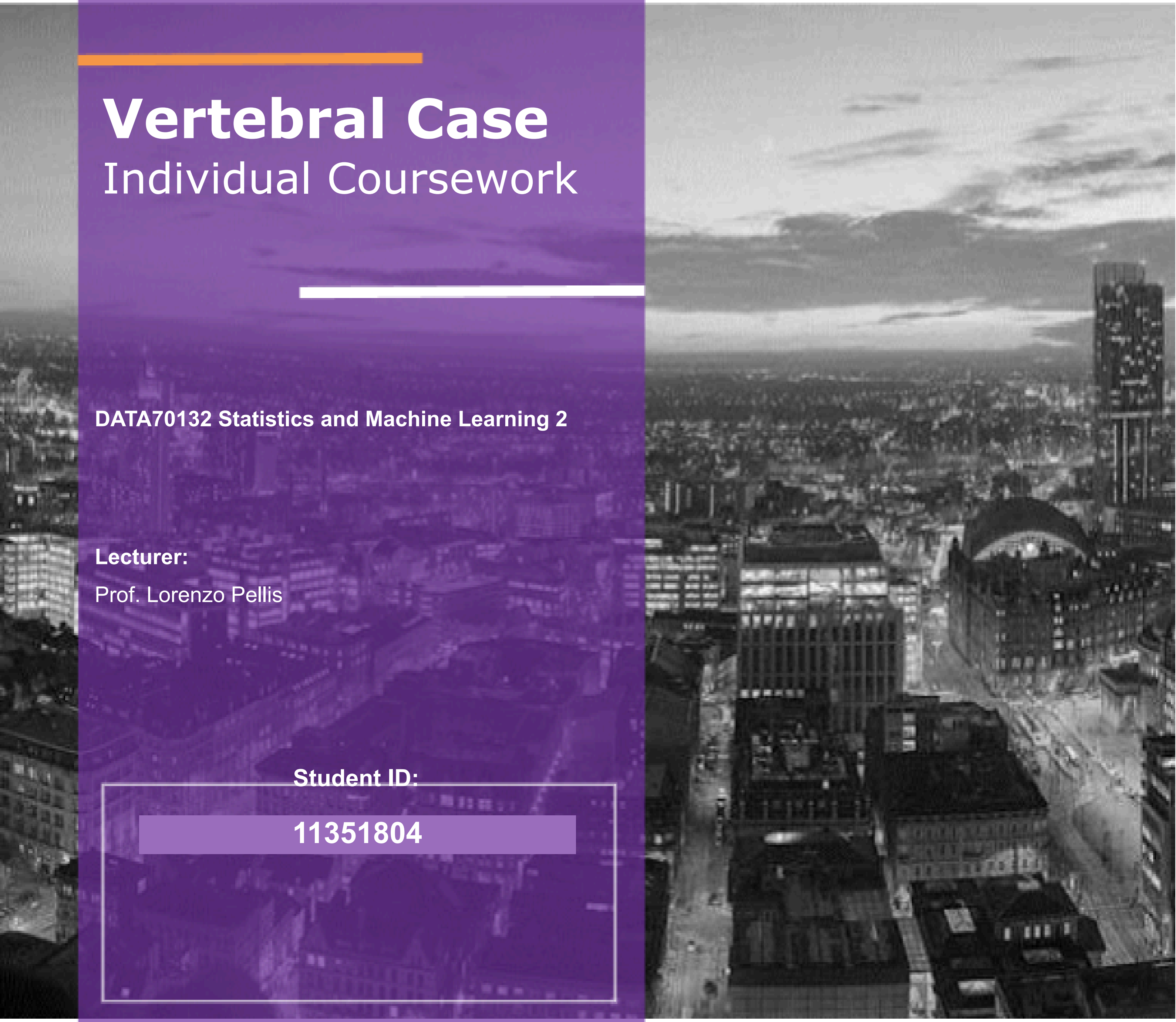
MANCHESTER
1824
The University of Manchester



Vertebral Case
Individual Coursework

DATA70132 Statistics and Machine Learning 2

Lecturer:
Prof. Lorenzo Pellis

Student ID:
11351804

2024

Chapter 1: Introduction

1.1 Background & Problem Statement

The Vertebral dataset is a collection of six vertebral features. The task is to build a machine learning model that can classify the condition of a patient based on the given features that can aid in the diagnosis and potentially improve patient outcomes.

1.2 Objective

Classify patients' condition based on the feature provided by employing the most appropriate EDA technique, data preprocessing, suitable unsupervised and supervised ML model, and evaluate performance metrics for accurate classification.

1.3 Dataset

The dataset consisting of 310 instances, provides six features as detailed explained below.

