Université du Québec à Rimouski

Assignment 2

- Data Mining

Professor – Mehdi Adda, Ph. D.

- Mohammadi, Samira
- Noktehdan, Hamed

Table of Contents

Introduction & Contexte 3
General problem & Objectives 3
Function of Reduction Algorithms and their applications(ICA & LDA) 3
Function of Classification Algorithms and their applications(SVM & Random Forest) 4
Function of Clustering Algorithms and their applications 5
Data 6
Codes 7
Results 23
Contributors in this project 24
Conclusion 24
References 26

Introduction & Contexte

Heart disease is a major public health problem that causes significant morbidity and mortality worldwide. Identifying individuals at risk for heart disease and developing effective prevention and management strategies are critical public health priorities. In recent years, data mining and machine learning techniques have been increasingly used to analyze medical data and develop predictive models for identifying individuals at risk for heart disease.[1]

Context: In this project, we aim to apply data mining techniques to analyze a cardiovascular dataset and predict the presence of heart disease. The dataset we will use, called "processed.cleveland.data," was provided by the Cleveland Clinic. It is publicly available. The dataset contains 14 attributes related to heart disease. It has 5 integers as output indicating the severity of heart disease. [2]

General problem & Objectives

The project will be divided into two main parts: classification and clustering. In the classification part, we will split the dataset into training and testing sets and train two classification techniques to predict the diagnosis of heart disease using the 14 attributes of the dataset. We will evaluate the performance of the classifiers using appropriate evaluation metrics and refine the best classifiers using hyperparameter optimization. Specifically, we will use support vector machine (SVM) and random forest (RF) classifiers.

In the clustering part, we will apply two clustering techniques to group the data into different clusters and analyze the characteristics of each cluster. We will determine the optimal number of clusters and explore potential patterns and trends in the data. We will also assess the stability and reproducibility of clustering results and explore different combinations of data reduction and clustering techniques. Specifically, we will use k-means and fuzzy clustering algorithms.

Overall, this project aims to demonstrate the potential of data mining techniques for predicting and analyzing heart disease and contribute to ongoing efforts to improve heart disease prevention and management. By analyzing the processed.cleveland.data dataset using classification and clustering techniques, we hope to identify relevant characteristics for classification and clustering, explore the performance of different classifiers, and identify potential patterns and trends in the data.

Function of Reduction Algorithms and their applications(ICA & LDA)

Reduction algorithms are commonly used in machine learning and data analysis to reduce the dimensionality of datasets while preserving important information. Two commonly used reduction algorithms are Independent Component Analysis (ICA) and Linear Discriminant Analysis (LDA).

ICA is a statistical technique used to separate a multivariate signal into independent, non-Gaussian signals. ICA is used in a variety of applications, including signal processing, image processing, and neuroscience. In signal processing, ICA is used to separate mixed signals into their original sources. For example, in speech processing, ICA can be used to separate a mixed audio signal into the individual speech signals of different speakers.

LDA is a supervised learning algorithm that is used for classification tasks. LDA finds a linear combination of features that separates the data into different classes, and then uses this combination to classify new

instances into one of the predefined categories. Therefore, LDA is a classification algorithm that works by reducing the dimensionality of the data, finding the most discriminative features, and classifying new instances based on these features.[4]

In summary, reduction algorithms such as ICA and LDA are important tools in machine learning and data analysis. They are used to reduce the dimensionality of datasets and extract important features, which can then be used for various applications such as signal processing, image processing, and pattern recognition.

Function of Classification Algorithms and their applications(SVM & Random Forest)

Classification algorithms are a type of machine learning algorithms that are used to classify data into different categories or classes based on certain features. Two commonly used classification algorithms are Support Vector Machines (SVM) and Random Forest.[5]

Support Vector Machine (SVM) is a supervised learning algorithm that is used for classification, regression, and outlier detection. SVM is a linear model for classification that works by finding the best hyperplane that separates the data into different classes. The hyperplane is chosen so that it maximizes the margin between the classes, which means that it is the furthest distance from the closest data points from both classes.

In SVM, the data points are mapped to a high-dimensional space where a separating hyperplane can be found. This mapping is done by a kernel function, which transforms the input data into a higher dimensional space without actually computing the coordinates of the data in that space. This technique is called the "kernel trick," and it allows SVM to work efficiently in high-dimensional spaces.

SVM works by finding the support vectors, which are the data points that lie closest to the decision boundary or the hyperplane. These support vectors are used to define the margin, which is the distance between the decision boundary and the closest data points from both classes. The SVM algorithm then finds the hyperplane that maximizes this margin, which means that it is the most robust and generalized decision boundary.

SVM has several advantages, such as being able to handle high-dimensional data, being able to work with both linear and non-linear data, and being robust to noise and outliers. SVM is commonly used in various domains, such as image processing, natural language processing, bioinformatics, and finance.

In summary, SVM is a powerful and versatile algorithm that is used for classification, regression, and outlier detection. It works by finding the best hyperplane that separates the data into different classes, and it is able to handle high-dimensional and non-linear data.[6]

Random Forest is a supervised learning algorithm that is used for classification, regression, and feature selection. It is an ensemble learning method that consJtructs a multitude of decision trees at training time and outputs the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees.

Each decision tree is built using a random subset of the features and a random subset of the data. This is called "bagging," which stands for bootstrap aggregating. Bagging reduces overfitting by creating multiple training sets and training multiple models on these sets. The output of the random forest is then obtained by averaging the outputs of individual trees (in regression) or taking a majority vote (in classification).

