Student Name: Sawera Fazal Roll No:21A-026-se Section: 21A

## CS334 - Machine Learning

Lab 01

Instructor: Ms. Maham Ashraf E-mail: [mashraf@uit.edu](mailto:mashraf@uit.edu)

Semester: Fall, 2023

**Objective**

The purpose of this lab session is to introduce data preparation techniques for Machine Learning(ML) projects.

#### Instructions

You have to perform the following tasks yourselves. Raise your hand if you face any difficulty in understanding and solving these tasks. **Plagiarism** is an abhorrent practice and you should not engage in it.

#### How to Submit

* Submit lab work in a single .py file on Microsoft Teams. (No other format will be accepted)
* Lab work file name should be saved with your roll number (e.g. 19a-001-SE\_LW01.py)
* Submit home work in a single .py file on Microsoft Teams. (No other format will be accepted)
* Home work file name should be saved with your roll number (e.g. 19a-001-SE\_HW01.py)

#### Task 1 - Load Data From CSV

In this tutorial you will discover how to load your data in Python from scratch, including:

* 1. How to load a CSV file.
  2. How to convert strings from a file to floating point numbers.
  3. How to convert class values from a file to integers.

##### 1.1 Load CSV file

The standard file format for small datasets is Comma Separated Values or CSV. In its simplest form, CSV files are comprised of rows of data. Each row is divided into columns using a comma (,).

**Dataset 1:** The *Pima Indians Diabetes* Dataset involves the prediction of the onset of diabetes within 5 years. Download the dataset and save it into your current working directory with the filename pima- indians-diabetes.csv [().](https://www.kaggle.com/kumargh/pimaindiansdiabetescsv)

**Dataset 2:** Iris Flower Species Dataset involves the prediction of iris flower species. Download the dataset and save it into your current working directory with the filename iris.csv ([https:](https://www.kaggle.com/uciml/iris)

[//www.kaggle.com/uciml/iris](https://www.kaggle.com/uciml/iris)).

A limitation of this function is that it will load empty lines from data files and add them to our list of rows. We can overcome this by adding rows of data one at a time to our dataset and skipping empty rows. Below is the updated example with this new improved version of the load csv() function.

Listing 1: Example of Loading the Pima Indians Diabetes Dataset CSV File.

# Example of loading Pima Indians CSV dataset from csv import reader

# Load a CSV file

def load\_csv ( filename ): file = open ( filename , " r") lines = reader ( file ) dataset = list ( lines ) return dataset

# Load dataset

filename = ’ pima - indians - diabetes . csv ’ dataset = load\_csv ( filename )

print ( ’ Loaded data file { 0 } with { 1 } rows and { 2 } columns ’. format ( filename , len ( dataset ) , len ( dataset [ 0 ]) ))

# Example of loading Pima Indians CSV dataset from csv import reader

# Load a CSV file

def load\_csv ( filename ): dataset = list ()

with open ( filename , ’ r ’) as file : csv\_reader = reader ( file )

for row in csv\_reader : if not row :

continue

dataset . append ( row ) return dataset

# Load dataset

filename = ’ pima - indians - diabetes . csv ’ dataset = load\_csv ( filename )

print ( ’ Loaded data file { 0 } with { 1 } rows and { 2 } columns ’. format ( filename , len ( dataset ) , len ( dataset [ 0]) ))

Listing 2: Improved Example of Loading the Pima Indians Diabetes Dataset CSV File.

##### Task 1: Load iris.csv file

Download *iris.csv* file and load it into your system. Ignore empty lines if the are present in the dataset.

#### Convert String to Floats

Most, if not all machine learning algorithms prefer to work with numbers. Specifically, floating point numbers are preferred. Our code for loading a CSV file returns a dataset as a list of lists, but each value is a string.

We can write a small function to convert specific columns of our loaded dataset to floating point values. Below is this function called str column to float(). It will convert a given column in the dataset to oating point values, careful to strip any whitespace from the value before making the conversion.

def str\_column\_to\_float ( dataset , column ): for row in dataset :

row [ column ] = float ( row [ column ]. strip () )

Listing 3: Function For Converting String Data To Floats.

##### Task 2: Test your data

After converting string data into float values, test and verify your data for its successful conversion.

Table 1: Iris dataset

5.1,3.5,1.4,0.2,Iris-setosa

4.9,3.0,1.4,0.2,Iris-setosa

4.7,3.2,1.3,0.2,Iris-setosa

4.6,3.1,1.5,0.2,Iris-setosa

5.0,3.6,1.4,0.2,Iris-setosa

#### Convert String to Integers

The iris owers dataset is like the Pima Indians dataset, in that the columns contain numeric data. The difference is the final column, traditionally used to hold the outcome or value to be predicted for a given row. The final column in the iris flowers data is the iris flower species as a string. For example, in Table 1, the first 5 rows of the iris dataset contains string values. Some machine learning algorithms prefer all values to be numeric, including the outcome or predicted value. We can convert the class value in the iris flowers dataset to an integer by creating a map.

* 1. First, we locate all of the unique class values, which happen to be: Iris-setosa, Iris-versicolor and Iris-virginica.
  2. Next, we assign an integer value to each, such as: 0, 1 and 2.
  3. Finally, we replace all occurrences of class string values with their corresponding integer values.

Below is a function to do just that called str column to int(). Like the previously introduced str column to float() it operates on a single column in the dataset.

# Convert string column to integer

def str\_column\_to\_int ( dataset , column ): class\_values = [ row [ column ] for row in dataset ] unique = set ( class\_values )

lookup = dict ()

for i , value in enumerate ( unique ): lookup [ value ] = i

for row in dataset :

row [ column ] = lookup [ row [ column ]] return lookup

Listing 4: Function To Integer Encode String Class Values.

##### Task 3: Test your function

Test this new function in addition to the previous two functions for loading a CSV file and converting columns to floating point values. It should also return the dictionary mapping of class values to integer values, in case any users downstream want to convert predictions back to string values again. Running this example produces the output below. We can see the first row of the dataset before and after the data type conversions. We can also see the dictionary mapping of class values to integers.

Loaded data file iris.csv with 150 rows and 5 columns [’5.1’, ’3.5’, ’1.4’, ’0.2’, ’Iris-setosa’]

[5.1, 3.5, 1.4, 0.2, 1]

{’Iris-virginica’: 0, ’Iris-setosa’: 1, ’Iris-versicolor’: 2}

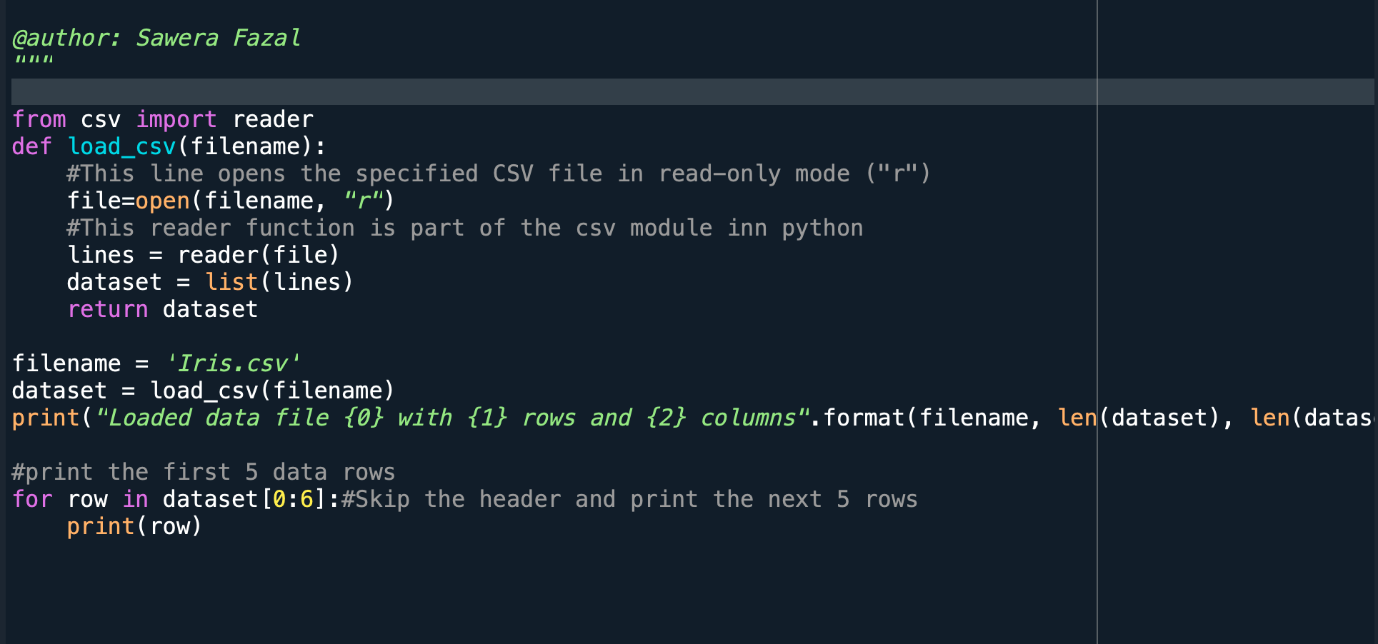
#### Exercises (Homework)

You learned how to load CSV files and perform basic data conversions. Data loading can be a difficult task given the variety of data cleaning and conversion that may be required from problem to problem. There are many extensions that you could make to make these examples more robust to new and different data files. Below are just a few ideas That you to implement yourself and submit as homework file on MS Teams:

* Detect and remove empty lines at the top or bottom of the file.
* Detect and handle missing values in a column.
* Detect and handle rows that do not match expectations for the rest of the file.
* Support for other delimiters such as pipe (|) or white space.
* Support more efficient data structures such as arrays.

Two libraries you may wish to use in practice for loading CSV data are NumPy and Pandas. NumPy offers the loadtxt()[1](#_bookmark0) function for loading data files as NumPy arrays. Pandas offers the read csv()[2](#_bookmark0) function that offers a lot of flexibility regarding data types, file headers and more.