Table 1 Dataset Description

Column Name	Description
pelvic_incidence	This is a measure of the angle between the plane of the sacral plate at its midpoint and a line connecting this point to the femoral head axis.
pelvic_tilt	This is the angle between a line perpendicular to the sacral plate at its midpoint and a line connecting this point to the femoral head axis.
lumbar_lordosis_angle	This is the angle between the superior and inferior endplates of L5.
sacral_slope	This is the angle between the horizontal plane and the sacral plate.
pelvic_radius	This is the distance from the femoral head axis to the sacral promontory.
grade_of_spondylolisthesis	This is the degree of slippage
class label	This is the diagnosis of the patient. It can be AB (Abnormal) or NO (Normal)

Chapter 2: Methodology

2.1 Unsupervised Learning - Hierarchical Clustering

Hierarchical is an agglomerative method that requires a distance and linkage, it starts with each data point as a separate cluster and gradually combines them into bigger clusters. The process continues until all points are in a single cluster. According to Lance & Williams (1967), it can be represented as:

$$d(Ci, Cj) = \min d(x,y)$$
$$X \in Ci,\ y \in Cj$$

Where:
Ci and *Cj* are clusters,
d(*Ci*,*Cj*) is the distance between clusters *Ci* and *Cj*,
x and y are data points in clusters *Ci* and *Cj* respectively,
d(x,y) is the distance between data points x and y.

2.2 Supervised Learning - KNN

kNN identifies the ‘k’ nearest neighbours in the dataset for a new observation, assigning it to the class with the majority of votes among these neighbours. The prediction is determined through a voting mechanism, with the class receiving the most votes considered the predicted class. According to Hastie et al. (2009), it can be represented as:

$$Y(x) = \text{argmax}j \sum_{i \in Nk(x)} I\,(yi = j)$$

Where:
Y(x) is the predicted class for the new observation x.
k is the number of neighbours to consider.
Nk(x) is the set of k observations closest to x.
yi is the class label of the i-th observation in *Nk(x)*.
I(yi=j) is an indicator function that is 1 if *yi=j* and 0 otherwise.
argmax*j* means finding the class *j* that maximizes the sum.

2.3 Evaluation

In unsupervised learning, ARI, NMI, and Silhouette scores assess clustering similarity, while supervised learning uses accuracy, precision, recall, and F1 to gauge model performance (See Appendix 1 for metric measurement details).

Chapter 3: Exploratory Data Analysis

3.1 Summary statistics

Table 2 shows the ‘grade_of_spondylolisthesis’ attribute appears to have outliers. The distribution of ‘grade_of_spondylolisthesis’ and ‘pelvic_radius’ seem to be skewed, inferred from their mean and median. The anomaly of this variable will be further treated during the data preprocessing phase.

Table 2 Summary Statistic of dataset

Stat	pelvic incidence	pelvic tilt	lumbar lordosis angle	sacral slope	pelvic radius	grade of spondylolisthesis
count	310	310	310	310	310	310
mean	60.496484	17.542903	51.93071	42.953871	117.920548	26.296742
std	17.236109	10.00814	18.553766	13.422748	13.317629	37.558883
min	26.15	-6.55	14	13.37	70.08	-11.06
0.25	46.4325	10.6675	37	33.3475	110.71	1.6
0.5	58.69	16.36	49.565	42.405	118.265	11.765
0.75	72.88	22.12	63	52.6925	125.4675	41.285
max	129.83	49.43	125.74	121.43	163.07	418.54

3.2 Kernel Density Estimation Pairplot

KDE pairplot (Appendix 2) explains data distribution grouped based on class_label (the current patient’s diagnosis). The pairplot reveals that certain pairs of variables show a clear separation between the classes, suggesting their potential utility in classification tasks later.

3.3 Correlation Analysis

Figure 1 depicts positive correlations, such as between pelvic_incidence and sacral_slope. However, in classification models, highly correlated variables may lead to redundancy and multicollinearity, impacting model clustering. This concern will be addressed in the variable assessment process.

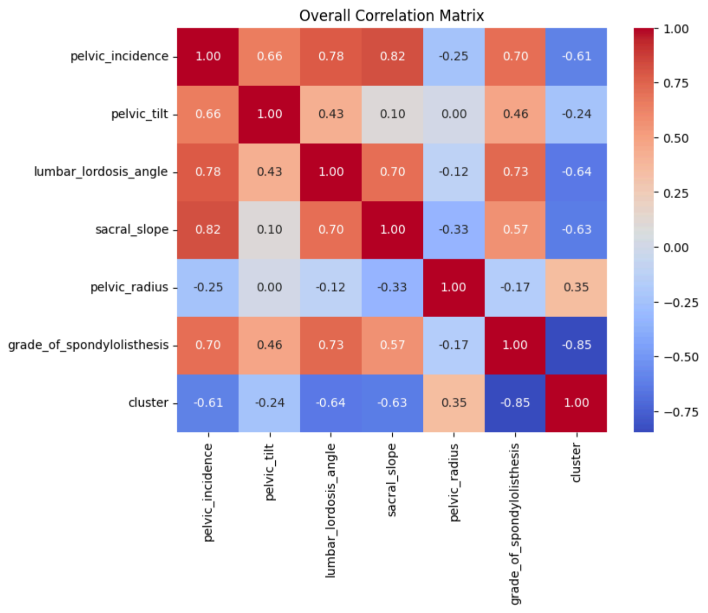


Figure 1 Correlation Analysis

Chapter 4: Data Preprocessing

4.1 Encode

Encoding is applied to the 'class_label' variable to convert categorical variables into integers, ensuring compatibility with scikit’s algorithms.