Random Forest is a supervised learning algorithm that is used for classification, regression, and feature selection. It is an ensemble learning method that constructs a multitude of decision trees at training time and outputs the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees.

Each decision tree is built using a random subset of the features and a random subset of the data. This is called "bagging," which stands for bootstrap aggregating. Bagging reduces overfitting by creating multiple training sets and training multiple models on these sets. The output of the random forest is then obtained by averaging the outputs of individual trees (in regression) or taking a majority vote (in classification).

Random Forest has several advantages, such as being able to handle missing values, being robust to noise and outliers, and being able to perform feature selection. Random Forest is commonly used in various domains, such as image processing, natural language processing, bioinformatics, and finance.

In summary, Random Forest is a powerful and versatile algorithm that is used for classification, regression, and feature selection. It works by constructing a multitude of decision trees using a random subset of the features and a random subset of the data, and it reduces overfitting by averaging the outputs of individual trees (in regression) or taking a majority vote (in classification).[7]

Overall, classification algorithms such as SVM and Random Forest are important tools in machine learning and data analysis. They are used to classify data into different categories based on a set of features, and can be applied in various domains such as pattern recognition, image processing, data mining, and bioinformatics.

Function of Clustering Algorithms and their applications

K-means and fuzzy clustering are two popular techniques used in data clustering, which is the process of grouping similar objects or data points together based on certain characteristics.

K-means clustering is a simple and efficient algorithm that partitions a set of data points into K clusters, where K is a pre-defined number chosen by the user. The algorithm starts by randomly selecting K cluster centers, and then iteratively assigns each data point to the nearest center based on the Euclidean distance between them. After each assignment, the center of each cluster is updated to the mean of all the data points assigned to it. This process continues until the assignments of the data points no longer change significantly.[8]

Fuzzy clustering, also known as fuzzy c-means clustering, is a soft clustering technique that assigns each data point to multiple clusters, with varying degrees of membership. Unlike k-means, which assigns each

data point to a single cluster, fuzzy clustering allows data points to belong to multiple clusters simultaneously, with membership values ranging from 0 to 1. The algorithm starts by randomly initializing membership values for each data point and cluster centers, and then iteratively updates these values based on the distance between the data points and the cluster centers. After each update, the cluster centers are recomputed as weighted averages of the data points, where the weights are the membership values.[9]

Both k-means and fuzzy clustering have their strengths and weaknesses, and the choice between them depends on the specific problem and the characteristics of the data being clustered. K-means is faster and easier to implement, but can be sensitive to the initial choice of cluster centers and may not work well with data that has overlapping clusters. Fuzzy clustering is more flexible and can handle overlapping clusters, but requires more computation and may not be as interpretable as k-means.

Data

The "processed.cleveland.data" dataset is a collection of medical data related to heart disease. The data was processed and provided by the Cleveland Clinic Foundation in 1988, and is commonly used as a benchmark dataset in machine learning and data analysis projects related to heart disease.

The dataset contains 14 attributes, including age, sex, chest pain type, resting blood pressure, serum cholesterol level, fasting blood sugar level, resting electrocardiographic results, maximum heart rate achieved, exercise-induced angina, ST segment depression, the slope of the peak exercise ST segment, the number of major vessels colored by fluoroscopy, and a binary variable indicating the presence or absence of heart disease.

Each attribute has a range of possible values, and some of the attributes are discrete while others are continuous. For example, the chest pain type attribute can take on values of 1, 2, 3, or 4, indicating different types of chest pain, while the maximum heart rate achieved attribute can take on a continuous range of values.

The data is often used to train and test machine learning models to predict the presence or absence of heart disease based on a patient's medical information. It is important to note that the dataset is relatively old and may not reflect current medical practices or demographic distributions.[2]

Codes

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split , GridSearchCV
from sklearn.decomposition import FastICA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

from sklearn import svm
from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy_score ,f1_score, confusion_matrix

from sklearn.metrics import classification_report

from sklearn.model_selection import cross_val_predict, KFold

from sklearn.neighbors import KNeighborsClassifier
from imblearn.over_sampling import RandomOverSampler
```

Fig1.

(Fig1) -> Importing necessary libraries

```
df = pd.read_csv(filepath_or_buffer="processed.cleveland.data",header=None)



df.columns = ['age', 'sex', 'cp',
                  'trestbps', 'chol', 'fbs',
                  'restecg','thalach','exang','oldpeak',
                  'slope', 'ca' , 'thal' ,'num']

df.head()
```

Fig2.

##(Fig2) -> Loading data and defining header

```
# Replace all instances of "?" with NaNs
for col in df.columns:
    df[col] = df[col].replace("?", np.nan)

# Drop rows with missing values
df = df.dropna()
```

Fig3.

##(Fig3) -> Removing missing values

```
# Group cleaned DataFrame by a column
grouped_df = df.groupby('num')