Task 1

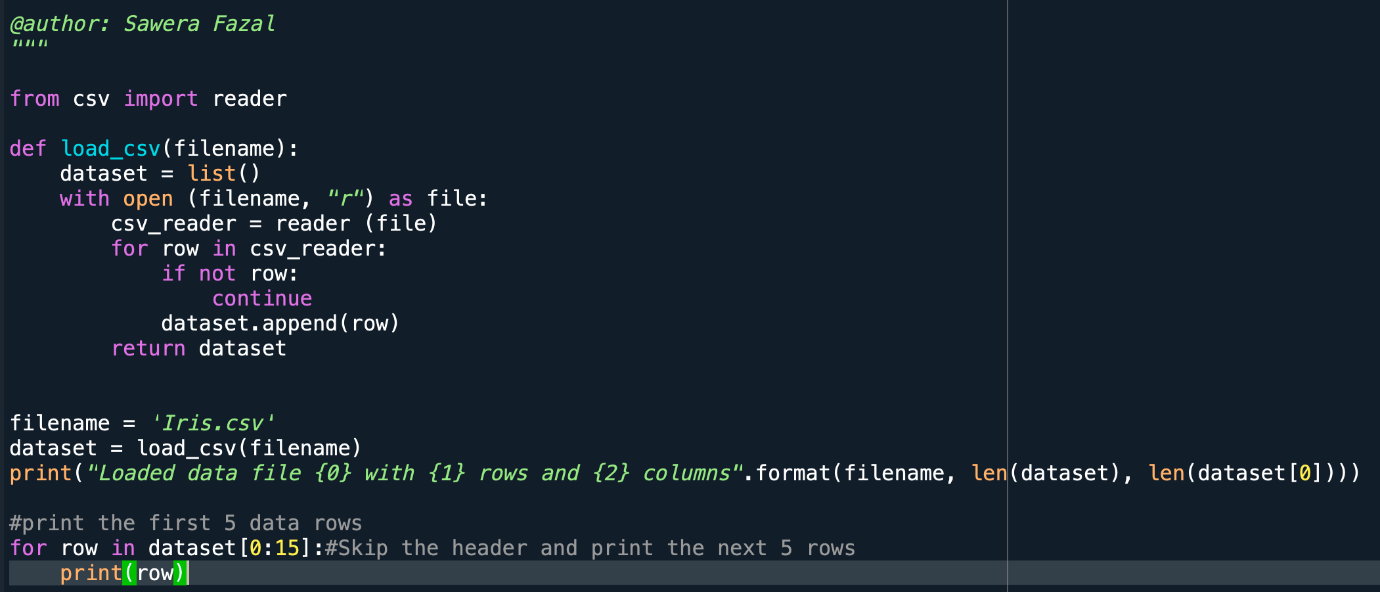


OUTPUT

A screen shot of a computer program

Description automatically generated

Task 2

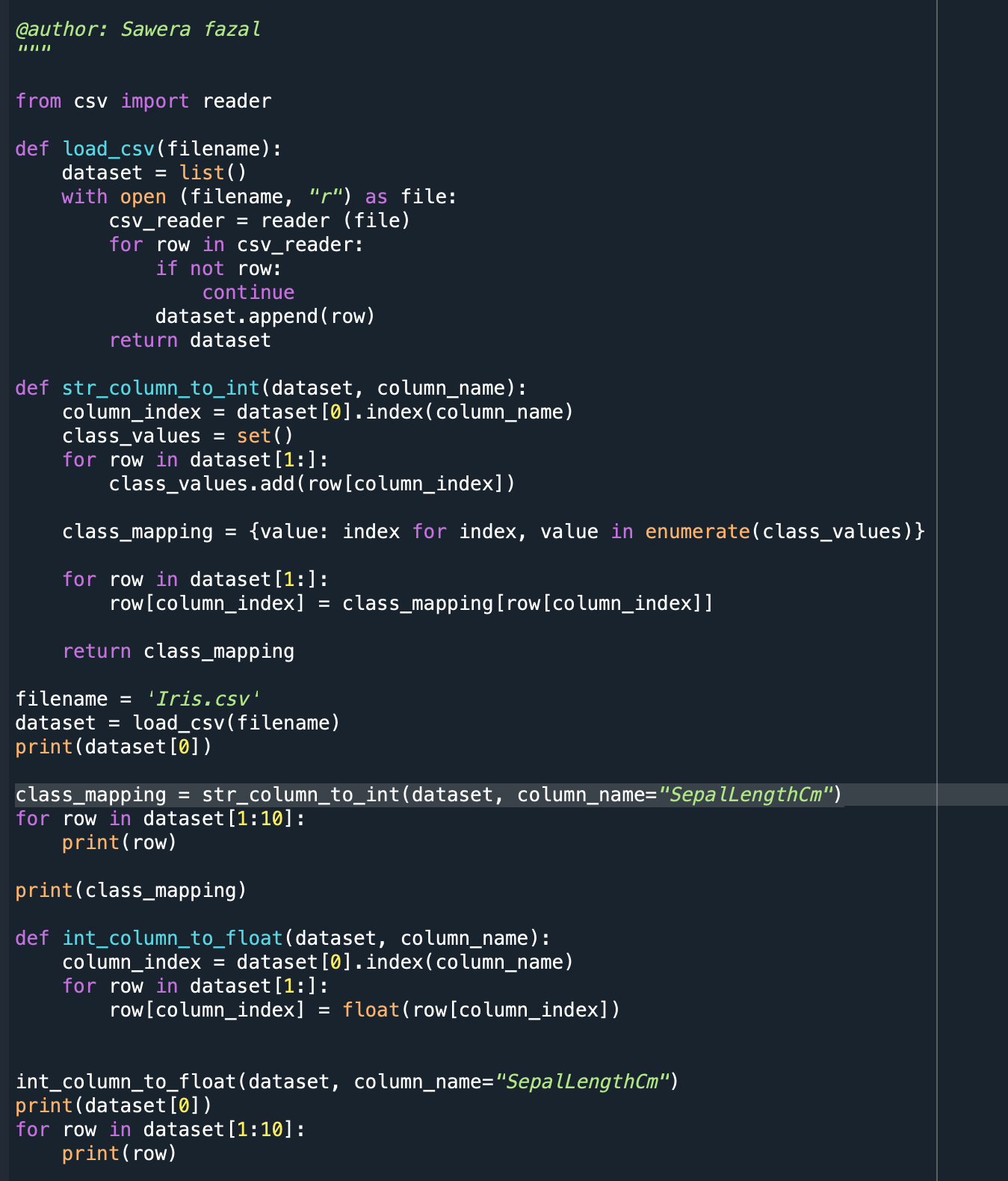


OUTPUT

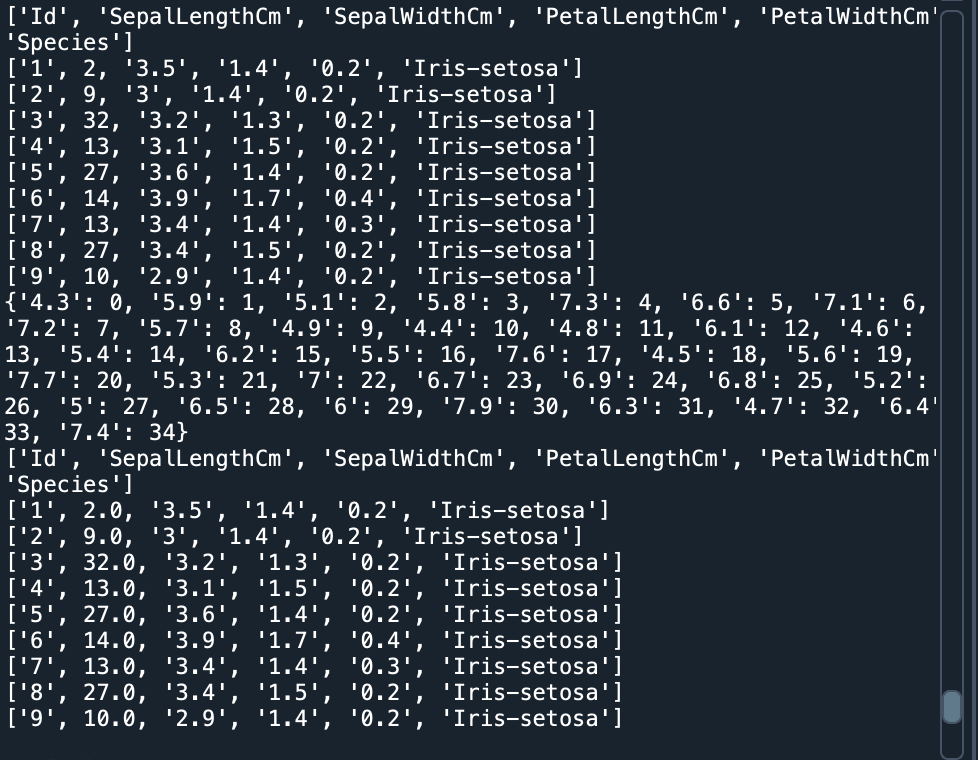
A computer screen shot of white text

Description automatically generated

Task 3



OUTPUT



LAB TASK

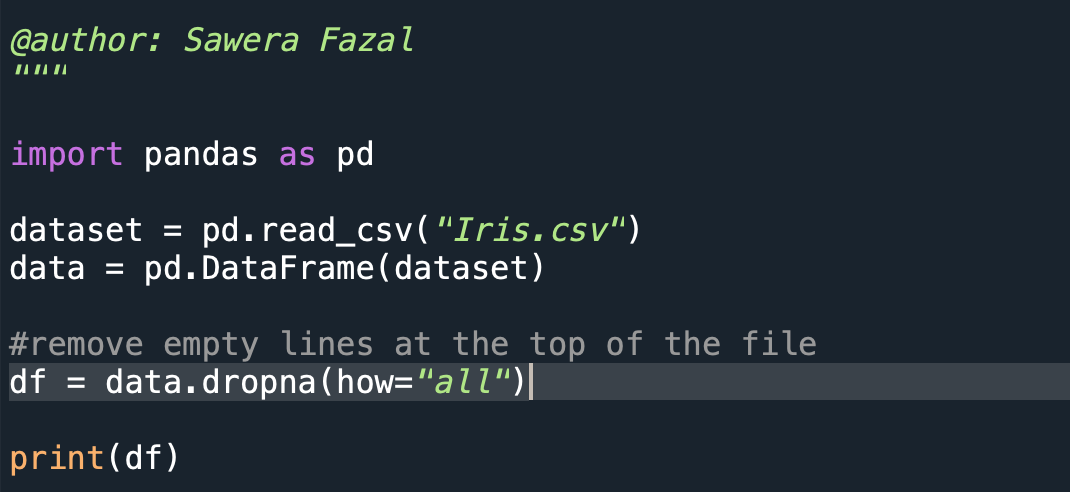
A screen shot of a computer program

Description automatically generated

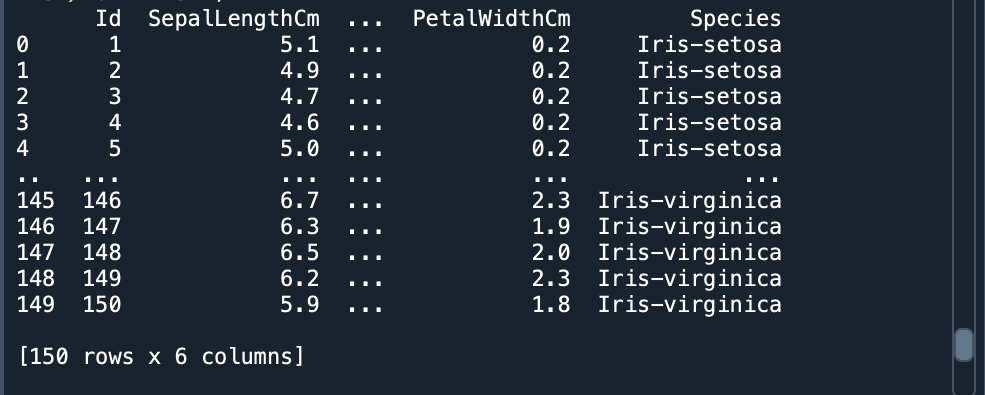
OUTPUT



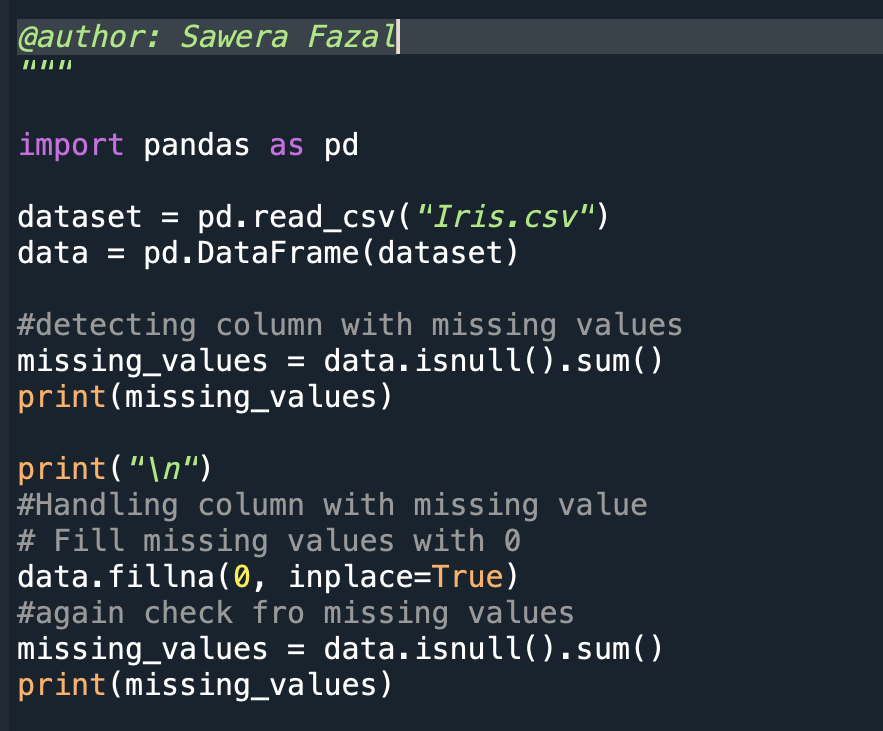
HOME TASK 1



OUTPUT



HOME TASK 2



OUTPUT

A computer screen shot of a computer

Description automatically generated

HOME TASK 3

A screenshot of a computer program

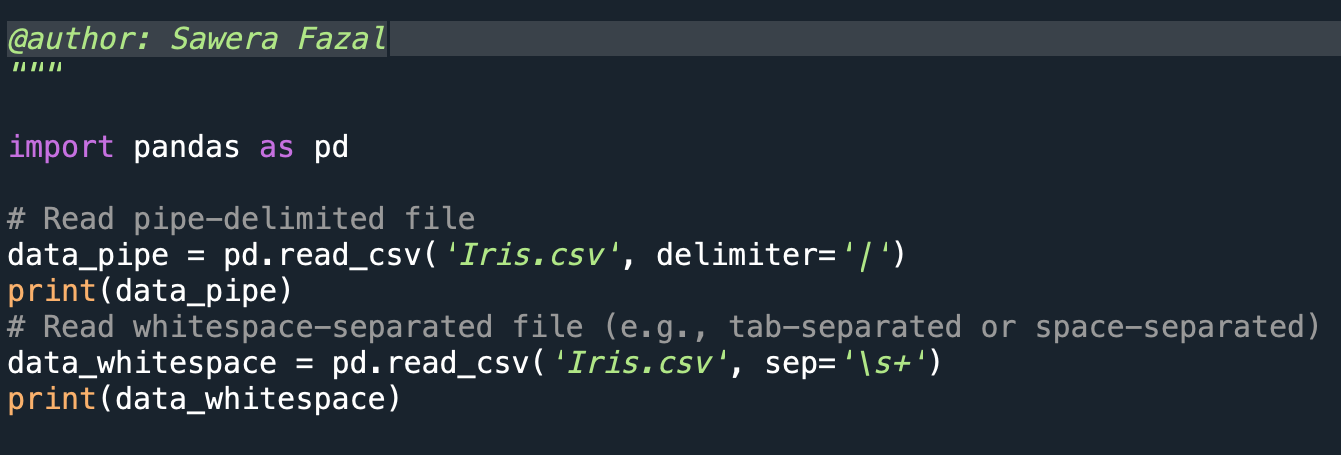
Description automatically generated

OUTPUT

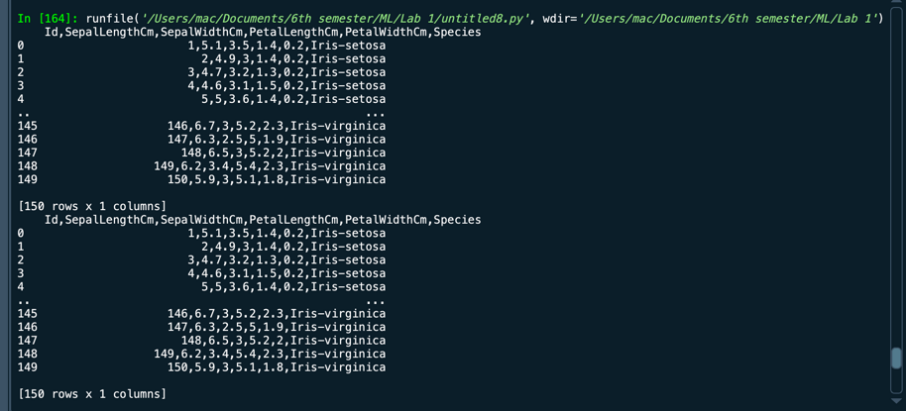
A screenshot of a computer

Description automatically generated

HOME TASK 4



OUTPUT

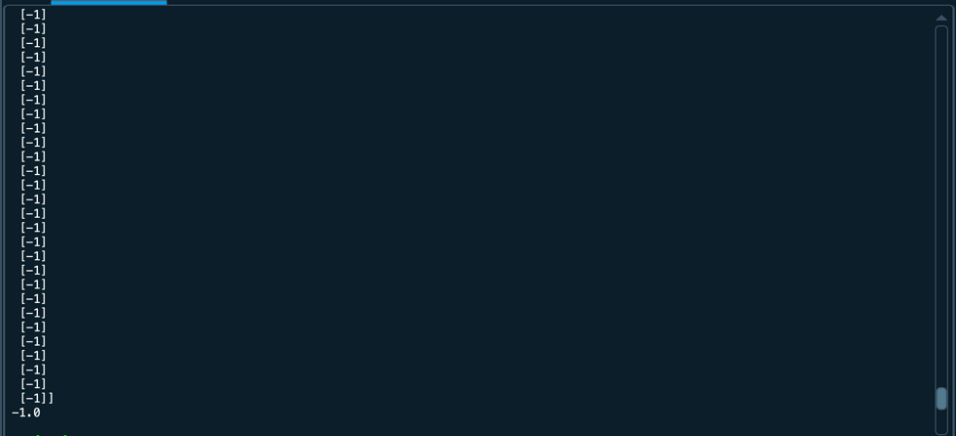


HOME TASK 5

A screenshot of a computer program

Description automatically generated

OUTPUT



Student Name: Sawera Fazal Roll No:21A-026-se Section: 21A

## CS334 - Machine Learning

Lab 02

Instructor: Dr. Maham Ashraf E-mail: [mashraf@uit.edu](mailto:mashraf@uit.edu)

Semester: Fall, 2023

**Objective**

The purpose of this lab session is to introduce data preparation techniques for Machine Learning(ML) projects.

#### Instructions

You have to perform the following tasks yourselves. Raise your hand if you face any difficulty in understanding and solving these tasks. **Plagiarism** is an abhorrent practice and you should not engage in it.

#### How to Submit

* Submit lab work in a single .py file on Microsoft Teams. (No other format will be accepted)
* Lab work file name should be saved with your roll number (e.g. 19a-001-SE\_LW01.py)
* Submit home work in a single .py file on Microsoft Teams. (No other format will be accepted)
* Home work file name should be saved with your roll number (e.g. 19a-001-SE\_HW01.py)

#### Summarize and Visualize your data

##### Data summary

Descriptive statistics is packed with information and insights. A pair of experienced eyes will take a look at every single data point and extract valuable information from the summary table. Let’s first see what a table of summary statistics looks like for a given dataset. We will use a built-in dataset that comes with *seaborn* library in Python.

import seaborn as sns import pandas as pd

df = sns . load\_dataset ( ’ tips ’) df . head ()

Listing 1: Load data in data frame.