4.2 Outlier Detection and Removal

Local Outlier Factor (LOF) method performs the best preprocessing results due to its density-based outlier detection method. This outlier removal improves the performance of the clustering algorithm in the modelling stage and higher evaluation scores.

4.3 Split the dataset into Train & Validation sets (*only on Supervised mode*l)

The Train_df is divided into training (80%) and validation sets (20%) to facilitate robust model evaluation and prevent data leaking.

Chapter 5: Modelling, Prediction, and Discussion

5.1 Unsupervised Learning

5.1.1 Clusters and Variable Assessment

Utilising both elbow plot and dendrogram, the aim was to determine the optimal number of clusters. The elbow plot suggests an optimal range of 3-4 clusters. Similarly, the dendrogram highlights well-separated clusters at a distance between 200-300. Consequently, the optimal number of clusters is also in the range of 3-4.

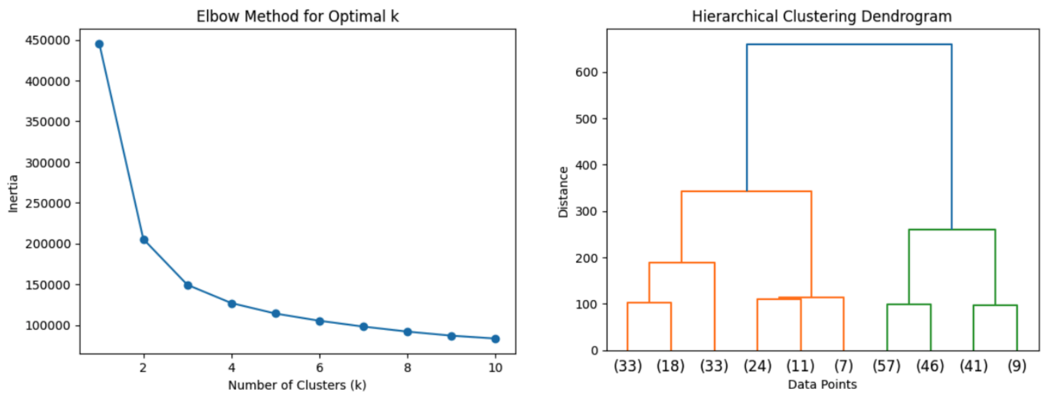


Figure 2 Elbow Plot (left) and Dendrogram Plot (right) to find the optimal k Cluster

Then, iteration over the range of 3-4 clusters and all of the variables are performed to get the most optimal variable combinations and a single cluster. The analysis reveals that three clusters and variable combinations appear as the most optimal settings for the modelling, as shown on Table 3.

Table 3 Best Cluster and Variable combination (Before vs After iteration)

	Before Iteration	After Iteration
Variables	pelvic_incidence, pelvic_tilt, lumbar_lordosis_angle, sacral_slope, pelvic_radius, grade_of_spondylolisthesis	pelvic_radius, grade_of_spondylolisthesis, lumbar_lordosis_angle
Number of Clusters	3 or 4	3

5.1.2 Prediction and Cross-validation of Unsupervised Model

Five K-fold Cross-validation is preferred over dataset splitting for Hierarchical modelling due to limited dataset. The trained features and parameters yield significant results. The model's Silhouette score indicates excellent performance, suggesting the identification of subtle patterns or additional patient conditions beyond 'Abnormal' and 'Normal' as shown on Table 4. Moreover, Figure 3 illustrates the robustness of the hierarchical model in determining underlying patterns compared to the true label.

Table 4 Metric Evaluation Hierarchical model with 5 k-fold CV

ARI	NMI	Silhouette
0.36	0.33	0.87

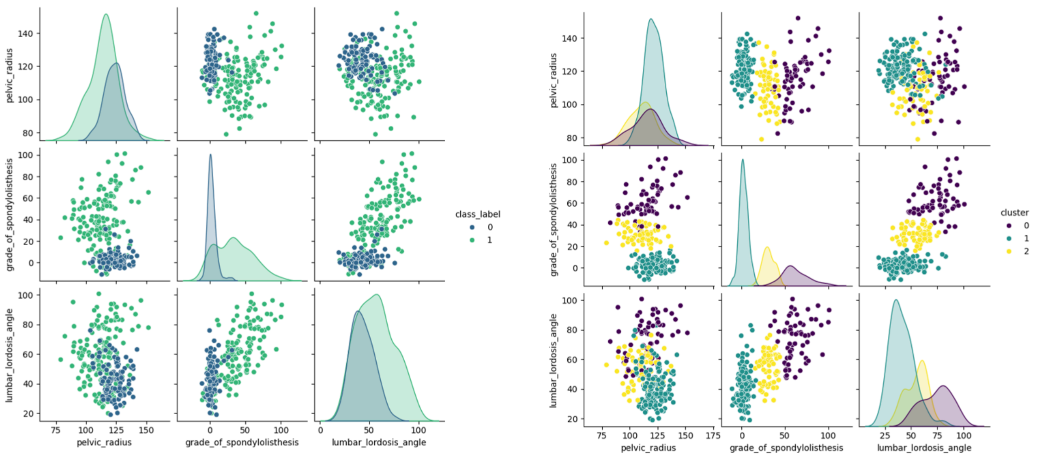


Figure 3 Pairplot of the true label (Left) and Pairplot from Hierarchical Clustering (Right)