# Illustrate the categories
grouped_df.size().plot(kind='bar', figsize=(8, 6), title='Category Distribution')
```

```
<Axes: title={'center': 'Category Distribution'}, xlabel='num'>
```

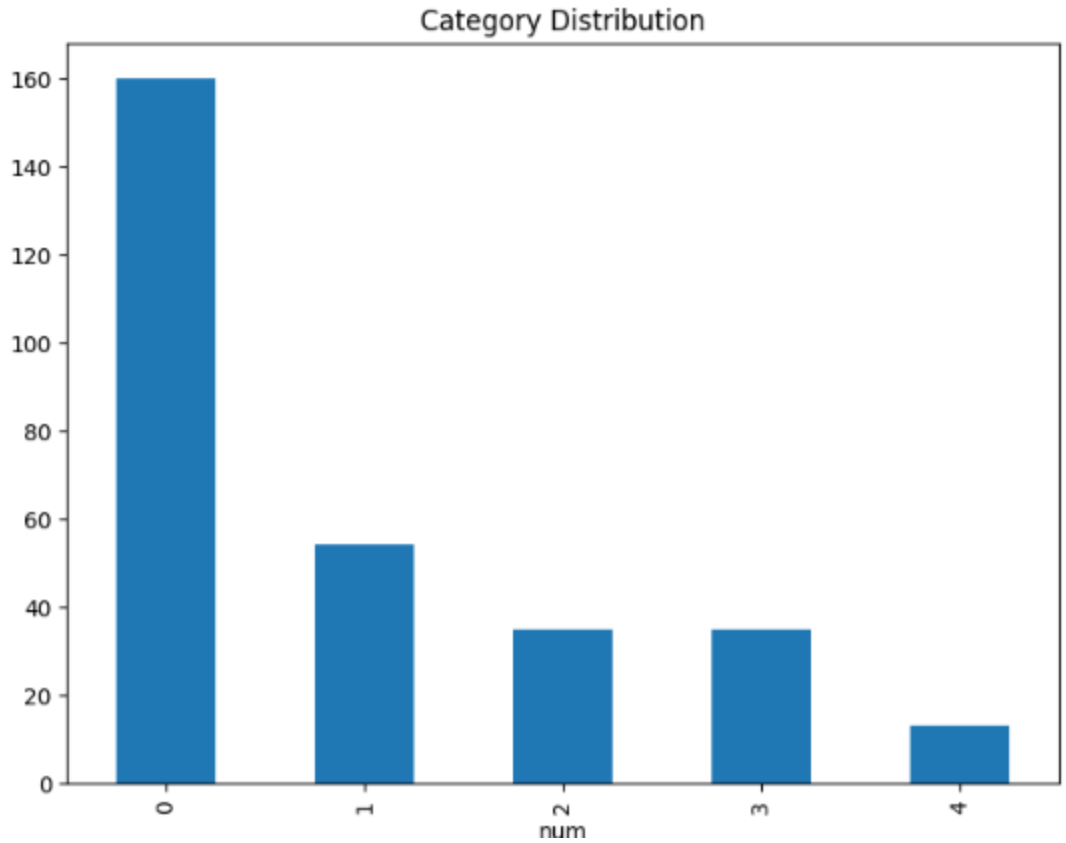

Fig4.

##(Fig4) -> The code groups the DataFrame df by the column 'num', counts the number of instances in each group, and creates a bar plot to show the distribution of categories. Each bar in the plot represents a unique value in the 'num' column, and the height of the bar represents the count of instances for that value.

```
input_parameters = ['age', 'sex', 'cp','trestbps', 'chol', 'fbs','restecg','thalach','exang','oldpeak','slope', 'ca' , 'thal']
output_parameter = ['num']
```

```
x=df[input_parameters]
y= df[output_parameter]
```

Fig5.

##(Fig5)-> First, input_parameters and output_parameter are defined as lists of column names. input_parameters contains the names of columns that will be used as input variables in some model, and output_parameter contains the name of the column that will be used as the output variable.

Next, x is assigned to the subset of df that contains only the columns listed in input_parameters, and y is assigned to the subset of df that contains only the column listed in output_parameter.

The resulting x and y data can be used for further analysis, such as training a machine learning model to predict the value of the output variable (num) based on the input variables (age, sex, cp, etc.).

```
## Define oversampler
from imblearn.over_sampling import SMOTE

# Initialize the SMOTE oversampler
oversampler = SMOTE(random_state=42)

# Resample the data to balance the classes
X, Y = oversampler.fit_resample(x, y)
```

Fig6.

(Fig6) -> This code defines and applies the SMOTE oversampling method to resample the data and balance the classes. Here's a breakdown of the code: First, the SMOTE class is imported from the imblearn.over_sampling module.Next, the oversampler object is created by initializing an instance of the SMOTE class with a random state of 42.Finally, the fit_resample() method of the oversampler object is called with the input data x and target variable y. This method resamples the data to balance the classes by generating synthetic samples of the minority class using the SMOTE algorithm, and returns the resampled feature matrix and target vector as X and Y, respectively.

```
# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=42)
```

Fig7.

(Fig7) -> Splitting data into training and test sets

```
# apply ICA to the training data
ica = FastICA(n_components=10)
X_train_ica = ica.fit_transform(X_train)

# apply ICA to the test data
X_test_ica = ica.transform(X_test)
```

Fig 8.

(Fig 8) -> In this code, Independent Component Analysis (ICA) is applied to the input data for feature extraction. ICA is a linear transformation technique that separates a multivariate signal into independent, non-Gaussian components.