Each observation (row) in this dataset represents dining in a restaurant. The columns names here are self-explanatory. Among the numeric columns, ’total\_bill’ refers to how much bills the diners paid and ’tip’ represents the amount of tip they paid.

Just a simple method call df.describe() gives you the summary statistics for the numeric columns (I’ll touch upon categorical columns towards the end).

#### Your Tasks

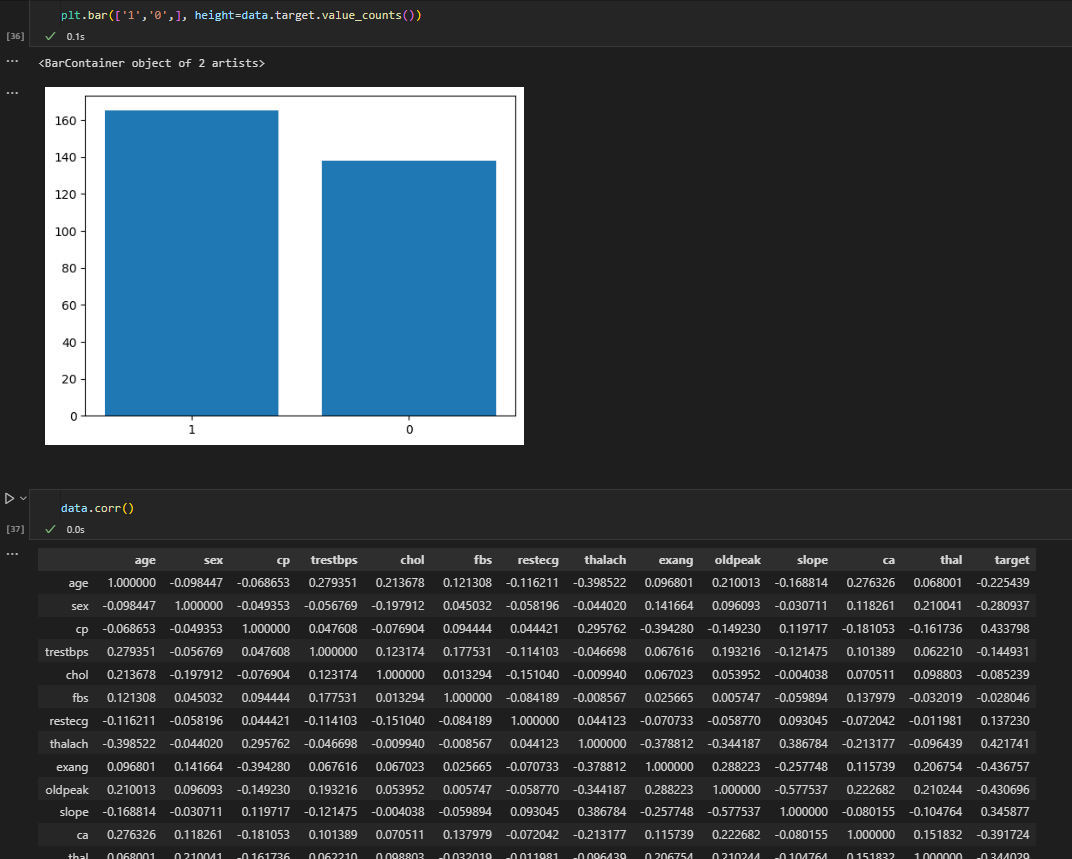
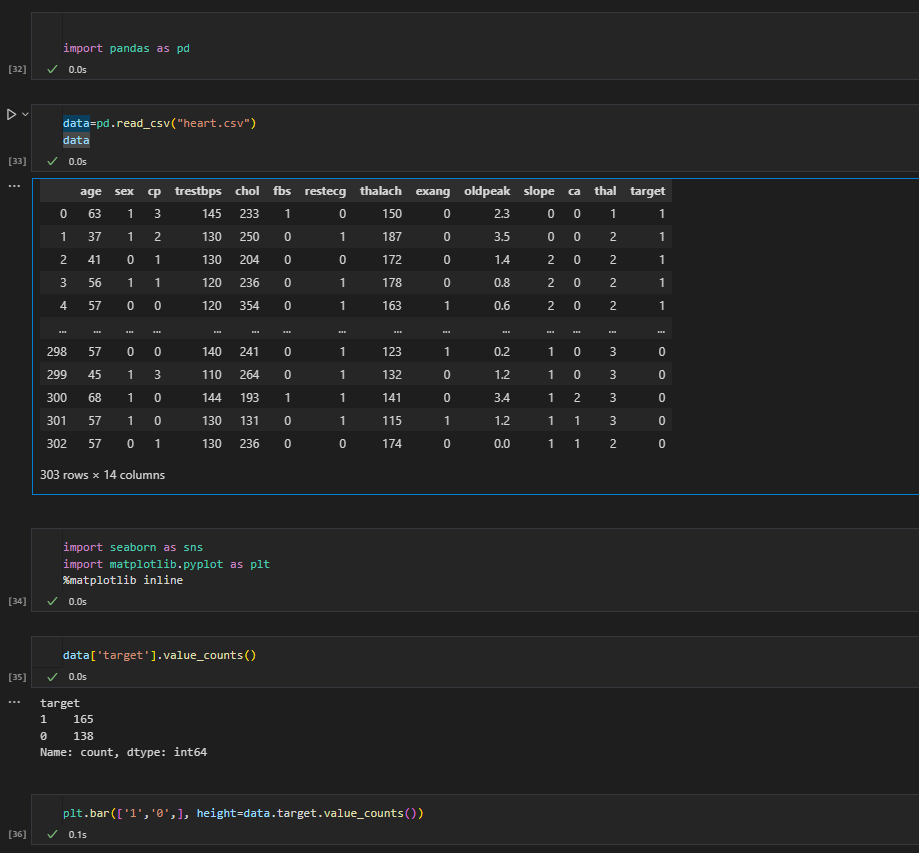
Task 1 Run the example as given in Listing - 1. Print top 10 rows of the data and also print the data summary as given in Figure - 2.

Task 2 Repeat Task 1 on *pima-indians-diabetes.csv* dataset.

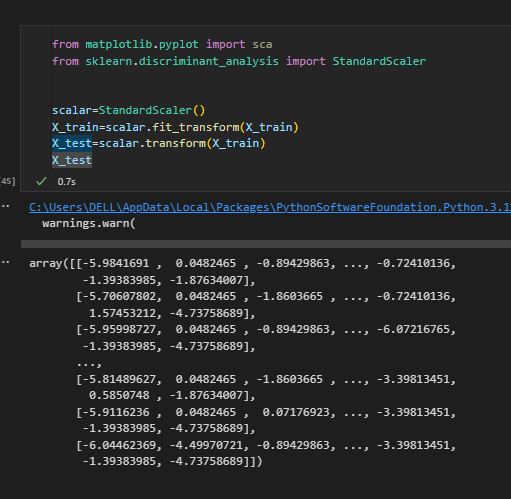
Task 3 Generate dot plot and distribution plot on *heart.csv* dataset (h[ttps://www.kaggle.com/fedesoriano/heart-](http://www.kaggle.com/fedesoriano/heart-) failure-prediction?select=heart.csv).

Task 4 Run the example as given in Listing - 3 and review the number of missing values in the dataset before and after the data imputation transform.

Task 5 Repeat Task 4 on *heart.csv* dataset.

A screenshot of a computer screen

Description automatically generatedA screenshot of a computer

Description automatically generated

Student Name: Sawera Fazal Roll No:21A-026-se Section: 21A

## CS334 - Machine Learning

Lab 03

Instructor: Ms. Maham Ashraf E-mail: [mashraf@uit.edu](mailto:mashraf@uit.edu)

Semester: Fall, 2023

**Objective**

The purpose of this lab session is to introduce Analytical Base Table (ABT) for Machine Learning(ML) projects.

#### Instructions

You have to perform the following tasks yourselves. Raise your hand if you face any difficulty in understanding and solving these tasks. **Plagiarism** is an abhorrent practice and you should not engage in it.

#### How to Submit

* Submit lab work in a single .py file on Microsoft Teams. (No other format will be accepted)
* Lab work file name should be saved with your roll number (e.g. 19a-001-SE\_LW01.py)
* Submit home work in a single .py file on Microsoft Teams. (No other format will be accepted)
* Home work file name should be saved with your roll number (e.g. 19a-001-SE\_HW01.py)

#### Data Processing

##### Analytics Base Table (ABT)

An **Analytics Base Table (ABT)** is a simple, flat, tabular data structure made up of rows and columns. The columns are divided into a set of descriptive features and a single target feature. Each row contains a value for each descriptive feature and the target feature and represents an instance about which a prediction can be made.

Pandas-DataFrame is a Python library specially designed to load and manipulate data in ABT.

import pandas as pd

data = pd . read\_csv (" Motor Insura nce Fraud Claim ABTFull . csv ") df = pd . Data Frame ( data )

df . head ()

Listing 1: Load data in data frame.

The above code load data from a csv file and place in a pandas data frame object. Now data is in ready for further processing.

#### Lab Tasks

1. Download [Zameen.com property](https://www.kaggle.com/datasets/huzzefakhan/zameencom-property-data-pakistan) data from [https://www.kaggle.com/datasets/huzzefakhan/](https://www.kaggle.com/datasets/huzzefakhan/zameencom-property-data-pakistan) [zameencom-property-data-pakistan](https://www.kaggle.com/datasets/huzzefakhan/zameencom-property-data-pakistan)
2. Describe the data properties of each column,
   * Data type of each column
   * missing values in each column
   * null values in each column
   * outliers in each columns
3. Handle null values by replacing with suitable values
4. Suppose you have to predict the cost of a house, for this purpose select the appropriate coulmns that will help you to develop machine laerning model. Save the selected coulmns dataset in a separate csv file.
5. List down the descriptive variables and target variable. 6 Describe the statistics of the new data.

7 Compute the covariance and correlation matrix among descriptive variables. 8 Group the data by city, location, and area.

1. Count the total values of each item of all attributes.
2. Encode categorical values of *’property\_type’* and *’province\_name’* fearures with numbers.
3. A screenshot of a computer program

   Description automatically generatedA screenshot of a computer program

   Description automatically generatedA screenshot of a computer program

   Description automatically generated

Student Name: Sawera Fazal Roll No:21A-026-se Section: 21A

## CS334 - Machine Learning

Lab 04 - Feature Selection Methods in ML (Part - 1) Instructor: Ms. Maha m Ashraf

E-mail: [mashraf@uit.edu](mailto:mashraf@uit.edu) Semester: Fall, 2023

**Objective**

The purpose of this lab session is to introduce feature selection methods for machine learning model. This lab is divided into two parts, in Part-1 we will use only Filter for feature selection. In Part-2, we will use Wrapper, Embedded and Hybrid methods for feature selections.

#### Instructions

You have to perform the following tasks yourselves. Raise your hand if you face any difficulty in understanding and solving these tasks. **Plagiarism** is an abhorrent practice and you should not engage in it.

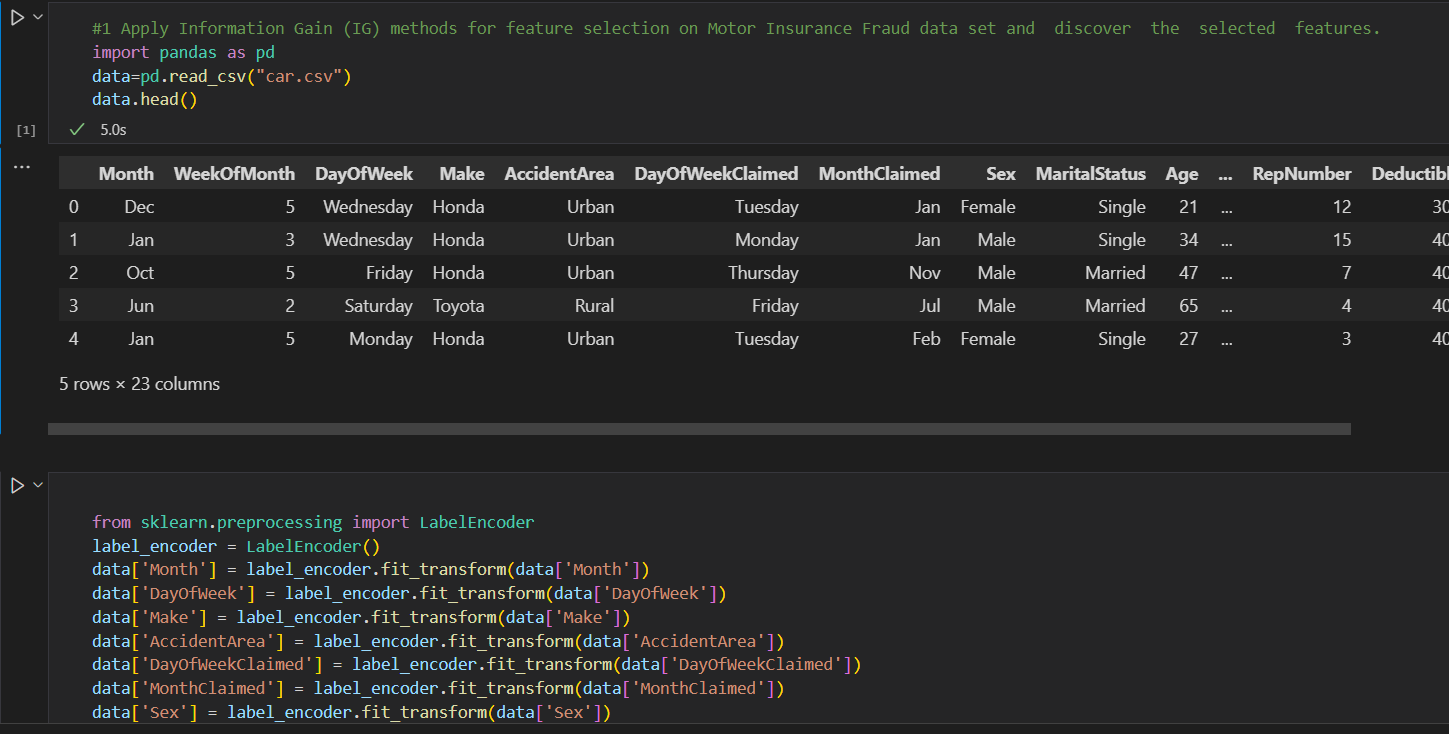
#### How to Submit

* Submit lab work in a single .py file on Microsoft Teams. (No other format will be accepted)
* Lab work file name should be saved with your roll number (e.g. 19a-001-SE\_LW01.py)
* Submit home work in a single .py file on Microsoft Teams. (No other format will be accepted)
* Home work file name should be saved with your roll number (e.g. 19a-001-SE\_HW01.py)

#### 

#### Homework

1. Apply *Information Gain (IG)* methods for feature selection on *Motor Insurance Fraud* data set and discover the selected features.