5.2 Supervised Learning

5.2.1 Nearest Neighbours Assessment

The kNN model requires a number of nearest neighbours as the parameter to determine the ‘k’ that are closest to the new observation. Hence, iterative processes on the training and validation sets with different neighbours are done and plotted as Bias vs Variance tradeoff as shown on Figure 4. K value that yields the lowest validation error is ‘k=9’.

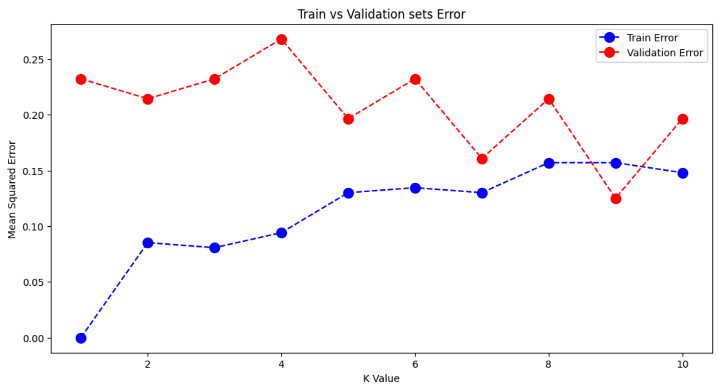


Figure 4 Plot of Bias vs Variance Tradeoff on Train and Validation set

5.2.2 Prediction of Supervised Model

The KNN model, with an optimal k of 9, demonstrated strong performance as shown on Table 5, these results highlight the model’s effective learning and generalisation capabilities on the validation set.

Table 5 Metric Evaluation of KNN model

	Train Set	Validation Set
Accuracy	0.856	0.892
Precision	0.857	0.892
Recall	0.856	0.892
F1	0.856	0.892

Moreover, Confusion Matrix on Figure 5 shows that the model achieved correct predictions for both classes, the Validation sets even shows only 3 misclassifications. The ROC curve for the validation sets on Figure 6 also illustrate a slightly higher performance than Train, suggesting that the model generalised well to unseen data.

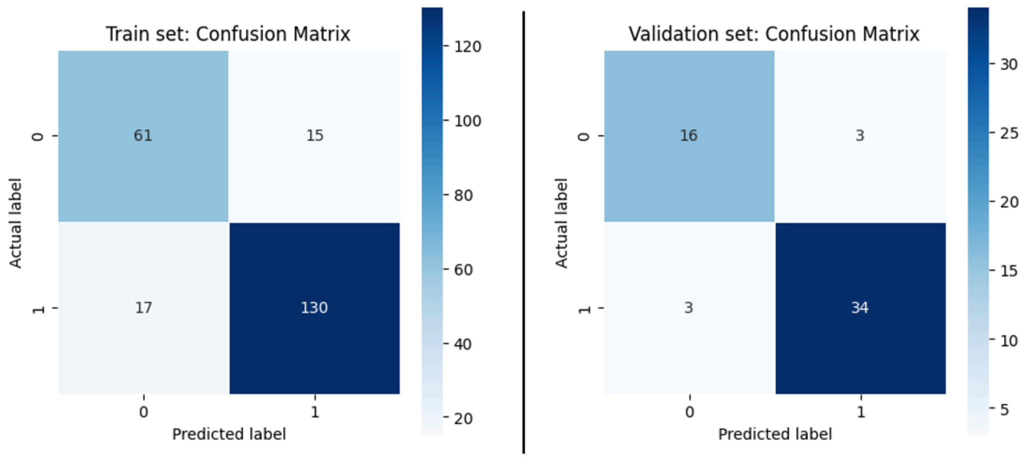


Figure 5 Confusion Matrix of Train (left) and Validation sets (right)

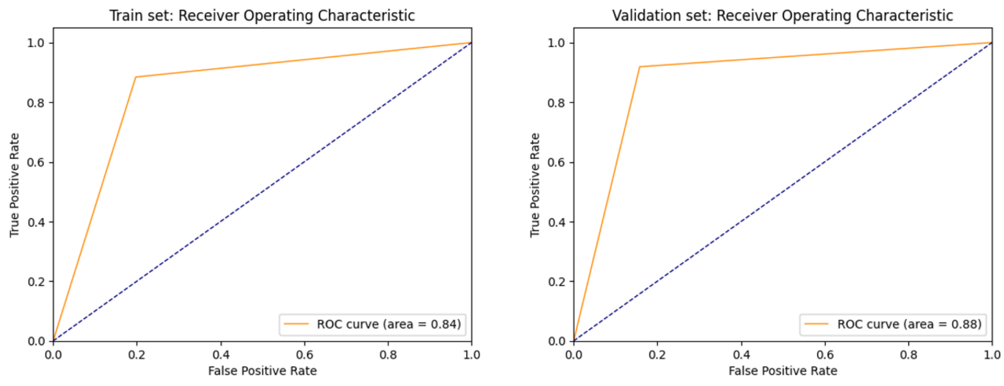


Figure 6 ROC Curve of Train (left) and Validation sets (right)