```
# get the number of features in the dataset
num_features = df.shape[1] - 1

# get the number of unique class labels
num_classes = len(df['num'].unique())

# calculate the maximum number of components allowed by LDA
max_components = min(num_features, num_classes - 1)


print(max_components)
```

```
4
```

```
# apply LDA to the training data
lda = LinearDiscriminantAnalysis(n_components = max_components)
X_train_lda = lda.fit_transform(X_train, y_train)

# apply LDA to the test data
X_test_lda = lda.transform(X_test)
```

Fig9.

(Fig9) -> This code performs Linear Discriminant Analysis (LDA) on a dataset. It first calculates the number of features and unique class labels in the data. Then, it determines the maximum number of components that LDA can extract. The LDA model is fitted to the training data using the maximum number of components, and the training data is transformed to a lower-dimensional space. Finally, the test data is also transformed to the same space using the same LDA model.

```
##SVM with hyper_parameters Without reduction
param_grid = {'C': [0.1, 1, 10, 100], 'gamma': [0.001, 0.01, 0.1, 1], 'kernel': ['linear', 'rbf']}

# Initialize the SVM classifier
svm_clf = svm.SVC()

# Define the grid search cross-validation strategy
cv = GridSearchCV(svm_clf, param_grid, cv=10)

# Fit the model to the training data using grid search cross-validation
cv.fit(X_train, y_train)

# Print the best hyperparameters
print("Best hyperparameters:", cv.best_params_)

# Fit the model to the training data using the best hyperparameters
svm_clf_best = svm.SVC(C=cv.best_params_['C'], gamma=cv.best_params_['gamma'], kernel=cv.best_params_['kernel'])
svm_clf_best.fit(X_train, y_train)

# Predict the classes of the test set
y_pred = svm_clf_best.predict(X_test)

# Compute the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy:', accuracy)

# Compute the classification report and confusion matrix
report = classification_report(y_test, y_pred)
print('Classification Report:\n', report)

matrix = confusion_matrix(y_test, y_pred)
print('Confusion Matrix:\n', matrix)
```

Fig10.

(Fig10) -> This code performs Support Vector Machine (SVM) classification on a dataset using hyperparameter tuning. It defines a grid of hyperparameters, including the values of regularization parameter C, kernel coefficient gamma, and kernel type (linear or radial basis function).

The svm.SVC() function from the sklearn.svm module is used to initialize an SVM classifier. Then, the GridSearchCV function from the sklearn.model_selection module is used to define a cross-validation strategy that searches for the best hyperparameters in the grid. The fit() method is used to fit the model to the training data using grid search cross-validation.

After obtaining the best hyperparameters, a new SVM classifier is initialized using the best hyperparameters. The fit() method is used again to fit this new classifier to the training data. The predict() method is used to predict the classes of the test set, and the accuracy_score(), classification_report(), and confusion_matrix() functions from the sklearn.metrics module are used to compute the accuracy, classification report, and confusion matrix of the model, respectively.

```
#SVM without dimention reduction and 10-fold cross-validation
# Initialize the SVM classifier with a linear kernel
svm_clf = svm.SVC(kernel='linear')
# Define the KFold cross-validation strategy
cv = KFold(n_splits=10, shuffle=True, random_state=42)

# Make predictions using 10-fold cross-validation on the training data
y_pred_cv  = cross_val_predict(svm_clf, X_train, y_train, cv=cv)

# Print the classification report
print("Classification report on training data with 10-fold cross-validation:")
print(classification_report(y_train, y_pred_cv))

# Print the confusion matrix
print("Confusion matrix on training data with 10-fold cross-validation:")
print(confusion_matrix(y_train, y_pred_cv))

# Fit the model to the training data
svm_clf.fit(X_train, y_train)

# Predict the classes of the test set
y_pred = svm_clf.predict(X_test)

# Compute the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy:', accuracy)

# Compute the classification report and confusion matrix
report = classification_report(y_test, y_pred)
print('Classification Report:\n', report)

matrix = confusion_matrix(y_test, y_pred)
print('Confusion Matrix:\n', matrix)
```

Fig11.

(Fig11) -> This code is performing Support Vector Machine (SVM) classification on a dataset with a linear kernel and 10-fold cross-validation.The first step is to initialize the SVM classifier with a linear kernel using the svm.SVC() function from the sklearn.svm module.Next, the KFold cross-validation strategy is defined using the KFold function from the sklearn.model_selection module. The n_splits parameter is set to 10 to perform 10-fold cross-validation, shuffle is set to True to shuffle the data before splitting, and random_state is set to 42 to ensure reproducibility.Then, the cross_val_predict() function from the sklearn.model_selection module is used to make predictions using 10-fold cross-validation on the training data. This function performs K-fold cross-validation and returns the predicted class labels for each sample in the training data. The classification_report() function from the sklearn.metrics module is used to print the classification report on the training data with 10-fold cross-validation, which includes precision, recall, F1-score, and support for each class. The confusion_matrix() function from the sklearn.metrics module is used to print the confusion matrix on the training data with 10-fold cross-validation, which shows the number of true positives, false positives, true negatives, and false negatives for each class. Next, the svm_clf.fit() function is used to fit the model to the training data.

Then, the svm_clf.predict() function is used to predict the classes of the test set. The accuracy_score() function from the sklearn.metrics module is used to compute the accuracy of the model. Finally, the classification_report() function and confusion_matrix() function are used again to compute the classification report and confusion matrix on the test data, respectively.