A screen shot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated

1. Apply *χ*2 methods for feature selection on *diabetes* data set and discover the selected features.

#2 Apply χ2 methods for feature selection on diabetes data set and discover the selected features.

import pandas as pd

df = pd.read\_csv('diabetes.csv')

# Select specific columns for analysis

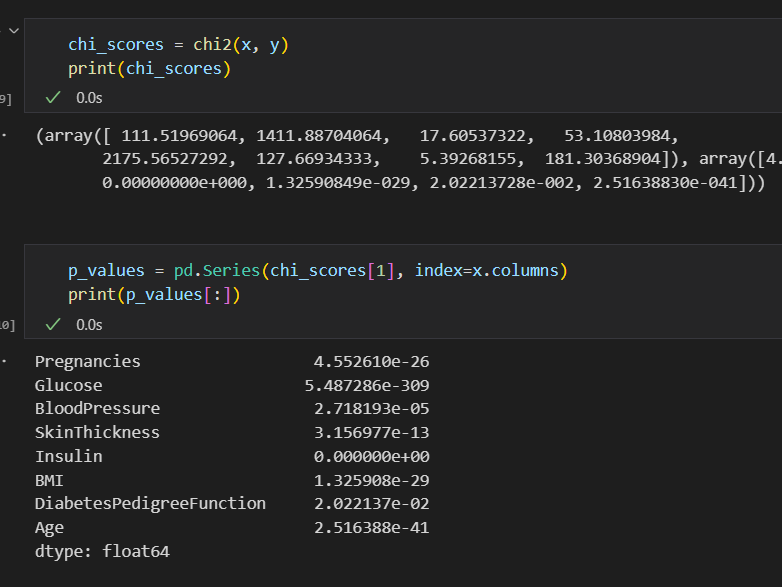
#The χ² test is particularly suitable for analyzing the association between categorical variables.

df = df.loc[:, ['Pregnancies','Glucose','BloodPressure','SkinThickness','Insulin','BMI','DiabetesPedigreeFunction','Age','Outcome']]

df

A screenshot of a computer

Description automatically generatedA screenshot of a computer screen

Description automatically generatedA screenshot of a computer program

Description automatically generated

3Apply *Fisher score* methods for feature selection on *Motor Insurance Fraud* data set and discover the selected features.

#3 Apply Fisher score methods for feature selection on Motor Insurance Fraud data set and discover the  selected  features

import pandas as pd

import numpy as np

from regex import B

from sklearn.feature\_selection import f\_classif

# Load the data

data = pd.read\_csv('car.csv')

# Select the desired features

selected\_features = ['Month',   'WeekOfMonth',  'DayOfWeek' ,'Make' ,'AccidentArea',    'DayOfWeekClaimed', 'MonthClaimed',

'Sex', 'MaritalStatus', 'Age','RepNumber'   ,'Deductible'   ,'DriverRating','PastNumberOfClaims','PoliceReportFiled'

,'WitnessPresent'   ,'AgentType',   'Year','BasePolicy',    'FraudFound']

# Data preprocessing if necessary

# Encoding categorical variables if needed

label\_encoder = LabelEncoder()

data['Month'] = label\_encoder.fit\_transform(data['Month'])

data['DayOfWeek'] = label\_encoder.fit\_transform(data['DayOfWeek'])

data['Make'] = label\_encoder.fit\_transform(data['Make'])

data['AccidentArea'] = label\_encoder.fit\_transform(data['AccidentArea'])

data['DayOfWeekClaimed'] = label\_encoder.fit\_transform(data['DayOfWeekClaimed'])

data['MonthClaimed'] = label\_encoder.fit\_transform(data['MonthClaimed'])

data['Sex'] = label\_encoder.fit\_transform(data['Sex'])

data['MaritalStatus'] = label\_encoder.fit\_transform(data['MaritalStatus'])

data['Fault'] = label\_encoder.fit\_transform(data['Fault'])

data['VehicleCategory'] = label\_encoder.fit\_transform(data['VehicleCategory'])

data['PoliceReportFiled'] = label\_encoder.fit\_transform(data['PoliceReportFiled'])

data['WitnessPresent'] = label\_encoder.fit\_transform(data['WitnessPresent'])

data['AgentType'] = label\_encoder.fit\_transform(data['AgentType'])

data['BasePolicy'] = label\_encoder.fit\_transform(data['BasePolicy'])

data['FraudFound'] = label\_encoder.fit\_transform(data['FraudFound'])

a = data.drop('FraudFound', axis=1)

b = data['FraudFound']

# Fill missing values if any

data = data.fillna(0) # You may want to use a more suitable method for your data

# Extract the features and the target variable

a = data[selected\_features[:-1]]  # Features

b = data[selected\_features[-1]]   # Target

# Calculate Fisher Score

fisher\_score, p\_value = f\_classif(X, Y)

# Print the results

for i in range(len(selected\_features) - 1):

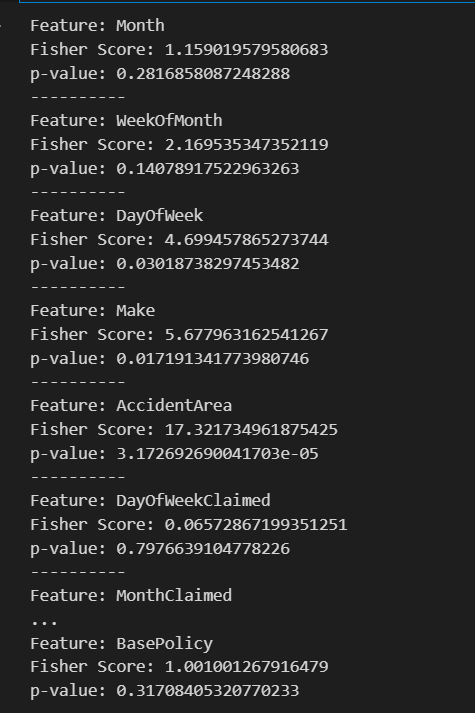
    print("Feature:", selected\_features[i])

    print("Fisher Score:", fisher\_score[i])

    print("p-value:", p\_value[i])

    print("----------")

#p values small ==most imp



4Apply *Correlation & Variance* methods for feature selection on *Property* data set and discover the selected features.

#4 Apply Correlation & Variance methods for feature selection on Property data set and discover the  selected  features

# correlation should be more with target value and covariance with with features

import pandas as pd

from sklearn.preprocessing import LabelEncoder

# Load the data

data = pd.read\_csv('Property.csv')

data.head()

# Calculate correlation coefficients for the entire dataset

correlation\_matrix = data.corr()

# Calculate variance for the entire dataset

variance = data.var()

# Select the desired features

selected\_features = ['location\_id','page\_url','property\_type','price','price\_bin','location','city','province\_name','locality','latitude','longitude','baths','area','area\_marla','area\_sqft',

                     'purpose','bedrooms','date\_added','year','month','day','agency','agent']

# Applying LabelEncoder to the actual DataFrame columns

label\_encoder = LabelEncoder()

data['location\_id'] = label\_encoder.fit\_transform(data['location\_id'])

data['page\_url'] = label\_encoder.fit\_transform(data['page\_url'])

data['property\_type'] = label\_encoder.fit\_transform(data['property\_type'])

data['price'] = label\_encoder.fit\_transform(data['price'])

data['price\_bin'] = label\_encoder.fit\_transform(data['price\_bin'])

data['location'] = label\_encoder.fit\_transform(data['location'])

data['city'] = label\_encoder.fit\_transform(data['city'])

data['province\_name'] = label\_encoder.fit\_transform(data['province\_name'])

data['locality'] = label\_encoder.fit\_transform(data['locality'])

data['latitude'] = label\_encoder.fit\_transform(data['latitude'])

data['longitude'] = label\_encoder.fit\_transform(data['longitude'])

data['baths'] = label\_encoder.fit\_transform(data['baths'])

data['area'] = label\_encoder.fit\_transform(data['area'])

data['area\_marla'] = label\_encoder.fit\_transform(data['area\_marla'])

data['area\_sqft'] = label\_encoder.fit\_transform(data['area\_sqft'])

data['purpose'] = label\_encoder.fit\_transform(data['purpose'])

data['bedrooms'] = label\_encoder.fit\_transform(data['bedrooms'])

data['date\_added'] = label\_encoder.fit\_transform(data['date\_added'])

data['year'] = label\_encoder.fit\_transform(data['year'])

data['month'] = label\_encoder.fit\_transform(data['month'])

data['day'] = label\_encoder.fit\_transform(data['day'])

data['agency'] = label\_encoder.fit\_transform(data['agency'])

data['agent'] = label\_encoder.fit\_transform(data['agent'])

# Calculate correlation coefficients for the selected features

selected\_corr\_matrix = data[selected\_features].corr()

# Calculate variance for the selected features

selected\_variance = data[selected\_features].var()

# Print the results

print("Correlation Coefficients for the Entire Dataset:")

print(correlation\_matrix)

print("\nVariance for the Entire Dataset:")

print(variance)

print("\nCorrelation Coefficients for the Selected Features:")

print(selected\_corr\_matrix)

print("\nVariance for the Selected Features:")

print(selected\_variance)

5Apply *Mean Absolute Difference (MAD)* method for feature selection on *diabetes* data set and discover the selected features.

#5 Apply Mean Absolute Difference (MAD) method for feature selection on diabetes data set and discover  the  selected  features.

import pandas as pd

# Load data

datas = pd.read\_csv('diabetes.csv')

datas.head()

datas.columns

# Load data

import pandas as pd

datas = 'diabetes.csv'

names = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',

       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],

d= pd.read\_csv(datas, names=names)

# Print the first few rows of the DataFrame

print(d.head())

# Extracting the features

X = d.iloc[:, :4]

# Calculating Mean Absolute Deviation

mad = X.mad(axis=0)

print("Mean absolute deviation of columns:")

print(mad)

6Apply *Dispersion Ratio* method for feature selection on *Churn* data set and discover the selected features.

#6. Apply Dispersion  Ratio  method for feature selection on Churn  data set and discover the selected features

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

# Load data

data = 'Churn\_Modelling.csv'

names = ['Geography','Gender','Age']

df = pd.read\_csv(data, names=names)

# Convert the DataFrame to numeric values

df = df.apply(pd.to\_numeric, errors='coerce')

# Extracting the features

X = df.iloc[:, :4]

X = X + 1

# Calculating Arithmetic Mean

am = np.mean(X, axis=0)

# Calculating Geometric Mean

gm = np.power(np.product(X, axis=0), 1 / X.shape[0])

# Ratio of arithmetic mean and geometric mean

disp\_ratio = am / gm

# Plotting the bar graph

plt.bar(np.arange(X.shape[1]), disp\_ratio, color='teal')

plt.show()

A white rectangular object with black lines

Description automatically generated with medium confidence

Student Name: Sawera Fazal Roll No:21A-026-se Section: 21A

## CS334 - Machine Learning

Lab 05 - Feature Selection Methods in ML (Part - 2) Instructor: Ms. Maham Ashraf

E-mail: [mashraf@uit.edu](mailto:mashraf@uit.edu) Semester: Fall, 2023

**Objective**

The purpose of this lab session is to introduce feature selection methods for machine learning model. This lab is divided into two parts, in Part-1 we have used Filter methods for extracting important features from data sets. In Part-2, we will use Wrapper, Embedded and Hybrid methods for feature selections.

#### Instructions

You have to perform the following tasks yourselves. Raise your hand if you face any difficulty in understanding and solving these tasks. **Plagiarism** is an abhorrent practice and you should not engage in it.

#### How to Submit

* Submit lab work in a single .py file on Microsoft Teams. (No other format will be accepted)
* Lab work file name should be saved with your roll number (e.g. 19a-001-SE\_LW01.py)
* Submit home work in a single .py file on Microsoft Teams. (No other format will be accepted)
* Home work file name should be saved with your roll number (e.g. 19a-001-SE\_HW01.py)

#### Wrapper Methods: for Features Selection

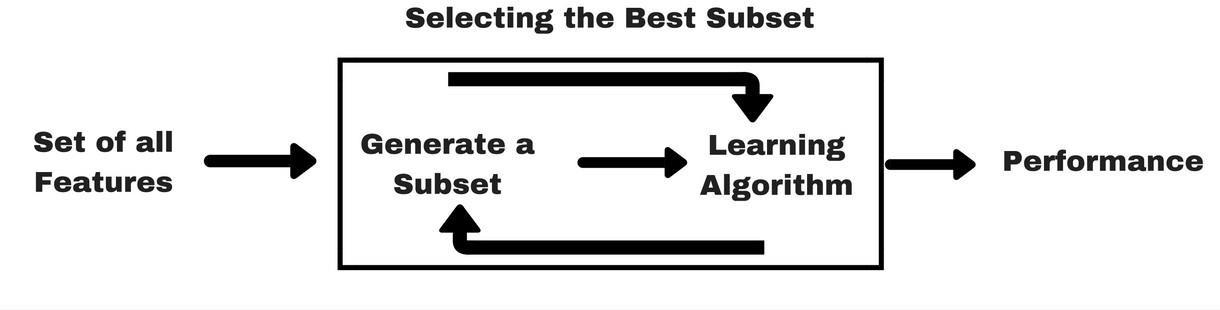


Figure 1: Wrapper Method

Wrappers require some method to search the space of all possible subsets of features, assessing their quality by learning and evaluating a classifier with that feature subset. The feature selection

#### Homework

1. Apply *Forward Feature Selection* methods on **housing** data set and discover the selected features.

#1. Apply Forward Feature Selection methods on housing data set and discover the selected features

# 1. Load  data

import pandas as pd

dataframe = pd.read\_csv("housing.csv")  # Replace 'pima-indians-diabetes.csv' with your dataset's filename

# Assume 'target\_feature' is your target feature, modify it as per your dataset

X\_df = dataframe.drop('MEDV', axis=1)  # Assuming 'target\_feature' is the target feature

y\_series = dataframe['MEDV']  # Assuming 'target\_feature' is the target feature

X = X\_df  # Features

y = y\_series  # Target variable

# 2. Split data into descriptive and target features in X and y variables respectively.

# 3. Load important libraries

from sklearn.linear\_model import LogisticRegression

from mlxtend.feature\_selection import SequentialFeatureSelector as SFS

# 4. Apply Model

lr = LogisticRegression(class\_weight='balanced', solver='lbfgs',random\_state=42, n\_jobs=-1, max\_iter=500)

lr.fit(X, y)

# 5. Select best features

bfs = SFS(lr,

          k\_features='best',

          forward=True,

          floating=False,

          verbose=2,

          scoring='accuracy',

          cv=0)

bfs = bfs.fit(X, y)

# 6. Print feature list

features = list(bfs.k\_feature\_names\_)

print(features)

1. Apply *Backward Feature Elimination* methods on **diabetes** data set and discover the selected features.

#2Apply Backward Feature Elimination methods on diabetes data set and discover the selected features.

from sklearn.neighbors import KNeighborsClassifier

import pandas as pd

dataframe = pd.read\_csv("diabetes.csv")  # Replace 'pima-indians-diabetes.csv' with your dataset's filename

# Assume 'target\_feature' is your target feature, modify it as per your dataset

X\_df = dataframe.drop('Outcome', axis=1)  # Assuming 'target\_feature' is the target feature

y\_series = dataframe['Outcome']  # Assuming 'target\_feature' is the target feature

# 4. Apply Model

lr = LogisticRegression(class\_weight='balanced', solver='lbfgs', random\_state=42, n\_jobs=-1, max\_iter=500)

lr.fit(X, y)