5.3 Discussion

The supervised model (kNN) and the unsupervised model (Hierarchical) can inform each other in several ways. The kNN, with its bias-variance tradeoff suggesting 9 neighbours, achieves a high accuracy on the both sets, this model's performance can be enhanced by the insights gained from the hierarchical model. The hierarchical suggests the optimal number of clusters to be 3 and recommends using only 3 out of the 6 available features. This information can be used to reduce the redundancy in the kNN, improving its performance. Then, a high silhouette score in the hierarchical model indicates well-separated clusters, which implies that the classes in the kNN model are also likely to be well-separated.

Chapter 6: Conclusion

This study successfully designed and implemented an unsupervised and supervised model, leveraging comprehensive EDA and data preprocessing. By employing a kFold cross-validation and clusters assessment, the Hierarchical model effectively determined optimal features and clusters, yielding promising results. While the kNN model, through a Bias vs Variance Tradeoff, can define the optimal k neighbours and achieve a higher accuracy on the unseen data. These findings have practical implications for accurate condition identification of vertebral-related illness in the future.

References

Lance, G. N., & Williams, W. T. (1967). A General Theory of Classificatory Sorting Strategies. Computer Journal, 9(4), 373–380.

Hastie, T., Tibshirani, R., & Friedman, J. (2009). The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer.

Syriopoulos, P. K., Kalampalikis, N. G., Kotsiantis, S. B., & Vrahatis, M. N. (2023). kNN Classification: a review. Annals of Mathematics and Artificial Intelligence1.

Cunningham, P., & Delany, S. J. (2020). k-Nearest Neighbour Classifiers: 2nd Edition (with Python examples)2.

Bayyapu, K. R., & Dolog, P. (2010). Tag and Neighbour Based Recommender System for Medical Events3.

Halkidi, M. (2014). Hierarchical Clustering. Encyclopaedia of Database Systems4.

Murtagh, F. (2014). Hierarchical Clustering. International Encyclopedia of Statistical Science5

Appendix

Appendix 1. Metric Evaluation for Unsupervised and Supervised ML Models

Metric	Formula
Accuracy	(TP + TN) / Total Population
Precision	TP / (TP + FP)
Recall	TP / (TP + FN)
F1-Score	(2 x Precision x Recall) / (Precision + Recall)

Definitions:

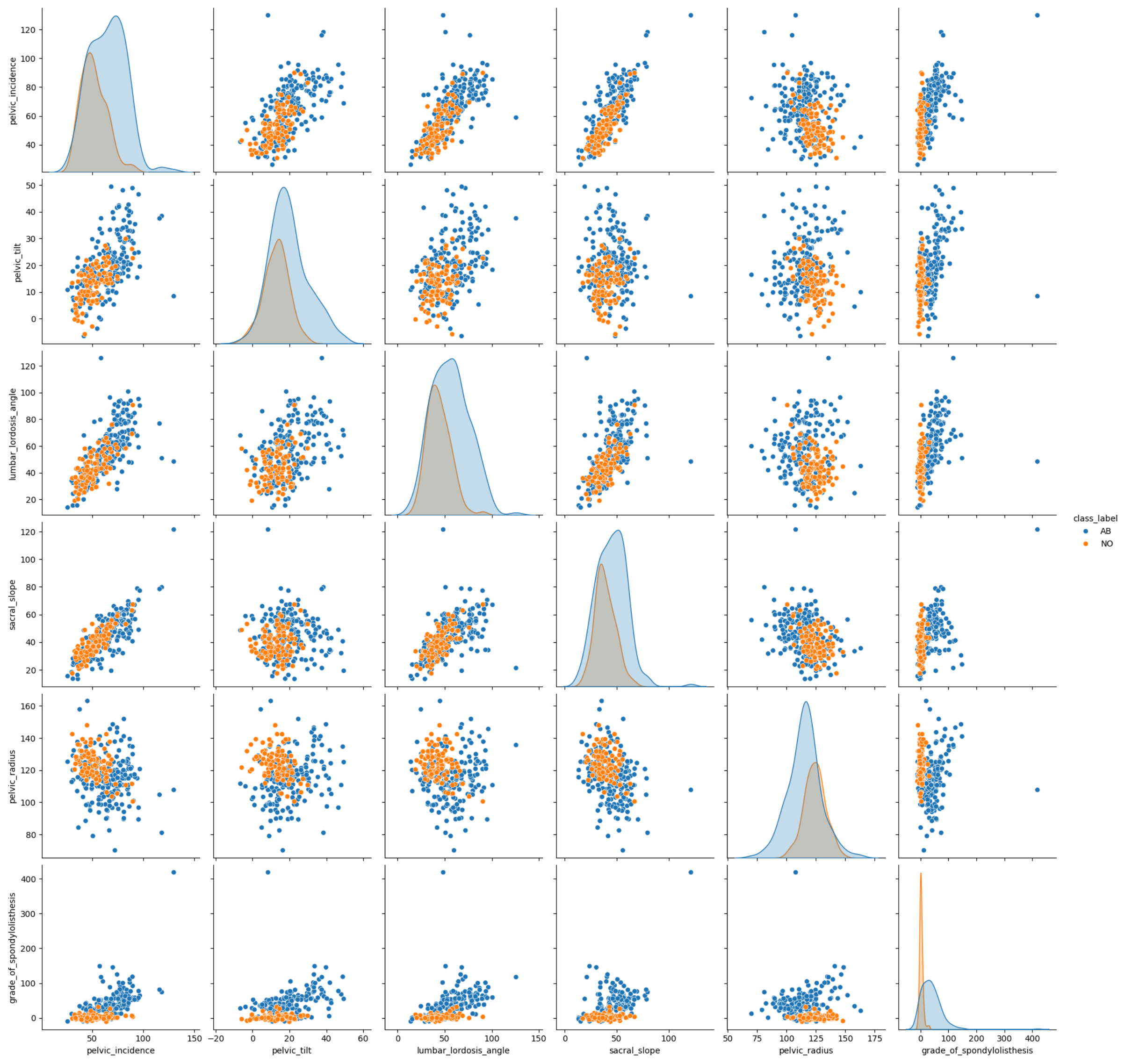
- True Positives (TP): Number of instances correctly predicted as positive.
- True Negatives (TN): Number of instances correctly predicted as negative.
- False Positives (FP): Number of instances incorrectly predicted as positive.
- False Negatives (FN): Number of instances incorrectly predicted as negative.