```
#SVM with hyper_parameters With reduction(ICA)
param_grid = {'C': [0.1, 1, 10, 100], 'gamma': [0.001, 0.01, 0.1, 1], 'kernel': ['linear', 'rbf']}

# Initialize the SVM classifier
svm_clf = svm.SVC()

# Define the grid search cross-validation strategy
cv = GridSearchCV(svm_clf, param_grid, cv=10)

# Fit the model to the training data using grid search cross-validation
cv.fit(X_train_ica, y_train)

# Print the best hyperparameters
print("Best hyperparameters:", cv.best_params_)

# Fit the model to the training data using the best hyperparameters
svm_clf_best = svm.SVC(C=cv.best_params_['C'], gamma=cv.best_params_['gamma'], kernel=cv.best_params_['kernel'])
svm_clf_best.fit(X_train_ica, y_train)

# Predict the classes of the test set
y_pred = svm_clf_best.predict(X_test_ica)

# Compute the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy:', accuracy)

# Compute the classification report and confusion matrix
report = classification_report(y_test, y_pred)
print('Classification Report:\n', report)

matrix = confusion_matrix(y_test, y_pred)
print('Confusion Matrix:\n', matrix)
```

Fig12.

(Fig12) -> This code trains a support vector machine (SVM) classification model with hyperparameter tuning using Independent Component Analysis (ICA) for feature reduction.The SVM classifier is initialized with default hyperparameters, and a grid of hyperparameters is defined using the param_grid dictionary. A GridSearchCV object is created with the SVM classifier and the param_grid dictionary, which performs a grid search cross-validation with 10 folds.The best hyperparameters found during the grid search are used to create a new SVM classifier, which is trained on the training data. The accuracy of the model is then computed using the test set, and a classification report and confusion matrix are generated to evaluate the performance of the model.

```python
#SVM with dimention reduction ICA and 10-fold cross-validation Without hyperparameters
# Initialize the SVM classifier with a linear kernel
svm_clf = svm.SVC(kernel='linear')

# Define the KFold cross-validation strategy
cv = KFold(n_splits=10, shuffle=True, random_state=42)

# Make predictions using 10-fold cross-validation on the training data
y_pred_cv  = cross_val_predict(svm_clf, X_train_ica, y_train, cv=cv)

# Print the classification report
print("Classification report on training data with 10-fold cross-validation:")
print(classification_report(y_train, y_pred_cv))

# Print the confusion matrix
print("Confusion matrix on training data with 10-fold cross-validation:")
print(confusion_matrix(y_train, y_pred_cv))

# Fit the model to the training data
svm_clf.fit(X_train_ica, y_train)

# Predict the classes of the test set
y_pred = svm_clf.predict(X_test_ica)

# Compute the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy:', accuracy)

# Compute the classification report and confusion matrix
report = classification_report(y_test, y_pred)
print('Classification Report:\n', report)
matrix = confusion_matrix(y_test, y_pred)
print('Confusion Matrix:\n', matrix)
```

Fig13.

(Fig13) -> This code implements a support vector machine (SVM) classification model with linear kernel and Independent Component Analysis (ICA) for dimension reduction. It performs 10-fold cross-validation on the training data and prints the classification report and confusion matrix. Then, it fits the model to the training data, predicts the classes of the test set, and computes the accuracy, classification report, and confusion matrix.

```
#SVM with hyper_parameters With reduction(LDA)
param_grid = {'C': [0.1, 1, 10, 100], 'gamma': [0.001, 0.01, 0.1, 1], 'kernel': ['linear', 'rbf']}

# Initialize the SVM classifier
svm_clf = svm.SVC()

# Define the grid search cross-validation strategy
cv = GridSearchCV(svm_clf, param_grid, cv=10)

# Fit the model to the training data using grid search cross-validation
cv.fit(X_train_lda, y_train)

# Print the best hyperparameters
print("Best hyperparameters:", cv.best_params_)

# Fit the model to the training data using the best hyperparameters
svm_clf_best = svm.SVC(C=cv.best_params_['C'], gamma=cv.best_params_['gamma'], kernel=cv.best_params_['kernel'])
svm_clf_best.fit(X_train_lda, y_train)

# Predict the classes of the test set
y_pred = svm_clf_best.predict(X_test_lda)

# Compute the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy:', accuracy)

# Compute the classification report and confusion matrix
report = classification_report(y_test, y_pred)
print('Classification Report:\n', report)

matrix = confusion_matrix(y_test, y_pred)
print('Confusion Matrix:\n', matrix)
```

Fig14.

(Fig14) -> This code uses a SVM with hyperparameter tuning and LDA feature reduction. It initializes the SVM with default hyperparameters, creates a GridSearchCV object to search for the best hyperparameters using cross-validation, fits the model using the training data, creates a new SVM classifier with the best hyperparameters found, predicts the classes of the test set, computes the accuracy of the model, generates a text report showing the main classification metrics, and generates a confusion matrix to evaluate the performance of the model.