# 5. Select best features

bfs = SFS(lr,

          k\_features='best',

          forward=False,

          floating=False,

          verbose=2,

          scoring='accuracy',

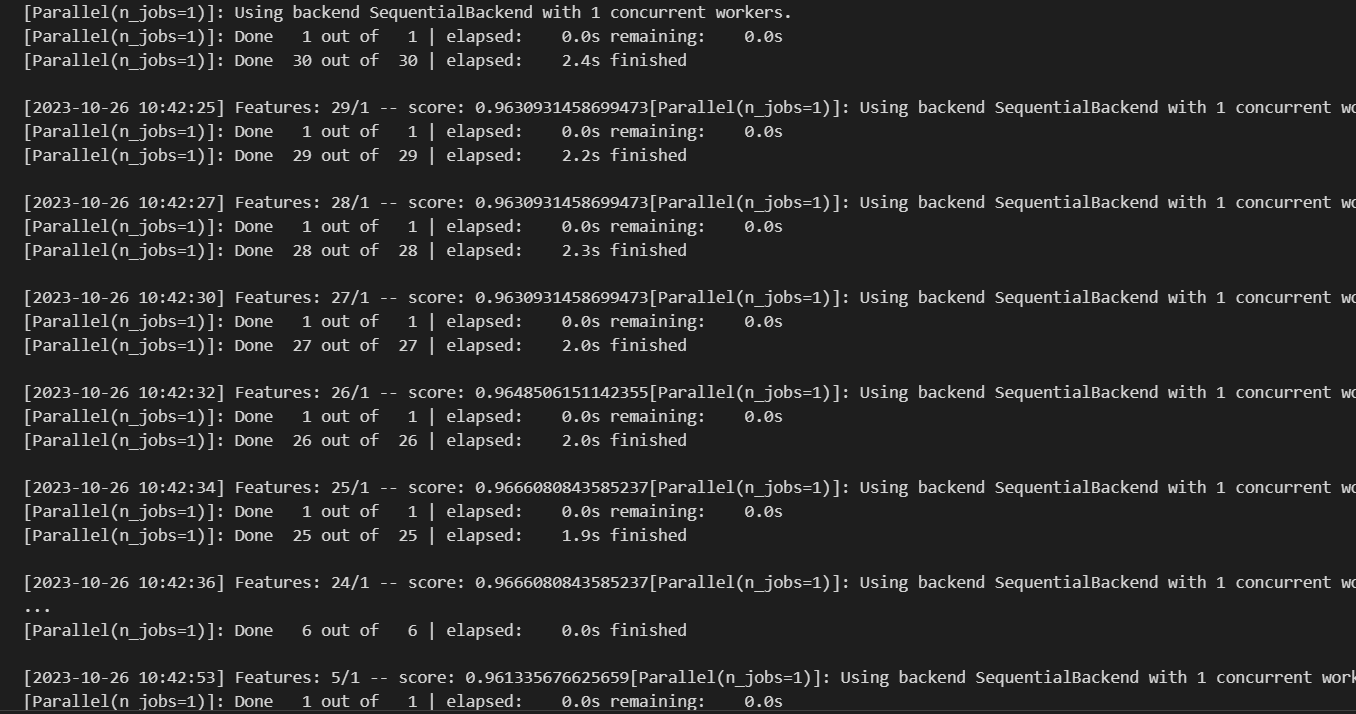
          cv=0)

bfs = bfs.fit(X, y)

# 6. Print feature list

features = list(bfs.k\_feature\_names\_)

print(features)



1. Apply *Exhaustive Feature Selection* methods on **Motor Insurance Fraud** data set and discover the selected features.

#3.  Apply Exhaustive  Feature Selection  methods on Motor Insurance Fraud dataset and discover the  selected features.

from sklearn.neighbors import KNeighborsClassifier

from mlxtend.feature\_selection import ExhaustiveFeatureSelector as EFS

import pandas as pd

dataframee = pd.read\_csv("insurance\_claims.csv")  # Replace 'pima-indians-diabetes.csv' with your dataset's filename

# Assume 'target\_feature' is your target feature, modify it as per your dataset

X\_d = dataframee.drop('Policy\_state', axis=1)  # Assuming 'target\_feature'is the target feature

y\_serie = dataframee['Policy\_state']  # Assuming 'target\_feature' is thetarget feature

# 2 Split data into descriptive and target features in X and y variablesrespectively.

# Assuming X\_df and y\_series are your DataFrame and Series, respectively.

knn = KNeighborsClassifier(n\_neighbors=3)

efs1 = EFS(knn,

           min\_features=1,

           max\_features=4,

           scoring='accuracy',

           print\_progress=True,

           cv=5)

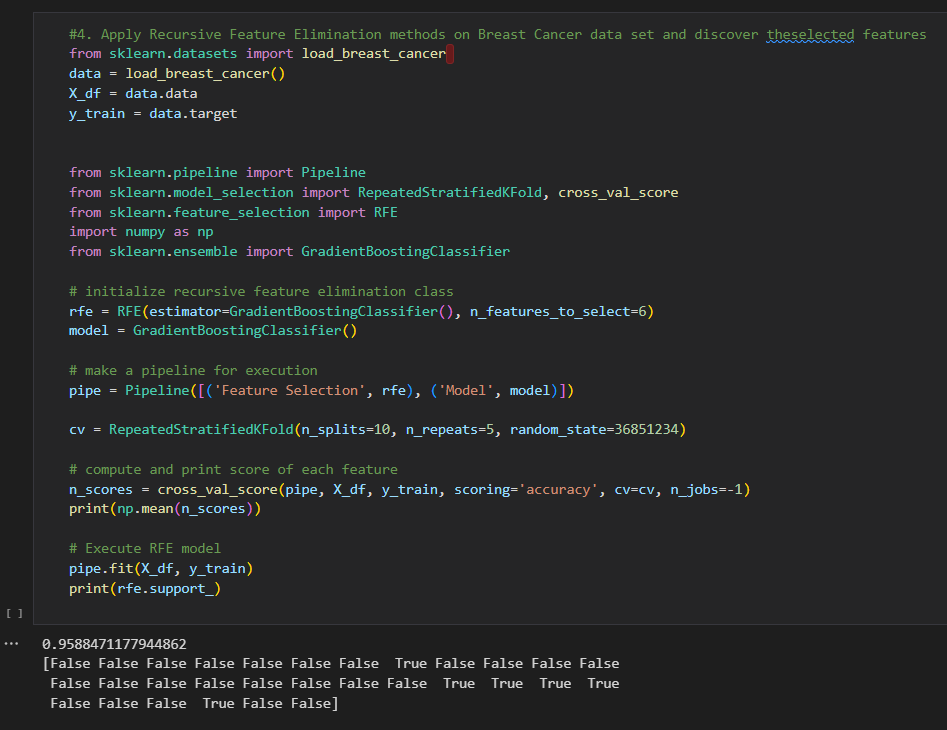
efs1 = efs1.fit(X\_d, y\_serie)

print('Best accuracy score: %.2f' % efs1.best\_score\_)

print('Best subset (indices): ', efs1.best\_idx\_)

print('Best subset (corresponding names): ', efs1.best\_feature\_names\_)

1. Apply *Recursive Feature Elimination* methods on **Breast Cancer** data set and discover the selected features.



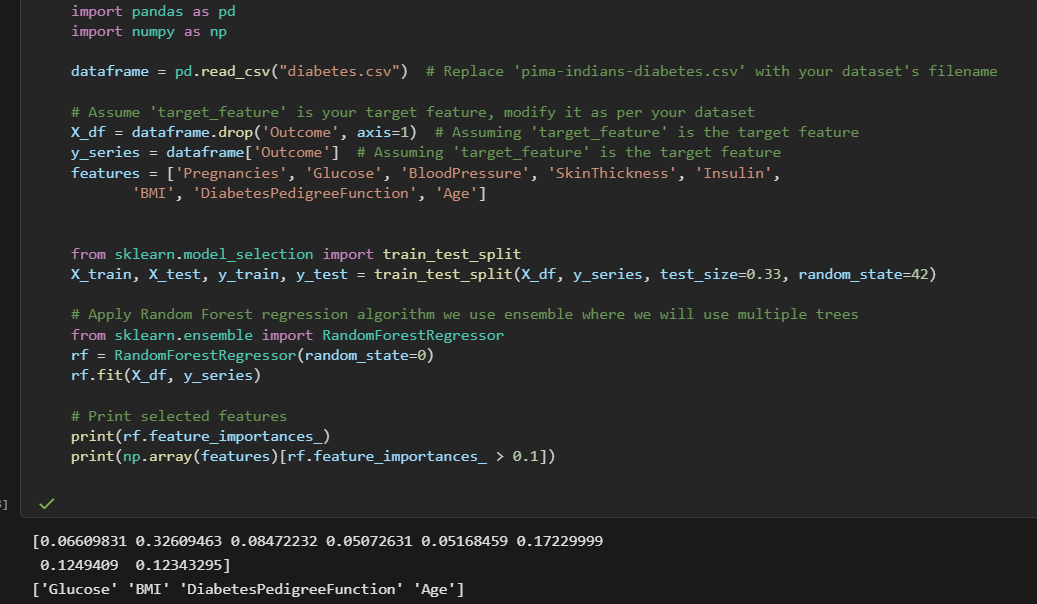
1. Apply *Lasso regression (* 1*L)* method for feature selection on **housing** data set and discover the selected features.

A screen shot of a computer program

Description automatically generatedA screenshot of a computer

Description automatically generated

1. Apply *Random forest features selection* method with 50 random states on **diabetes** data set and discover the selected features.



Student Name: Roll No: Section:

## CS334 - Machine Learning

Lab 06

Instructor: Ms. Maham Ashraf E-mail: [mashraf@uit.edu](mailto:mashraf@uit.edu)

Semester: Fall, 2023

**Objective**

The purpose of this lab session is to introduce learn machine learning model using Decision Tree Algorithms.

#### Instructions

You have to perform the following tasks yourselves. Raise your hand if you face any difficulty in understanding and solving these tasks. **Plagiarism** is an abhorrent practice and you should not engage in it.

#### How to Submit

* Submit lab work in a single .py file on Microsoft Teams. (No other format will be accepted)
* Lab work file name should be saved with your roll number (e.g. 19a-001-SE\_LW01.py)
* Submit home work in a single .py file on Microsoft Teams. (No other format will be accepted)
* Home work file name should be saved with your roll number (e.g. 19a-001-SE\_HW01.py)

#### Lab Tasks

1. Implement *GINI* method to compute the feature entropy.
2. Compute the remainder of the feature by *GINI* method.
3. Compute information gain by *GINI* method.
4. Compute Gain Ratio (GR), for *GINI* method.
5. write the method that print your decision tree

# Lab Task

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris

from sklearn.tree import DecisionTreeClassifier, plot\_tree

class Gini:

    def \_\_init\_\_(self, data, t\_label):

        self.data = data

        self.t\_label = t\_label

        self.clf = None

    def preprocess\_data(self):

        pass

    def get\_feature\_entropy(self, data=None, t\_label=None):

        if data is None:

            data = self.data

        if t\_label is None:

            t\_label = self.t\_label

        target = data[t\_label]

        class\_list = target.unique()

        total\_row = data.shape[0]

        total\_entr = 0

        for c in class\_list:

            total\_class\_count = data[data[t\_label] == c].shape[0]

            total\_class\_entropy = 1 - (total\_class\_count / total\_row) \*\* 2

            total\_entr += total\_class\_entropy

        return total\_entr

    def get\_rem\_by\_entropy(self):

        desc\_features = self.data.drop([self.t\_label], axis=1)

        target\_feature = self.data[self.t\_label]

        target\_list = target\_feature.unique()

        class\_count = desc\_features.shape[0]

        rem\_list = []

        for item in desc\_features.columns:

            rem\_feature\_entropy = 0

            class\_list = desc\_features[item].unique()

            new\_feature = desc\_features[item]

            for level in class\_list:

                label\_class\_count = desc\_features[desc\_features[item] == level].shape[0]

                entropy\_class = 0

                feature\_level\_entropy = 0

                sum\_feature\_entropy = 0

                if label\_class\_count != 0:

                    probability\_class = label\_class\_count / class\_count

                    for tvalue in target\_list:

                        count\_level\_frequency = 0

                        for i in range(class\_count):

                            if (new\_feature[i] == level) and (target\_feature[i] == tvalue):

                                count\_level\_frequency += 1

                        if count\_level\_frequency != 0:

                            feature\_prob = count\_level\_frequency / label\_class\_count

                            feature\_level\_entropy = 1 - (feature\_prob \*\* 2)

                            sum\_feature\_entropy += feature\_level\_entropy

                    prob\_Xfeature\_entropy = probability\_class \* sum\_feature\_entropy

                    rem\_feature\_entropy += prob\_Xfeature\_entropy

            rem\_list.append(rem\_feature\_entropy)

        return rem\_list

    def get\_info\_gain\_by\_entropy(self):

        target\_entropy = self.get\_feature\_entropy()

        rem = self.get\_rem\_by\_entropy()

        IG\_list = [target\_entropy - rem[i] for i in range(len(rem))]

        return IG\_list

    def get\_GR\_by\_entropy(self):

        feature\_list = self.data.drop([self.t\_label], axis=1).columns

        GR\_list = []

        count = 0

        IG = self.get\_info\_gain\_by\_entropy()

        for item in feature\_list:

            feat\_entropy = self.get\_feature\_entropy(None, item)

            GR\_list.append(IG[count] / feat\_entropy)

            count += 1

        return GR\_list

    def build\_decision\_tree(self, max\_depth=None, random\_state=None):

        # Build Decision Tree classifier

        self.clf = DecisionTreeClassifier(criterion='gini', max\_depth=max\_depth, random\_state=random\_state)

        X = self.data.drop(self.t\_label, axis=1)

        y = self.data[self.t\_label]

        self.clf.fit(X, y)

    def visualize\_tree(self):

        if self.clf is not None:

            plt.figure(figsize=(15, 10))

            plot\_tree(self.clf, filled=True, feature\_names=self.data.drop(self.t\_label, axis=1).columns,

                      class\_names=list(map(str, self.clf.classes\_)), rounded=True)

            plt.show()

        else:

            print("Decision tree not built. Please run build\_decision\_tree first.")

# Load the Iris dataset

iris = load\_iris()

iris\_df = pd.DataFrame(data=np.c\_[iris['data'], iris['target']], columns=iris['feature\_names'] + ['target'])

iris\_df['target'] = iris\_df['target'].astype(int)

# Example usage with the Iris dataset

gini\_instance = Gini(iris\_df, 'target')

# Get Information Gain by Entropy

info\_gain\_list = gini\_instance.get\_info\_gain\_by\_entropy()

print("Information Gain by Entropy:", info\_gain\_list)

# Get Gain Ratio by Entropy

gain\_ratio\_list = gini\_instance.get\_GR\_by\_entropy()

print("Gain Ratio by Entropy:", gain\_ratio\_list)

gini\_instance.build\_decision\_tree(max\_depth=4, random\_state=1)

gini\_instance.visualize\_tree()

Output:

# 

#### Homework

1. Modify your ID3 algorithm that can accept continuous (e.g. descriptive and target) features.
2. Compute threshold of each continuous descriptive feature.
3. Select continuous feature for split the tree based on its variance.
4. If the target feature is continuous, split your tree by weighted variance method.
5. write the method that print your decision tree.