Metric	Formula
ARI	(RI - Expected_RI) / (max(RI_expected) - Expected_RI)
NMI	I(X;Y) / sqrt(H(X) * H(Y))
Silhouette	b-a / ((max(a,b))

Definitions:

- RI (Rand Index):| (TP + TN) / (TP + FP + FN + TN)
- Expected_RI: [((TP + FP) * (TP + FN)) + ((TN + FP) * (TN + FN))] / (TP + FP + FN + TN)^2
- I(X;Y) : Mutual Information between clusters X and Y
- H(X), H(Y): Entropy of clusters X and Y, respectively
- a: The mean distance between a sample and all other points in the same class.
- b: The mean distance between a sample and all other points in the next nearest cluster.

Appendix 2. Kernel Density Estimation Pairplot of Vertebral Dataset



1. Import Library

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import silhouette_score
from sklearn.metrics import calinski_harabasz_score
from sklearn.metrics import adjusted_rand_score, normalized_mutual_info_score
from sklearn.metrics import make_scorer, accuracy_score, precision_score, recall_
from scipy.stats import zscore
from sklearn.cluster import AgglomerativeClustering
from sklearn.mixture import GaussianMixture
from sklearn.pipeline import make_pipeline
from sklearn import preprocessing
from sklearn.model_selection import cross_validate
from sklearn.metrics import mean_squared_error
from math import sqrt
import sklearn as sk
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
```

2. Dataset Loading & Summary Statistics

```
#Read dataset and checking the info and shape
file_path = '/content/drive/MyDrive/0. ML Stats Sem2/vertebral_column_data.txt'

# Read the text file into a DataFrame
df = pd.read_csv(file_path, sep=' ', header=None)

# Assign the column name as assigned
new_header = ['pelvic_incidence', 'pelvic_tilt', 'lumbar_lordosis_angle', 'sacral
df.columns = new_header
df.head(5)
df.info()
df.shape
```

```
#Summary statistics of verterbal dataset
df.describe()
```

```
df.head(2)
```

3. Exploratory Data Analysis

```
#Kernel Density Estimation Plot
sns.pairplot(df, hue="class_label", size=3, diag_kind="kde")
```

```
#Correlation Plot
# Drop 'class_label' column before creating the correlation matrix
df_without_class = df.drop(['class_label'], axis=1)

# Create a correlation matrix
correlation_matrix = df_without_class.corr()

# Plot the correlation matrix using a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Overall Correlation Matrix')
plt.show()
```

4. Data Preprocessing

4.1 Encode

```
#Transform the categorical column of class_label into numerical where AB (Abnorma
class_label_mapping = {'AB': 1, 'NO': 0}

try:
    df['class_label'] = df['class_label'].map(class_label_mapping)
    print('Encode done without error!')
except Exception as e:
    print(f'Error during encoding: {e}')
```

Outlier Detection & Removal (Local Outlier Factor LOF)

```
from sklearn.neighbors import LocalOutlierFactor

# Initialize the Local Outlier Factor model
lof = LocalOutlierFactor(n_neighbors=20, contamination=0.1)

# Fit the model and predict the outliers
df['outlier'] = lof.fit_predict(df)

# LOF labels outliers as -1
outliers = df['outlier'] == -1

# Remove outliers
df = df[~outliers]

# Drop the outlier column
df.drop('outlier', axis=1, inplace=True)
```

```
df.info()
df.shape
```

5. Unsupervised Learning - Hierarchical Clustering

5.1 Elbow Plot