```
#Random Forests without dimention reduction and 10-fold cross-validation

# Create the random forest classifier
rf = RandomForestClassifier(n_estimators=100, random_state=42)

# Use 10-fold cross-validation to train the model
kf = KFold(n_splits=10, shuffle=True, random_state=42)
y_pred = cross_val_predict(rf, X_train, y_train, cv=kf)

# Print the classification report
print("Classification report on training data with 10-fold cross-validation:")
print(classification_report(y_train, y_pred))

# Print the confusion matrix
print("Confusion matrix on training data with 10-fold cross-validation:")
print(confusion_matrix(y_train, y_pred))


# Evaluate the model on the testing set
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)

# Compute the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy:', accuracy)

# Compute the classification report and confusion matrix
report = classification_report(y_test, y_pred)
print('Classification Report:\n', report)

matrix = confusion_matrix(y_test, y_pred)
print('Confusion Matrix:\n', matrix)
```

Fig 15.

(Fig15) -> This code builds a Random Forest Classifier without dimensionality reduction using the scikit-learn library. It evaluates the model's performance with 10-fold cross-validation, and prints the accuracy, classification report, and confusion matrix on the testing set. The code does not perform feature selection or dimensionality reduction techniques.

```python
#Random Forests with LDA dimention reduction and 10-fold cross-validation

# Create the random forest classifier
rf = RandomForestClassifier(n_estimators=100, random_state=42)

# Use 10-fold cross-validation to train the model
kf = KFold(n_splits=10, shuffle=True, random_state=42)
y_pred = cross_val_predict(rf, X_train_lda, y_train, cv=kf)

# Print the classification report
print("Classification report on training data with 10-fold cross-validation:")
print(classification_report(y_train, y_pred))

# Print the confusion matrix
print("Confusion matrix on training data with 10-fold cross-validation:")
print(confusion_matrix(y_train, y_pred))


# Evaluate the model on the testing set
rf.fit(X_train_lda, y_train)
y_pred = rf.predict(X_test_lda)

# Compute the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy:', accuracy)

# Compute the classification report and confusion matrix
report = classification_report(y_test, y_pred)
print('Classification Report:\n', report)

matrix = confusion_matrix(y_test, y_pred)
print('Confusion Matrix:\n', matrix)
```

Fig 16.

(Fig 16) -> This Python code builds a Random Forest Classifier with LDA dimensionality reduction using scikit-learn. It creates the Random Forest Classifier with n_estimators=100 and random_state=42 and uses KFold cross-validation with n_splits=10, shuffle=True, and random_state=42 to evaluate the model's performance. The code performs LDA dimensionality reduction before training the model, which can improve performance. The accuracy, classification report, and confusion matrix are printed for both the training and testing sets.

```
# Create the random forest classifier
rf = RandomForestClassifier(n_estimators=100, random_state=42)

# Use 10-fold cross-validation to train the model
kf = KFold(n_splits=10, shuffle=True, random_state=42)
y_pred = cross_val_predict(rf, X_train_ica, y_train, cv=kf)

# Print the classification report
print("Classification report on training data with 10-fold cross-validation:")
print(classification_report(y_train, y_pred))

# Print the confusion matrix
print("Confusion matrix on training data with 10-fold cross-validation:")
print(confusion_matrix(y_train, y_pred))


# Evaluate the model on the testing set
rf.fit(X_train_ica, y_train)
y_pred = rf.predict(X_test_ica)

# Compute the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy:', accuracy)

# Compute the classification report and confusion matrix
report = classification_report(y_test, y_pred)
print('Classification Report:\n', report)

matrix = confusion_matrix(y_test, y_pred)
print('Confusion Matrix:\n', matrix)
```

Fig17.

(Fig17) -> This is a Python code for building a Random Forest Classifier with ICA dimensionality reduction technique using the scikit-learn library. The code uses the Random Forest Classifier with n_estimators=100 and random_state=42, and KFold cross-validation method with n_splits=10, shuffle=True, and random_state=42 to evaluate the model's performance.First, the Random Forest Classifier is created with n_estimators=100 and random_state=42. Next, the KFold cross-validation method is used to make predictions on the training set using the random forest classifier, ICA-transformed training set (X_train_ica), and the KFold cross-validation object. The classification_report and confusion_matrix functions are called to print the classification report and confusion matrix for the training set predictions.The random forest classifier is fitted to the ICA-transformed training set using the fit method. The predict method is used to make predictions on the ICA-transformed testing set (X_test_ica). The accuracy_score, classification_report, and confusion_matrix functions are called to compute and print the accuracy, classification report, and confusion matrix for the testing set predictions. Note that this code performs ICA dimensionality reduction technique before training the random forest classifier, which can improve the model's performance.

```
from sklearn.cluster import KMeans
!pip install scikit-fuzzy
import skfuzzy as fuzz
```

Fig18.

(Fig18)-> The code imports the necessary libraries: KMeans from sklearn.cluster, and skfuzzy as fuzz for clustering.

```
#K-means algorithm without dimention reduction

# Instantiate a KMeans object with the desired number of clusters
kmeans = KMeans(n_clusters=4)

# Fit the KMeans model to the data
kmeans.fit(clustring_df)

# Get the cluster labels for each data point
labels = kmeans.predict(clustring_df)

# Add the cluster labels to the original DataFrame
clustring_df['cluster'] = labels

# Print the counts of data points in each cluster
print(clustring_df['cluster'].value_counts())

# Print the centroids of each cluster
centroids = kmeans.cluster_centers_
print(centroids)
```

Fig19.

(Fig19)-> This code performs K-means clustering on a dataset called "processed.cleveland.data", using the scikit-learn library in Python. Here's a step-by-step breakdown of what the code does:

1. The code instantiates a KMeans object with n_clusters=4. This means that we want to identify four clusters in the data.
2. The KMeans model is then fit to the data using the fit() method. This means that the algorithm is run on the input data to identify the best cluster centers.
3. The code then uses the predict() method to assign each data point to a cluster based on the KMeans model.
4. The cluster assignments are added to the original DataFrame "clustering_df" as a new column called "cluster".
5. The code prints the number of data points in each cluster using the value_counts() method.
6. Finally, the code prints the centroids of each cluster using the cluster_centers_ attribute.

K-means clustering is a commonly used unsupervised machine learning algorithm that attempts to group similar data points together into clusters based on their features. It does this by iteratively assigning data points to the nearest cluster centroid (center of mass), and then re-computing the centroid of each

cluster based on the new data assignments. The process continues until convergence, at which point the centroids no longer change and the clusters are considered "stable". The KMeans algorithm is one of the most popular and efficient clustering algorithms, and is widely used in industry and research.