# Home Task

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris

from sklearn.tree import DecisionTreeClassifier, plot\_tree

class Gini:

    def \_\_init\_\_(self, data, t\_label):

        self.data = data

        self.t\_label = t\_label

        self.clf = None

    def preprocess\_data(self):

        pass

    def get\_feature\_entropy(self, data=None, t\_label=None):

        if data is None:

            data = self.data

        if t\_label is None:

            t\_label = self.t\_label

        target = data[t\_label]

        class\_list = target.unique()

        total\_row = data.shape[0]

        total\_entr = 0

        for c in class\_list:

            total\_class\_count = data[data[t\_label] == c].shape[0]

            total\_class\_entropy = 1 - (total\_class\_count / total\_row) \*\* 2

            total\_entr += total\_class\_entropy

        return total\_entr

    def get\_rem\_by\_entropy(self):

        desc\_features = self.data.drop([self.t\_label], axis=1)

        target\_feature = self.data[self.t\_label]

        target\_list = target\_feature.unique()

        class\_count = desc\_features.shape[0]

        rem\_list = []

        for item in desc\_features.columns:

            rem\_feature\_entropy = 0

            class\_list = desc\_features[item].unique()

            new\_feature = desc\_features[item]

            for level in class\_list:

                label\_class\_count = desc\_features[desc\_features[item] == level].shape[0]

                entropy\_class = 0

                feature\_level\_entropy = 0

                sum\_feature\_entropy = 0

                if label\_class\_count != 0:

                    probability\_class = label\_class\_count / class\_count

                    for tvalue in target\_list:

                        count\_level\_frequency = 0

                        for i in range(class\_count):

                            if (new\_feature[i] == level) and (target\_feature[i] == tvalue):

                                count\_level\_frequency += 1

                        if count\_level\_frequency != 0:

                            feature\_prob = count\_level\_frequency / label\_class\_count

                            feature\_level\_entropy = 1 - (feature\_prob \*\* 2)

                            sum\_feature\_entropy += feature\_level\_entropy

                    prob\_Xfeature\_entropy = probability\_class \* sum\_feature\_entropy

                    rem\_feature\_entropy += prob\_Xfeature\_entropy

            rem\_list.append(rem\_feature\_entropy)

        return rem\_list

    def get\_info\_gain\_by\_entropy(self):

        target\_entropy = self.get\_feature\_entropy()

        rem = self.get\_rem\_by\_entropy()

        IG\_list = [target\_entropy - rem[i] for i in range(len(rem))]

        return IG\_list

    def get\_GR\_by\_entropy(self):

        feature\_list = self.data.drop([self.t\_label], axis=1).columns

        GR\_list = []

        count = 0

        IG = self.get\_info\_gain\_by\_entropy()

        for item in feature\_list:

            feat\_entropy = self.get\_feature\_entropy(None, item)

            GR\_list.append(IG[count] / feat\_entropy)

            count += 1

        return GR\_list

    def build\_decision\_tree(self, max\_depth=None, random\_state=None):

        # Build Decision Tree classifier

        self.clf = DecisionTreeClassifier(criterion='gini', max\_depth=max\_depth, random\_state=random\_state)

        X = self.data.drop(self.t\_label, axis=1)

        y = self.data[self.t\_label]

        self.clf.fit(X, y)

    def visualize\_tree(self):

        if self.clf is not None:

            plt.figure(figsize=(15, 10))

            plot\_tree(self.clf, filled=True, feature\_names=self.data.drop(self.t\_label, axis=1).columns,

                      class\_names=list(map(str, self.clf.classes\_)), rounded=True)

            plt.show()

        else:

            print("Decision tree not built. Please run build\_decision\_tree first.")

# Load the Iris dataset

iris = load\_iris()

iris\_df = pd.DataFrame(data=np.c\_[iris['data'], iris['target']], columns=iris['feature\_names'] + ['target'])

iris\_df['target'] = iris\_df['target'].astype(int)

# Example usage with the Iris dataset

gini\_instance = Gini(iris\_df, 'target')

# Get Information Gain by Entropy

info\_gain\_list = gini\_instance.get\_info\_gain\_by\_entropy()

print("Information Gain by Entropy:", info\_gain\_list)

# Get Gain Ratio by Entropy

gain\_ratio\_list = gini\_instance.get\_GR\_by\_entropy()

print("Gain Ratio by Entropy:", gain\_ratio\_list)

gini\_instance.build\_decision\_tree(max\_depth=4, random\_state=1)

gini\_instance.visualize\_tree()

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris

from sklearn.tree import DecisionTreeClassifier, plot\_tree

class Gini:

    def \_\_init\_\_(self, data, t\_label):

        self.data = data

        self.t\_label = t\_label

        self.clf = None

    def preprocess\_data(self):

        pass

    def get\_feature\_entropy(self, data=None, t\_label=None):

        if data is None:

            data = self.data

        if t\_label is None:

            t\_label = self.t\_label

        target = data[t\_label]

        class\_list = target.unique()

        total\_row = data.shape[0]

        total\_entr = 0

        for c in class\_list:

            total\_class\_count = data[data[t\_label] == c].shape[0]

            total\_class\_entropy = 1 - (total\_class\_count / total\_row) \*\* 2

            total\_entr += total\_class\_entropy

        return total\_entr

    def get\_rem\_by\_entropy(self):

        desc\_features = self.data.drop([self.t\_label], axis=1)

        target\_feature = self.data[self.t\_label]

        target\_list = target\_feature.unique()

        class\_count = desc\_features.shape[0]

        rem\_list = []

        for item in desc\_features.columns:

            rem\_feature\_entropy = 0

            class\_list = desc\_features[item].unique()

            new\_feature = desc\_features[item]

            for level in class\_list:

                label\_class\_count = desc\_features[desc\_features[item] == level].shape[0]

                entropy\_class = 0

                feature\_level\_entropy = 0

                sum\_feature\_entropy = 0

                if label\_class\_count != 0:

                    probability\_class = label\_class\_count / class\_count

                    for tvalue in target\_list:

                        count\_level\_frequency = 0

                        for i in range(class\_count):

                            if (new\_feature[i] == level) and (target\_feature[i] == tvalue):

                                count\_level\_frequency += 1

                        if count\_level\_frequency != 0:

                            feature\_prob = count\_level\_frequency / label\_class\_count

                            feature\_level\_entropy = 1 - (feature\_prob \*\* 2)

                            sum\_feature\_entropy += feature\_level\_entropy

                    prob\_Xfeature\_entropy = probability\_class \* sum\_feature\_entropy

                    rem\_feature\_entropy += prob\_Xfeature\_entropy

            rem\_list.append(rem\_feature\_entropy)

        return rem\_list

    def get\_info\_gain\_by\_entropy(self):

        target\_entropy = self.get\_feature\_entropy()

        rem = self.get\_rem\_by\_entropy()

        IG\_list = [target\_entropy - rem[i] for i in range(len(rem))]

        return IG\_list

    def get\_GR\_by\_entropy(self):

        feature\_list = self.data.drop([self.t\_label], axis=1).columns

        GR\_list = []

        count = 0

        IG = self.get\_info\_gain\_by\_entropy()

        for item in feature\_list:

            feat\_entropy = self.get\_feature\_entropy(None, item)

            GR\_list.append(IG[count] / feat\_entropy)

            count += 1

        return GR\_list

    def build\_decision\_tree(self, max\_depth=None, random\_state=None):

        # Build Decision Tree classifier

        self.clf = DecisionTreeClassifier(criterion='gini', max\_depth=max\_depth, random\_state=random\_state)

        X = self.data.drop(self.t\_label, axis=1)

        y = self.data[self.t\_label]

        self.clf.fit(X, y)

    def visualize\_tree(self):

        if self.clf is not None:

            plt.figure(figsize=(15, 10))

            plot\_tree(self.clf, filled=True, feature\_names=self.data.drop(self.t\_label, axis=1).columns,

                      class\_names=list(map(str, self.clf.classes\_)), rounded=True)

            plt.show()

        else:

            print("Decision tree not built. Please run build\_decision\_tree first.")

# Load the Iris dataset

iris = load\_iris()

iris\_df = pd.DataFrame(data=np.c\_[iris['data'], iris['target']], columns=iris['feature\_names'] + ['target'])

iris\_df['target'] = iris\_df['target'].astype(int)

# Example usage with the Iris dataset

gini\_instance = Gini(iris\_df, 'target')

# Get Information Gain by Entropy

info\_gain\_list = gini\_instance.get\_info\_gain\_by\_entropy()

print("Information Gain by Entropy:", info\_gain\_list)

# Get Gain Ratio by Entropy

gain\_ratio\_list = gini\_instance.get\_GR\_by\_entropy()

print("Gain Ratio by Entropy:", gain\_ratio\_list)

gini\_instance.build\_decision\_tree(max\_depth=4, random\_state=1)

gini\_instance.visualize\_tree()

Output:

A screenshot of a computer

Description automatically generated

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris

from sklearn.tree import DecisionTreeClassifier, plot\_tree

class Gini:

    def \_\_init\_\_(self, data, t\_label):

        self.data = data

        self.t\_label = t\_label

        self.clf = None

    def preprocess\_data(self):

        pass

    def get\_feature\_entropy(self, data=None, t\_label=None):

        if data is None:

            data = self.data

        if t\_label is None:

            t\_label = self.t\_label

        target = data[t\_label]

        class\_list = target.unique()

        total\_row = data.shape[0]

        total\_entr = 0

        for c in class\_list:

            total\_class\_count = data[data[t\_label] == c].shape[0]

            total\_class\_entropy = 1 - (total\_class\_count / total\_row) \*\* 2

            total\_entr += total\_class\_entropy

        return total\_entr

    def get\_rem\_by\_entropy(self):

        desc\_features = self.data.drop([self.t\_label], axis=1)

        target\_feature = self.data[self.t\_label]

        target\_list = target\_feature.unique()

        class\_count = desc\_features.shape[0]

        rem\_list = []

        for item in desc\_features.columns:

            rem\_feature\_entropy = 0

            class\_list = desc\_features[item].unique()

            new\_feature = desc\_features[item]

            for level in class\_list:

                label\_class\_count = desc\_features[desc\_features[item] == level].shape[0]

                entropy\_class = 0

                feature\_level\_entropy = 0

                sum\_feature\_entropy = 0

                if label\_class\_count != 0:

                    probability\_class = label\_class\_count / class\_count

                    for tvalue in target\_list:

                        count\_level\_frequency = 0

                        for i in range(class\_count):

                            if (new\_feature[i] == level) and (target\_feature[i] == tvalue):

                                count\_level\_frequency += 1

                        if count\_level\_frequency != 0:

                            feature\_prob = count\_level\_frequency / label\_class\_count

                            feature\_level\_entropy = 1 - (feature\_prob \*\* 2)

                            sum\_feature\_entropy += feature\_level\_entropy

                    prob\_Xfeature\_entropy = probability\_class \* sum\_feature\_entropy

                    rem\_feature\_entropy += prob\_Xfeature\_entropy

            rem\_list.append(rem\_feature\_entropy)

        return rem\_list

    def get\_info\_gain\_by\_entropy(self):

        target\_entropy = self.get\_feature\_entropy()

        rem = self.get\_rem\_by\_entropy()

        IG\_list = [target\_entropy - rem[i] for i in range(len(rem))]

        return IG\_list

    def get\_GR\_by\_entropy(self):

        feature\_list = self.data.drop([self.t\_label], axis=1).columns

        GR\_list = []

        count = 0

        IG = self.get\_info\_gain\_by\_entropy()

        for item in feature\_list:

            feat\_entropy = self.get\_feature\_entropy(None, item)

            GR\_list.append(IG[count] / feat\_entropy)

            count += 1

        return GR\_list

    def build\_decision\_tree(self, max\_depth=None, random\_state=None):

        # Build Decision Tree classifier

        self.clf = DecisionTreeClassifier(criterion='gini', max\_depth=max\_depth, random\_state=random\_state)

        X = self.data.drop(self.t\_label, axis=1)

        y = self.data[self.t\_label]

        self.clf.fit(X, y)

    def visualize\_tree(self):

        if self.clf is not None:

            plt.figure(figsize=(15, 10))

            plot\_tree(self.clf, filled=True, feature\_names=self.data.drop(self.t\_label, axis=1).columns,

                      class\_names=list(map(str, self.clf.classes\_)), rounded=True)

            plt.show()

        else:

            print("Decision tree not built. Please run build\_decision\_tree first.")

# Load the Iris dataset

iris = load\_iris()

iris\_df = pd.DataFrame(data=np.c\_[iris['data'], iris['target']], columns=iris['feature\_names'] + ['target'])

iris\_df['target'] = iris\_df['target'].astype(int)

# Example usage with the Iris dataset

gini\_instance = Gini(iris\_df, 'target')

# Get Information Gain by Entropy

info\_gain\_list = gini\_instance.get\_info\_gain\_by\_entropy()

print("Information Gain by Entropy:", info\_gain\_list)

# Get Gain Ratio by Entropy

gain\_ratio\_list = gini\_instance.get\_GR\_by\_entropy()

print("Gain Ratio by Entropy:", gain\_ratio\_list)

gini\_instance.build\_decision\_tree(max\_depth=4, random\_state=1)

gini\_instance.visualize\_tree()

Output:

A computer screen shot of a diagram

Description automatically generated

.

Student Name: Sawera Fazal Roll No:21A-026-se Section: 21A

## CS334 - Machine Learning

Lab 07 - Similarity Based Learning Instructor: Ms. Maham Ashraf

E-mail: [mashraf@uit.edu](mailto:mashraf@uit.edu) Semester: Fall, 2023

**Objective**

The purpose of this lab session is to introduce similarity based learning models, such as Nearest Neighbor (NN), KNN, and Weighted KNN .

#### Instructions

You have to perform the following tasks yourselves. Raise your hand if you face any difficulty in understanding and solving these tasks. **Plagiarism** is an abhorrent practice and you should not engage in it.