```
# Choose a range of cluster numbers (k)
k_values = range(1, 11)
```

```
# Fit KMeans models for each k and store inertia values
inertia_values = []
for k in k_values:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(df)
    inertia_values.append(kmeans.inertia_)
```

```
# Plot the elbow curve
plt.plot(k_values, inertia_values, marker='o')
plt.title('Elbow Method for Optimal k')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia')
plt.show()
```

5.2 Modelling - Hierarchical

```
from itertools import combinations

all_features = ['pelvic_tilt', 'pelvic_radius', 'grade_of_spondylolisthesis', 'lu

# Initialize variables to track the best scores
best_ari = -1
best_nmi = -1
best_silhouette = -1
best_features = None
best_num_clusters = None

# Define the number of splits for K-Fold cross-validation
n_splits = 5
kf = KFold(n_splits=n_splits)

# Iterate over different combinations of features
for num_features in range(1, len(all_features) + 1):
    for feature_combination in combinations(all_features, num_features):
        selected_features = list(feature_combination)

        # Standardize the data by mean and variance (True)
        scaler = StandardScaler(with_std=False)
        df_scaled = scaler.fit_transform(df[selected_features])

        # Initialize scores for cross-validation
        ari_scores = []
        nmi_scores = []
        silhouette_scores = []

        # Apply cross-validation
        for train_index, test_index in kf.split(df_scaled):
            # Split the data into train and test sets
            df_scaled_train, df_scaled_test = df_scaled[train_index], df_scaled[t
            df_train, df_test = df.iloc[train_index], df.iloc[test_index]

            # Iterate over different cluster numbers
            for num_clusters in range(2, 4):
                # Apply Hierarchical clustering
                hierarchical = AgglomerativeClustering(n_clusters=num_clusters)
                labels_train = hierarchical.fit_predict(df_scaled_train)
                labels_test = hierarchical.fit_predict(df_scaled_test)

                # Evaluate clustering performance for train set
                ari_score_train = adjusted_rand_score(df_train['class_label'], la
                nmi_score_train = normalized_mutual_info_score(df_train['class_la
                silhouette_score_train = silhouette_score(df_scaled_train, labels

                # Evaluate clustering performance for test set
                ari_score_test = adjusted_rand_score(df_test['class_label'], labe
                nmi_score_test = normalized_mutual_info_score(df_test['class_labe
                silhouette_score_test = silhouette_score(df_scaled_test, labels_t

                # Append scores to lists
                ari_scores.append(ari_score_test)
                nmi_scores.append(nmi_score_test)
                silhouette_scores.append(silhouette_score_test)

        # Calculate average scores
        avg_ari = sum(ari_scores) / n_splits
        avg_nmi = sum(nmi_scores) / n_splits
        avg_silhouette = sum(silhouette_scores) / n_splits

        # Update best scores if the current combination is better
        if avg_ari > best_ari:
            best_ari = avg_ari
            best_nmi = avg_nmi
            best_silhouette = avg_silhouette
            best_features = selected_features
            best_num_clusters = num_clusters

# Print the best results
print("Best Features:", best_features)
print("Best Number of Clusters:", best_num_clusters)
print("Best Average Adjusted Rand Index (ARI):", best_ari)
print("Best Average Normalized Mutual Information (NMI):", best_nmi)
print("Best Average Silhouette Score:", best_silhouette)
```

```
from scipy.cluster.hierarchy import linkage, dendrogram

# Apply Hierarchical clustering with the best number of clusters and features
hierarchical_best = AgglomerativeClustering(n_clusters=best_num_clusters)
labels_best = hierarchical_best.fit_predict(df_scaled)

# Create a linkage matrix
linkage_matrix = linkage(df_scaled, method='ward')

# Plot the dendrogram
dendrogram(linkage_matrix, labels=df.index, orientation='top', distance_sort='des
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('Data Points')
plt.ylabel('Distance')
plt.show()
```

```
# Plot the dendrogram with limited x-axis labels
dendrogram(linkage_matrix, labels=df.index, orientation='top', distance_sort='des
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('Data Points')
plt.ylabel('Distance')
plt.show()
```

```
# Assign cluster labels to data points using the best hierarchical clustering mod
df['cluster_label'] = hierarchical_best.fit_predict(df_scaled)

# Display a summary table
summary_table = df.groupby('cluster_label').apply(lambda x: ', '.join(map(str, x.
summary_table
```

5.3 Visualize

```
selected_features = ['pelvic_radius', 'grade_of_spondylolisthesis', 'lumbar_lordo

# Standardize the data by mean and variance (True)
scaler = StandardScaler(with_std=False)
df_scaled = scaler.fit_transform(df[selected_features])

# Apply Hierarchical clustering
hierarchical = AgglomerativeClustering(n_clusters=3)
df['cluster'] = hierarchical.fit_predict(df_scaled)