```
#K-means algorithm with ICA dimention reduction

clustring_df = df[input_parameters]

# Dimensionality reduction using ICA

ica = FastICA(n_components=10)
X_ica = ica.fit_transform(clustring_df)


# Instantiate a KMeans object with the desired number of clusters
kmeans = KMeans(n_clusters=4, random_state=42)

# Fit the KMeans model to the data
kmeans.fit(X_ica)

# Get the cluster labels for each data point
labels = kmeans.labels_

# Add the cluster labels to the original DataFrame
clustring_df['cluster_ICA'] = labels

# Print the counts of data points in each cluster
print(clustring_df['cluster_ICA'].value_counts())

# Print the centroids of each cluster
centroids = kmeans.cluster_centers_
print(centroids)
```

Fig19.

(Fig19)-> This code performs K-means clustering on a dataset after reducing its dimensionality using Independent Component Analysis (ICA). The dataset is first reduced to 10 independent components using the FastICA algorithm. Then, a KMeans object is created with 4 clusters, and the model is fit to the transformed data. The resulting cluster labels are added to the original dataset, and the counts of data points in each cluster are printed to the console. Finally, the centroids of each cluster are also printed to the console.

```
#Fuzzy Clustering algorithm without dimention reduction

clustring_df = df[input_parameters]

# Extract numeric data from dataframe and convert to numpy array
numeric_data = clustring_df.select_dtypes(include=[np.number]).values

# Normalize data
numeric_data_normalized = (numeric_data - np.mean(numeric_data, axis=0)) / np.std(numeric_data, axis=0)

# Set number of clusters
n_clusters = 5

# Apply fuzzy clustering
cntr, u, u0, d, jm, p, fpc = fuzz.cluster.cmeans(numeric_data_normalized.T, n_clusters, 2, error=0.005, maxiter=1000, seed=0)

# Extract cluster membership values from fuzzy clustering results
cluster_membership = np.argmax(u, axis=0)

# Add cluster membership values to original dataframe
clustring_df['cluster_fuzzy'] = cluster_membership

# Print the counts of data points in each cluster
print(clustring_df['cluster_fuzzy'].value_counts())

# Print the centroids of each cluster
centroids = cntr.T
print(centroids)
```

Fig20.

(Fig20)-> This code implements a fuzzy clustering algorithm using the fuzz.cluster.cmeans function from the fuzzywuzzy library. The algorithm works with a dataset contained in a Pandas dataframe df, and extracts the numeric data into a numpy array numeric_data. The data is then normalized using z-score normalization. The algorithm is set to identify n_clusters number of clusters.

The fuzz.cluster.cmeans function returns a number of values including the cluster centers (cntr), membership matrices (u), and the final fuzzy partition coefficient (fpc). Cluster membership values are extracted from the membership matrices using the np.argmax function. The resulting cluster membership values are added as a new column in the original dataframe. The code then prints the counts of data points in each cluster and the centroids of each cluster.

```
#Fuzzy Clustering algorithm with LDA dimention reduction

clustring_df = df[input_parameters]

# Dimensionality reduction using LDA
lda = LinearDiscriminantAnalysis(n_components = max_components)
X_lda = lda.fit_transform(clustring_df,df[output_parameter])

# Set number of clusters
n_clusters = 5

# Apply fuzzy clustering
cntr, u, u0, d, jm, p, fpc = fuzz.cluster.cmeans(X_lda.T, n_clusters, 2, error=0.005, maxiter=1000, seed=0)