#### How to Submit

* Submit lab work in a single .py file on Microsoft Teams. (No other format will be accepted)
* Lab work file name should be saved with your roll number (e.g. 19a-001-SE\_LW01.py)
* Submit home work in a single .py file on Microsoft Teams. (No other format will be accepted)
* Home work file name should be saved with your roll number (e.g. 19a-001-SE\_HW01.py)

#### K Nearest Neighbor Algorithm

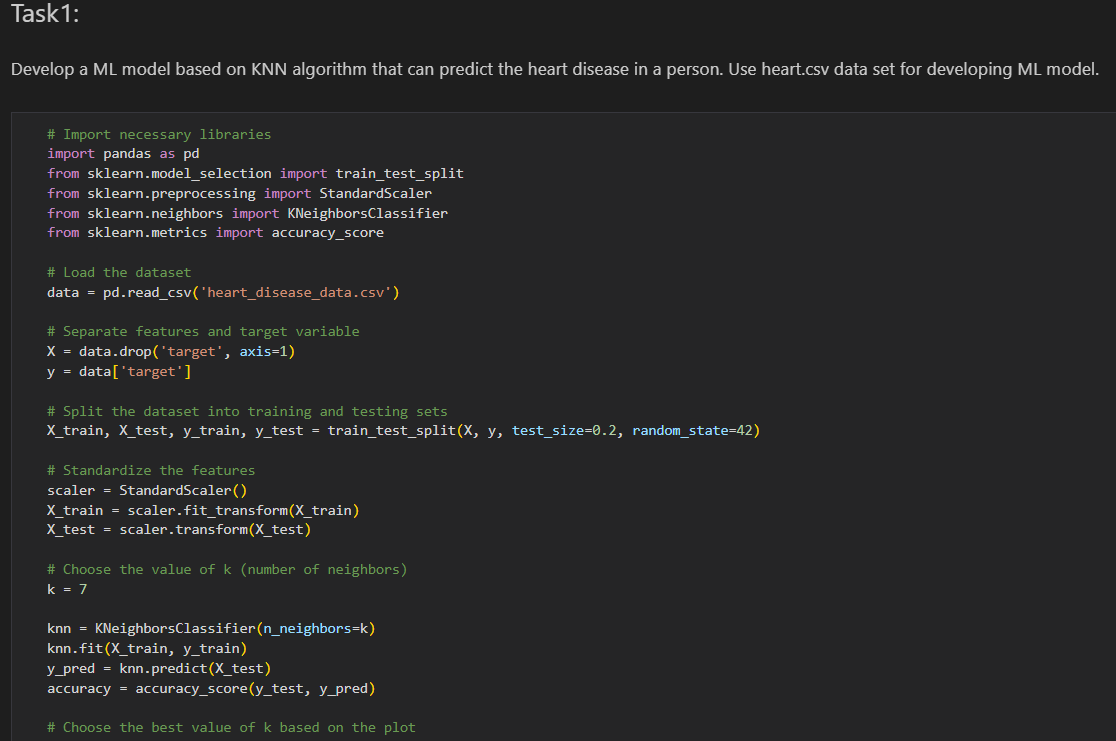
K Nearest Neighbor algorithm falls under the Supervised Learning category and is used for classi- fication (most commonly) and regression. It is a versatile algorithm also used for imputing missing values and resampling datasets. As the name (K Nearest Neighbor) suggests it considers K Nearest Neighbors (Data points) to predict the class or continuous value for the new Data point.

The algorithm’s learning is:

* 1. Instance-based learning: Here we do not learn weights from training data to predict output (as in model-based algorithms) but use entire training instances to predict output for unseen data.
  2. Lazy Learning: Model is not learned using training data prior and the learning process is postponed to a time when prediction is requested on the new instance.
  3. Non -Parametric: In KNN, there is no predefined form of the mapping function.

#### Lab Tasks

* 1. Develop a ML model based on KNN algorithm that can predict the heart disease in a person

 A screenshot of a computer code

Description automatically generated.

Use heart.csv data set for developing ML model.

2. Using Cosine similarity matrix, develop a movie recommender system that recommend similar

types of movies based on their genre, plot, and language. Use IMDBdata\_MainData.csv dataset for the recommender system.

import pandas as pd

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.metrics.pairwise import cosine\_similarity

# Load the dataset

movies\_data = pd.read\_csv('imdb\_movies.csv')

# Combine relevant features into a single column for text-based similarity

movies\_data['combined\_features'] = movies\_data['genre'] + ' ' + movies\_data['overview'] + ' ' + movies\_data['orig\_lang']

# Handle missing values in 'combined\_features'

movies\_data['combined\_features'].fillna('', inplace=True)

# Use CountVectorizer to convert text data into numerical vectors

count\_vectorizer = CountVectorizer(stop\_words='english')

count\_matrix = count\_vectorizer.fit\_transform(movies\_data['combined\_features'])

# Compute cosine similarity matrix

cosine\_sim\_matrix = cosine\_similarity(count\_matrix, count\_matrix)

# Function to get movie recommendations based on similarity

def get\_recommendations(movie\_title, cosine\_sim\_matrix, movies\_data):

    movie\_index = movies\_data.index[movies\_data['names'] == movie\_title].tolist()[0]

    similar\_movies = list(enumerate(cosine\_sim\_matrix[movie\_index]))

    # Sort the movies based on similarity scores

    sorted\_similar\_movies = sorted(similar\_movies, key=lambda x: x[1], reverse=True)

    # Get the top 5 recommendations (excluding the input movie itself)

    recommended\_movies = [(movies\_data['names'][i], sorted\_similar\_movies[i][1]) for i in range(1, 6)]

    return recommended\_movies

# Example: Get recommendations for a movie title

movie\_title = "Heart and Souls"

recommendations = get\_recommendations(movie\_title, cosine\_sim\_matrix, movies\_data)

# Display the recommendations

print(f"Recommendations for '{movie\_title}':")

for movie, similarity in recommendations:

    print(f"{movie} (Similarity Score: {similarity:.2f})")

A screen shot of a computer

Description automatically generated

3. Use Mahalanobis distance and detects the outlier rows in the Diabetes data set. Choose 95%

confidence interval for detecting outlier rows.

A screenshot of a computer program

Description automatically generated A screenshot of a computer screen

Description automatically generated

1. Draw KD-tree using Speed-Agility data.

A screenshot of a computer program

Description automatically generated A screen shot of a graph

Description automatically generated

5. plot KD-tree partitions on graph using Speed-Agility data

import pandas as pd

import numpy as np

from sklearn.neighbors import KDTree

import matplotlib.pyplot as plt

# Load your dataset

data = pd.read\_excel('toughestsport.xlsx')

df = pd.DataFrame(data)

# Extract 'SPD' and 'AGI' columns

x = df['SPD']

y = df['AGI']

# Create a list of points

X = np.array(list(zip(x, y)))

# Build KD-tree

tree = KDTree(X, leaf\_size=2)

# Function to plot KD-tree partitions

def plot\_kdtree\_partitions(node, depth, bbox):

    if node.data.shape[0] > 1:

        bbox = np.array(bbox).reshape(1, -1)  # Reshape bbox to 2D

        left\_indices = node.query(bbox[:, [0, 2]], return\_distance=False)[0]

        right\_indices = node.query(bbox[:, [1, 3]], return\_distance=False)[0]

        left\_bbox = [np.min(X[left\_indices, 0]), np.max(X[left\_indices, 0]), np.min(X[left\_indices, 1]), np.max(X[left\_indices, 1])]

        right\_bbox = [np.min(X[right\_indices, 0]), np.max(X[right\_indices, 0]), np.min(X[right\_indices, 1]), np.max(X[right\_indices, 1])]

        if depth % 2 == 0:

            plt.plot([X[left\_indices[-1], 0], X[left\_indices[-1], 0]], [bbox[0, 2], bbox[0, 3]], c='darkgreen', linestyle='--', linewidth=0.8)

        else:

            plt.plot([bbox[0, 0], bbox[0, 1]], [X[left\_indices[-1], 1], X[left\_indices[-1], 1]], c='darkgreen', linestyle='--', linewidth=0.8)

        plot\_kdtree\_partitions(tree.query(X[left\_indices], return\_distance=False), depth + 1, left\_bbox)

        if depth % 2 == 0:

            plt.plot([X[right\_indices[0], 0], X[right\_indices[0], 0]], [bbox[0, 2], bbox[0, 3]], c='darkgreen', linestyle='--', linewidth=0.8)

        else:

            plt.plot([bbox[0, 0], bbox[0, 1]], [X[right\_indices[0], 1], X[right\_indices[0], 1]], c='darkgreen', linestyle='--', linewidth=0.8)

        plot\_kdtree\_partitions(tree.query(X[right\_indices], return\_distance=False), depth + 1, right\_bbox)

# Scatter plot of the dataset

plt.scatter(x, y, color='orange', marker='o', label='Dataset Points')

# Plot KD-tree partitions

plot\_kdtree\_partitions(tree, 0, plt.axis())

plt.xlabel('Speed')

plt.ylabel('Agility')

plt.title('KD-tree Partitions')

plt.legend()

plt.show()

A screen shot of a graph

Description automatically generated

Student Name: sawera Roll No: 21A-026-se Section: 21A

CS334 - Machine Learning

Lab 08 - Ensemble Models (Part-2)

Instructor: Ms. Maham Ashraf E-mail: mashraf@uit.edu

Semester: Fall, 2023

# Objective

The purpose of this lab session is to introduce Ensemble models that use in Machine Learning for better performance.

# Instructions

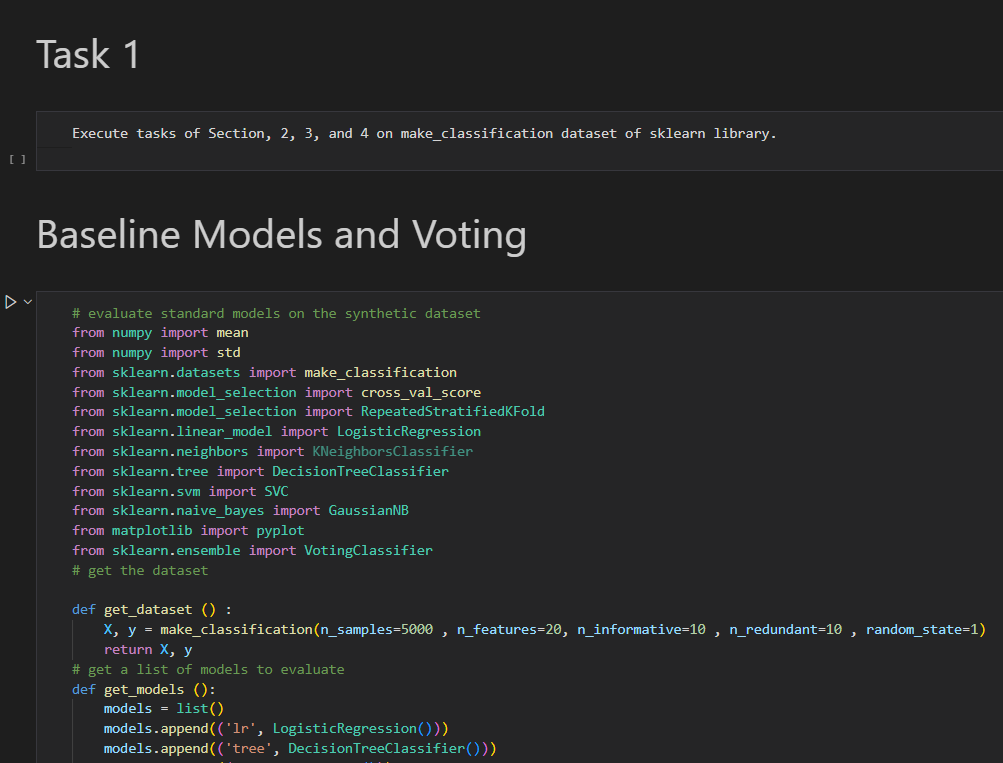
You have to perform the following tasks yourselves. Raise your hand if you face any difficulty in understanding and solving these tasks. **Plagiarism** is an abhorrent practice and you should not engage in it.

# How to Submit

* Submit lab work in a single .py file on Microsoft Teams. (No other format will be accepted)
* Lab work file name should be saved with your roll number (e.g. 19a-001-SE\_LW01.py)
* Submit home work in a single .py file on Microsoft Teams. (No other format will be accepted)
* Home work file name should be saved with your roll number (e.g. 19a-001-SE\_HW01.py)

## Lab Tasks

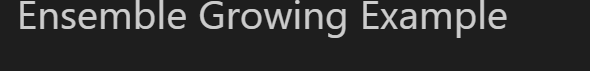
1. Execute tasks of Section, 2, 3, and 4 on *make\_classification* dataset of sklearn library.
2. Train and evaluate models as discussed in Section 2, 3, and 4 on *Diabetes* data set select the bestmodel based on evaluation scores.

* 
* # evaluate standard models on the synthetic dataset
* from numpy import mean
* from numpy import std
* from sklearn.datasets import make\_classification
* from sklearn.model\_selection import cross\_val\_score
* from sklearn.model\_selection import RepeatedStratifiedKFold
* from sklearn.linear\_model import LogisticRegression
* from sklearn.neighbors import KNeighborsClassifier
* from sklearn.tree import DecisionTreeClassifier
* from sklearn.svm import SVC
* from sklearn.naive\_bayes import GaussianNB
* from matplotlib import pyplot
* from sklearn.ensemble import VotingClassifier
* # get the dataset
* def get\_dataset () :
* X, y = make\_classification(n\_samples=5000 , n\_features=20, n\_informative=10 , n\_redundant=10 , random\_state=1)
* return X, y
* # get a list of models to evaluate
* def get\_models ():
* models = list()
* models.append(('lr', LogisticRegression()))
* models.append(('tree', DecisionTreeClassifier()))
* models.append (('nb', GaussianNB()))
* models.append(('svm', SVC(probability=True)))
* return models
* # evaluate a give model using cross - validation
* def evaluate\_model (model , X, y):
* # define the model evaluation procedure
* cv = RepeatedStratifiedKFold( n\_splits=10 , n\_repeats=3 , random\_state=1)
* # evaluate the model
* scores = cross\_val\_score(model, X, y, scoring='accuracy', cv=cv, n\_jobs= -1)
* return scores
* # define dataset
* X, y = get\_dataset()
* # get the models to evaluate
* models = get\_models()
* # evaluate the models and store results
* results , names = list() , list()
* for name , model in models :
* # evaluate model
* scores =evaluate\_model ( model ,X , y)
* # Store results
* results.append(scores)
* names.append(name)
* #summarize result
* print('>%s %.3f (%.3f)' % (name, mean(scores), std(scores)))
* # plot model performance for comparison
* pyplot.boxplot (results , labels=names , showmeans=True )
* pyplot.show ()
* #create the ensemble
* ensemble = VotingClassifier( estimators = models , voting = 'soft')
* # define the evaluation procedure
* cv = RepeatedStratifiedKFold ( n\_splits =10 , n\_repeats =3 , random\_state =1)
* # evaluate the ensemble
* scores = cross\_val\_score (ensemble , X, y, scoring ='accuracy', cv=cv , n\_jobs=-1)
* # summarize the result
* print('Mean Accuracy: %.3f (%.3f)' % (mean(scores), std(scores)))

A screenshot of a computer screen

Description automatically generated A computer screen shot of a program code

Description automatically generated A screenshot of a computer program

Description automatically generated  # perform a single round of growing the ensemble

def grow\_round (models\_in , models\_candidate , X, y):

    # establish a baseline

    baseline = evaluate\_ensemble (models\_in , X, y)

    best\_score , addition = baseline , None

    # enumerate adding each candidate and see if we can improve performance

    for m in models\_candidate :

        # copy the list of chosen models

        dup = models\_in.copy()

        # add the candidate

        dup.append(m)

        # evaluate new ensemble

        result = evaluate\_ensemble(dup,X,y)

        # check for new best

        if result > best\_score :

            # store the new best

            best\_score , addition = result , m

    return best\_score , addition

# grow an ensemble from scratch

def grow\_ensemble (models , X, y):

    best\_score , best\_list = 0.0 , list()

    # grow ensemble until no further improvement

    while True :

        # add one model to the ensemble

        score , addition = grow\_round (best\_list ,models ,X, y)