# Pair plot for actual labels
plt.figure(figsize=(10,10))
plt.suptitle('Pair Plot for Actual Labels', fontsize=20)
sns.pairplot(df, hue='class_label', palette='viridis', vars=selected_features)
plt.show()

# Pair plot for predicted labels
plt.figure(figsize=(10,10))
plt.suptitle('Pair Plot for Predicted Labels', fontsize=20)
sns.pairplot(df, hue='cluster', palette='viridis', vars=selected_features)
plt.show()
```

6. Supervised Learning - kNN

6.1 Bias vs Variance Tradeoff Plot

```
from sklearn.metrics import mean_squared_error

k_range = range(1, 11)

# Lists to store errors
train_errors = []
test_errors = []

# Loop over k values
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    clf = make_pipeline(preprocessing.StandardScaler(), knn)

    # Fit the model on the training data and predict on both train and test sets
    clf.fit(X_train, y_train)
    y_train_pred = clf.predict(X_train)
    y_test_pred = clf.predict(X_test)

    # Calculate the mean squared error for the train and test sets
    train_error = mean_squared_error(y_train, y_train_pred)
    test_error = mean_squared_error(y_test, y_test_pred)

    # Append to the error lists
    train_errors.append(train_error)
    test_errors.append(test_error)

# Plot
plt.figure(figsize=(12, 6))
plt.plot(k_range, train_errors, color='blue', linestyle='dashed', marker='o',
         markerfacecolor='blue', markersize=10, label='Train Error')
plt.plot(k_range, test_errors, color='red', linestyle='dashed', marker='o',
         markerfacecolor='red', markersize=10, label='Validation Error')
plt.title('Train vs Validation sets Error')
plt.xlabel('K Value')
plt.ylabel('Mean Squared Error')
plt.legend()
plt.show()

# Create a DataFrame from the errors
error_df = pd.DataFrame({'k': k_range, 'Train Error': train_errors, 'Validation E

# Print the DataFrame
print(error_df)
```

6.2 Modelling - KNN

```
# Assuming that df is your DataFrame and 'class_label' is your target column
X = df.drop('class_label', axis=1)
y = df['class_label']

# Split the data into 80% training and 20% testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_s

best_k = 1
best_score = 0

for k in range(1, 11):
    # Create a kNN classifier with k neighbors
    knn = KNeighborsClassifier(n_neighbors=k)

    # Fit the classifier to the training data
    knn.fit(X_train, y_train)

    # Predict the labels of the test set
    y_pred = knn.predict(X_test)

    # Calculate the F1 score
    f1 = f1_score(y_test, y_pred, average='weighted')

    # Update best_k and best_score if the current model is better
    if f1 > best_score:
        best_k = k
        best_score = f1

# Print the best k and best score
print("Best k:", best_k)
print("Best Test F1 Score:", best_score)

# Fit the best model to the training data
knn = KNeighborsClassifier(n_neighbors=best_k)
knn.fit(X_train, y_train)

# Predict the labels of the test set
y_pred = knn.predict(X_test)

# Predict the labels of the training set
y_train_pred = knn.predict(X_train)

# Print the accuracy, precision, recall, and F1 score for the training set
print("Training Accuracy:", accuracy_score(y_train, y_train_pred))
print("Training Precision:", precision_score(y_train, y_train_pred, average='weig
print("Training Recall:", recall_score(y_train, y_train_pred, average='weighted')
print("Training F1 Score:", f1_score(y_train, y_train_pred, average='weighted'))

# Print the accuracy, precision, recall, and F1 score for the test set
print("Test Accuracy:", accuracy_score(y_test, y_pred))
print("Test Precision:", precision_score(y_test, y_pred, average='weighted'))
print("Test Recall:", recall_score(y_test, y_pred, average='weighted'))
print("Test F1 Score:", f1_score(y_test, y_pred, average='weighted'))
```

6.2 Confusion Matrix and ROC Plot

Confusion Matrix on Train set

```
y_train_pred = knn.predict(X_train)

# Print confusion matrix for the training set
cm_train = confusion_matrix(y_train, y_train_pred)
print("Confusion Matrix (Train Set):")
print(cm_train)

# Compute ROC curve for the training set
fpr_train, tpr_train, thresholds_train = roc_curve(y_train, y_train_pred)
roc_auc_train = auc(fpr_train, tpr_train)

# Print ROC curve data for the training set
print("\nROC Curve Data (Train Set):")
print("False Positive Rate:", fpr_train)
print("True Positive Rate:", tpr_train)
print("Thresholds:", thresholds_train)
print("Area Under Curve:", roc_auc_train)
```

```
from sklearn.metrics import confusion_matrix, roc_curve, auc

cm = confusion_matrix(y_train, y_train_pred)

# Plot confusion matrix
plt.figure(figsize=(5,5))
sns.heatmap(cm, annot=True, fmt=".0f", linewidths=.5, square = True, cmap = 'Blue
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
plt.title('Train set: Confusion Matrix')  # Added title here

# Compute ROC curve
fpr, tpr, thresholds = roc_curve(y_train, y_train_pred)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=1, label='ROC curve (area = %0.2f)' % r
```