# Extract cluster membership values from fuzzy clustering results
cluster_membership = np.argmax(u, axis=0)

# Add cluster membership values to original dataframe
clustring_df['cluster_fuzzy'] = cluster_membership

# Print the counts of data points in each cluster
print(clustring_df['cluster_fuzzy'].value_counts())

# Print the centroids of each cluster
centroids = cntr.T
print(centroids)
```

Fig21.

(Fig21)-> This code implements a fuzzy clustering algorithm with dimensionality reduction using Linear Discriminant Analysis (LDA). The code first extracts the required columns from the input dataframe into clustring_df. Then, LDA is applied to reduce the dimensionality of the data using LinearDiscriminantAnalysis from the sklearn library. The number of components in LDA is set to max_components.

Next, the number of clusters for fuzzy clustering is set to n_clusters. The fuzzy clustering algorithm is then applied to the transformed data X_lda, which was obtained after the LDA dimensionality reduction.

The fuzz.cluster.cmeans function returns several values, including the cluster centers (cntr), membership matrices (u), and the final fuzzy partition coefficient (fpc). The code then extracts cluster membership values from the membership matrices using np.argmax. The resulting cluster membership values are added as a new column in the original dataframe. The code then prints the counts of data points in each cluster and the centroids of each cluster.

Results

Method	accuracy
SVM with hyper parameters without reduction and 10-fold cross-validation `Best hyperparameters: {'C': 100, 'gamma': 0.001, 'kernel': 'rbf'}`	81%
SVM with hyper parameters with LDA reduction and 10-fold cross-validation	62%
Random Forests with hyper parameters without reduction and 10-fold cross-validation `Best Hyperparameters: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100}`	87%
Random Forests with hyper parameters with ICA dimention reduction and 10-fold cross-validation	84%

Classification

K-means algorithm without dimension reduction	Cluster 3 has the largest number of data points (118). Cluster 1 has 90 data points. Cluster 0 has 84 data points. Cluster 2 has only 5 data points.
K-means algorithm with ICA dimension reduction	Cluster 0 has the largest number of data points (106). Cluster 1 has 76 data points. Cluster 3 has 58 data points. Cluster 2 has 57 data points.
Fuzzy Clustering algorithm without dimension reduction	• Cluster 2 has the largest number of data points (159). • Cluster 1 has 133 data points.

	• Cluster 4 has only 4 data points. • Cluster 0 has only 1 data point.
Fuzzy Clustering algorithm with LDA dimension reduction	• Cluster 2 has the largest number of data points (69). • Cluster 0 has 67 data points. • Cluster 3 has 60 data points. • Cluster 4 has 54 data points. • Cluster 1 has 47 data points

Clustering

Contributors in this project

All parts of this project have been done equally by presenters and each one has a 50 percent of contribution to research, study, applying techniques in Python, and preparing explanation and presentation files.

Conclusion

Based on the results, we can see that the Random Forest algorithm is a strong performer in this dataset. It achieved the highest accuracy score of 87% without any dimensionality reduction, which suggests that the model is able to capture the underlying patterns in the data without the need for dimensionality reduction techniques.

In contrast, the SVM algorithm without dimensionality reduction achieved an accuracy score of 81%, which is still a respectable score. However, when LDA dimensionality reduction was applied to the SVM model, the accuracy score dropped significantly to 62%. This suggests that the LDA dimensionality reduction technique may not have been effective in capturing the relevant features in the data for this specific problem.

When dimensionality reduction was applied to the Random Forest algorithm using ICA, the accuracy score remained quite high at 84%, which indicates that the ICA technique was able to capture the relevant features in the data to a certain extent.

Overall, it seems that the Random Forest algorithm may be a better choice for this dataset than the SVM algorithm, as it achieved higher accuracy scores without the need for dimensionality reduction techniques. However, it is always a good idea to try out multiple algorithms and techniques and compare their performance before settling on a final model.

We can see that the K-means algorithm without dimension reduction and the fuzzy clustering algorithm without dimension reduction both identified a cluster with a very small number of data points (cluster 2 in K-means and cluster 4 in fuzzy clustering). This suggests that these algorithms might not be the best choice for this particular dataset.

In contrast, the K-means algorithm with ICA dimension reduction and the fuzzy clustering algorithm with LDA dimension reduction did not have such a problem and identified clusters that are more evenly distributed. However, there are differences in the specific clusters that each algorithm identified as the largest (cluster 3 in K-means with ICA and cluster 2 in fuzzy clustering with LDA).

Overall, the choice of algorithm and dimension reduction technique depends on the specific dataset and the goals of the analysis. It's important to carefully consider the results and choose the approach that best suits the particular task at hand.

References

[1] Levenson, J. W., Skerrett, P. J., & Gaziano, J. M. (2002). Reducing the global burden of cardiovascular disease: the role of risk factors. *Preventive cardiology*, *5*(4), 188-199.

[2] https://archive.ics.uci.edu/ml/machine-learning-databases/heart-disease/

[3] Lu, W., & Rajapakse, J. C. (2005). Approach and applications of constrained ICA. *IEEE transactions on neural networks*, *16*(1), 203-212.

[4] Samaitis, V., Yilmaz, B., & Jasiuniene, E. (2023). Adhesive bond quality classification using machine learning algorithms based on ultrasonic pulse-echo immersion data. *Journal of Sound and Vibration*, *546*, 117457.

[5] Kaur, S., & Jindal, S. (2016). A survey on machine learning algorithms. *Int J Innovative Res Adv Eng (IJIRAE)*, *3*(11), 2349-2763.

[6] Aljarah, I., Al-Zoubi, A. M., Faris, H., Hassonah, M. A., Mirjalili, S., & Saadeh, H. (2018). Simultaneous feature selection and support vector machine optimization using the grasshopper optimization algorithm. *Cognitive Computation*, *10*, 478-495.

[7] Naji, M. A., El Filali, S., Aarika, K., Benlahmar, E. H., Abdelouhahid, R. A., & Debauche, O. (2021). Machine learning algorithms for breast cancer prediction and diagnosis. *Procedia Computer Science*, *191*, 487-492.

[8] Sujatha, S., & Sona, A. S. (2013). New fast k-means clustering algorithm using modified centroid selection method. *International Journal of Engineering Research & Technology (IJERT)*, *2*(2), 1-9.

[9] Bora, D. J., Gupta, D., & Kumar, A. (2014). A comparative study between fuzzy clustering algorithm and hard clustering algorithm. *arXiv preprint arXiv:1404.6059*.