        # check for no improvement

        if addition is None :

            print ('>no further improvement')

            break

        # keep track of best score

        best\_score = score

        # remove new model from the list of candidates

        models.remove(addition)

        # add new model to the list of models in the ensemble

        best\_list.append(addition)

        # report results along the way

        names = ','. join ([n for n,\_ in best\_list ])

        print ('>%.3f (%s)' % (score , names ))

    return best\_score , best\_list

# define dataset

X, y = get\_dataset ()

# get the models to evaluate

models = get\_models ()

# prune the ensemble

score , model\_list = grow\_ensemble(models,X, y)

names = ','. join ([ n for n , \_ in model\_list ])

print ('Models : %s '% names )

print ('Final Mean Accuracy : %.3f' % score )

A black screen with text on it

Description automatically generated # evaluate standard models on the synthetic dataset

from numpy import mean

from numpy import std

from sklearn.model\_selection import cross\_val\_score

from sklearn.model\_selection import RepeatedStratifiedKFold

from sklearn.linear\_model import LogisticRegression

from sklearn.neighbors import KNeighborsClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.svm import SVC

from sklearn.naive\_bayes import GaussianNB

from matplotlib import pyplot

from sklearn.ensemble import VotingClassifier

from sklearn.datasets import load\_diabetes

import pandas as pd

# get the dataset

def get\_dataset () :

    # Import the dataset

    data = pd.read\_csv('diabetes.csv')

    # Extracting features and target variable

    X = data.drop("Outcome", axis=1)

    y = data['Outcome']

    return X, y

# get a list of models to evaluate

def get\_models ():

    models = list()

    models.append(('lr', LogisticRegression()))

    models.append(('tree', DecisionTreeClassifier()))

    models.append (('nb', GaussianNB()))

    models.append(('svm', SVC(probability=True)))

    return models

# evaluate a give model using cross - validation

def evaluate\_model (model , X, y):

    # define the model evaluation procedure

    cv = RepeatedStratifiedKFold( n\_splits=10 , n\_repeats=3 , random\_state=1)

    # evaluate the model

    scores = cross\_val\_score(model, X, y, scoring='accuracy', cv=cv, n\_jobs= -1)

    return scores

# define dataset

X, y = get\_dataset()

# get the models to evaluate

models = get\_models()

# evaluate the models and store results

results , names = list() , list()

for name , model in models :

    # evaluate model

    scores =evaluate\_model ( model ,X , y)

    # Store results

    results.append(scores)

    names.append(name)

    #summarize result

    print('>%s %.3f (%.3f)' % (name, mean(scores), std(scores)))

# plot model performance for comparison

pyplot.boxplot (results , labels=names , showmeans=True )

pyplot.show ()

#create the ensemble

ensemble = VotingClassifier( estimators = models , voting = 'soft')

# define the evaluation procedure

cv = RepeatedStratifiedKFold ( n\_splits =10 , n\_repeats =3 , random\_state =1)

# evaluate the ensemble

scores = cross\_val\_score (ensemble , X, y, scoring ='accuracy', cv=cv , n\_jobs=-1)

# summarize the result

print('Mean Accuracy: %.3f (%.3f)' % (mean(scores), std(scores)))

A screenshot of a graph

Description automatically generated # evaluate a list of models

def evaluate\_ensemble (models , X, y):

    # check for no models

    if len(models) == 0:

        return 0.0

    # create the ensemble

    ensemble = VotingClassifier(estimators = models , voting = 'soft')

    # define the evaluation procedure

    cv = RepeatedStratifiedKFold ( n\_splits=10 , n\_repeats=3 , random\_state=1)

    # evaluate the ensemble

    scores = cross\_val\_score (ensemble , X, y, scoring ='accuracy', cv=cv ,n\_jobs= -1)

    # return mean score

    return mean(scores)

# perform a single round of pruning the ensemble

def prune\_round (models\_in , X, y):

    # establish a baseline

    baseline = evaluate\_ensemble (models\_in , X, y)

    best\_score , removed = baseline , None

    # enumerate removing each candidate and see if we can improve performance

    for m in models\_in :

        # copy the list of chosen models

        dup = models\_in.copy()

        # remove this model

        dup.remove(m)

        #evaluate new ensemble

        result = evaluate\_ensemble (dup, X , y)

        # check for new best

        if result > best\_score :

            # store the new best

            best\_score , removed = result , m

    return best\_score , removed

# prune an ensemble from scratch

def prune\_ensemble ( models , X , y):

    best\_score = 0.0

    # prune ensemble until no further improvement

    while True :

        # remove one model to the ensemble

        score , removed = prune\_round(models ,X, y)

        # check for no improvement

        if removed is None :

            print ('>no further improvement')

            break

        # keep track of best score

        best\_score = score

        # remove model from the list

        models.remove( removed )

        # report results along the way

        print('>%.3f (removed: %s)' % (score, removed[0]))

        return best\_score , models

# define dataset

X, y = get\_dataset ()

# get the models to evaluate

models = get\_models ()

# prune the ensemble

score , model\_list = prune\_ensemble(models,X, y)

names = ','. join ([ n for n , \_ in model\_list ])

print ('Models : %s '% names )

print ('Final Mean Accuracy : %.3f' % score )

# perform a single round of growing the ensemble

def grow\_round (models\_in , models\_candidate , X, y):

    # establish a baseline

    baseline = evaluate\_ensemble (models\_in , X, y)

    best\_score , addition = baseline , None

    # enumerate adding each candidate and see if we can improve performance

    for m in models\_candidate :

        # copy the list of chosen models

        dup = models\_in.copy()

        # add the candidate

        dup.append(m)

        # evaluate new ensemble

        result = evaluate\_ensemble(dup,X,y)

        # check for new best

        if result > best\_score :

            # store the new best

            best\_score , addition = result , m

    return best\_score , addition

# grow an ensemble from scratch

def grow\_ensemble (models , X, y):

    best\_score , best\_list = 0.0 , list()

    # grow ensemble until no further improvement

    while True :

        # add one model to the ensemble

        score , addition = grow\_round (best\_list ,models ,X, y)

        # check for no improvement

        if addition is None :

            print ('>no further improvement')

            break

        # keep track of best score

        best\_score = score

        # remove new model from the list of candidates

        models.remove(addition)

        # add new model to the list of models in the ensemble

        best\_list.append(addition)

        # report results along the way

        names = ','. join ([n for n,\_ in best\_list ])

        print ('>%.3f (%s)' % (score , names ))

    return best\_score , best\_list

# define dataset

X, y = get\_dataset ()

# get the models to evaluate

models = get\_models ()

# prune the ensemble

score , model\_list = grow\_ensemble(models,X, y)

names = ','. join ([ n for n , \_ in model\_list ])

print ('Models : %s '% names )

print ('Final Mean Accuracy : %.3f' % score )

A screenshot of a computer program

Description automatically generated

Student Name: Roll No: Section:

CS334 - Machine Learning

Lab 09 - Ensemble Models (Part-2)

Instructor: Ms. Maham Ashraf E-mail: mashraf@uit.edu

Semester: Fall, 2023

# Objective

The purpose of this lab session is to introduce Ensemble models that use in Machine Learning for better performance.

# Instructions

You have to perform the following tasks yourselves. Raise your hand if you face any difficulty in understanding and solving these tasks. **Plagiarism** is an abhorrent practice and you should not engage in it.

# How to Submit

* Submit lab work in a single .py file on Microsoft Teams. (No other format will be accepted)
* Lab work file name should be saved with your roll number (e.g. 19a-001-SE\_LW01.py)
* Submit home work in a single .py file on Microsoft Teams. (No other format will be accepted)
* Home work file name should be saved with your roll number (e.g. 19a-001-SE\_HW01.py)

# Lab Tasks

1. Use *BaggingClassifier* and *RandomForestClassifier* from *sklearn* library and implement them on *diabetes* data set.
2. from sklearn.model\_selection import KFold, cross\_val\_score
3. from sklearn.ensemble import BaggingClassifier
4. from sklearn.tree import DecisionTreeClassifier
5. from sklearn.model\_selection import train\_test\_split
6. from sklearn . ensemble import RandomForestClassifier
7. from sklearn.datasets import load\_diabetes
8. from sklearn.metrics import f1\_score, accuracy\_score
9. from sklearn.metrics import confusion\_matrix
10. import seaborn as sns
11. import matplotlib.pyplot as plt
12. import pandas as pd
13. dataframe = pd.read\_csv("diabetes.csv")
14. X = dataframe.drop('Outcome', axis=1)
15. y = dataframe['Outcome']
16. X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=2020)
17. max\_features = 3
18. kfold = KFold(n\_splits=10, shuffle=True, random\_state=2020)
19. decision\_tree = DecisionTreeClassifier(max\_features = max\_features)
20. num\_trees = 100
21. # Decision Tree base estimator
22. bagging\_model = BaggingClassifier(base\_estimator=decision\_tree, n\_estimators = num\_trees, random\_state=2020)
23. results = cross\_val\_score(bagging\_model, X\_train, y\_train, cv=kfold)
24. print("Bagging classifier Accuracy: %0.2f (+/- %0.2f)" % (results.mean(), results.std()))
25. # Bagging Classifier with Decision Tree base estimator
26. bagging\_model.fit(X\_train, y\_train)
27. y\_pred\_bagging = bagging\_model.predict(X\_test)
28. accuracy\_bagging = accuracy\_score(y\_test, y\_pred\_bagging)
29. f1\_bagging = f1\_score(y\_test, y\_pred\_bagging)
30. print("Bagging Classifier Accuracy: %0.2f" % accuracy\_bagging)
31. print("Bagging Classifier F1 Score: %0.2f" % f1\_bagging)
32. # Bagging Classifier with Decision Tree base estimator
33. conf\_matrix\_bagging = confusion\_matrix(y\_test, y\_pred\_bagging)
34. # Plot confusion matrix for Bagging Classifier
35. plt.figure(figsize=(4, 4))
36. sns.heatmap(conf\_matrix\_bagging, annot=True, fmt="d", cmap="Blues", cbar=False,
37. xticklabels=['Class 0', 'Class 1'], yticklabels=['Class 0', 'Class 1'])
38. plt.title("Confusion Matrix - Bagging Classifier")
39. plt.xlabel("Predicted Labels")
40. plt.ylabel("True Labels")
41. plt.show()
42. # Random Forest Classifier
43. num\_trees\_rf = 100
44. max\_features\_rf = 3
45. kfold\_rf = KFold(n\_splits=10, shuffle=True, random\_state=2020)
46. rf\_model = RandomForestClassifier(n\_estimators=num\_trees\_rf, max\_features= max\_features)
47. # Cross-validation
48. results\_rf = cross\_val\_score(rf\_model, X\_train, y\_train, cv=kfold\_rf)
49. print("Random Forest classifier Accuracy: %0.2f (+/- %0.2f)" % (results\_rf.mean(),results\_rf.std()))
50. #accuracy and F1 score
51. rf\_model.fit(X\_train, y\_train)
52. y\_pred\_rf = rf\_model.predict(X\_test)
53. accuracy\_rf = accuracy\_score(y\_test, y\_pred\_rf)
54. f1\_rf = f1\_score(y\_test, y\_pred\_rf)
55. print("Random Forest Classifier Accuracy: %0.2f" % accuracy\_rf)
56. print("Random Forest Classifier F1 Score: %0.2f" % f1\_rf)
57. # Random Forest Classifier
58. conf\_matrix\_rf = confusion\_matrix(y\_test, y\_pred\_rf)
59. # Plot confusion matrix for Random Forest Classifier
60. plt.figure(figsize=(4, 4))
61. sns.heatmap(conf\_matrix\_rf, annot=True, fmt="d", cmap="Blues", cbar=False,
62. xticklabels=['Class 0', 'Class 1'], yticklabels=['Class 0', 'Class 1'])
63. plt.title("Confusion Matrix - Random Forest Classifier")
64. plt.xlabel("Predicted Labels")
65. plt.ylabel("True Labels")
66. plt.show()



2Compare the results of Task - 1 classification tasks using accuracy, F1, and confusion matrix

from sklearn.model\_selection import KFold, cross\_val\_score

from sklearn.ensemble import BaggingClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn.datasets import make\_classification

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import f1\_score, accuracy\_score

from sklearn.metrics import confusion\_matrix

import seaborn as sns

import matplotlib.pyplot as plt

X, y = make\_classification(n\_samples=100, n\_features=20, n\_classes=2, random\_state=2020)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=2020)

max\_features = 3

kfold = KFold(n\_splits=10, shuffle=True, random\_state=2020)

decision\_tree = DecisionTreeClassifier(max\_features = max\_features)

num\_trees = 100

# Decision Tree base estimator

bagging\_model = BaggingClassifier(base\_estimator=decision\_tree, n\_estimators = num\_trees, random\_state=2020)

# Cross-validation

results = cross\_val\_score(bagging\_model, X\_train, y\_train, cv=kfold)

print("Bagging Classifier cross validation Accuracy: %0.2f (+/- %0.2f)" % (results.mean(), results.std()))

# Bagging Classifier with Decision Tree base estimator

bagging\_model.fit(X\_train, y\_train)

y\_pred\_bagging = bagging\_model.predict(X\_test)

accuracy\_bagging = accuracy\_score(y\_test, y\_pred\_bagging)

f1\_bagging = f1\_score(y\_test, y\_pred\_bagging)

print("Bagging Classifier Accuracy: %0.2f" % accuracy\_bagging)

print("Bagging Classifier F1 Score: %0.2f" % f1\_bagging)

# Bagging Classifier with Decision Tree base estimator

conf\_matrix\_bagging = confusion\_matrix(y\_test, y\_pred\_bagging)

# Plot confusion matrix for Bagging Classifier

plt.figure(figsize=(4, 4))

sns.heatmap(conf\_matrix\_bagging, annot=True, fmt="d", cmap="Blues", cbar=False,

            xticklabels=['Class 0', 'Class 1'], yticklabels=['Class 0', 'Class 1'])

plt.title("Confusion Matrix - Bagging Classifier")

plt.xlabel("Predicted Labels")

plt.ylabel("True Labels")

plt.show()

# Random Forest Classifier

num\_trees\_rf = 100

max\_features\_rf = 3

kfold\_rf = KFold(n\_splits=10, shuffle=True, random\_state=2020)

rf\_model = RandomForestClassifier(n\_estimators=num\_trees\_rf, max\_features= max\_features)

# Cross-validation

results\_rf = cross\_val\_score(rf\_model, X\_train, y\_train, cv=kfold\_rf)

print("Random Forest Accuracy: %0.2f (+/- %0.2f)" % (results\_rf.mean(),results\_rf.std()))

#accuracy and F1 score

rf\_model.fit(X\_train, y\_train)

y\_pred\_rf = rf\_model.predict(X\_test)

accuracy\_rf = accuracy\_score(y\_test, y\_pred\_rf)

f1\_rf = f1\_score(y\_test, y\_pred\_rf)

print("Random Forest Classifier Accuracy: %0.2f" % accuracy\_rf)

print("Random Forest Classifier F1 Score: %0.2f" % f1\_rf)

# Random Forest Classifier

conf\_matrix\_rf = confusion\_matrix(y\_test, y\_pred\_rf)

# Plot confusion matrix for Random Forest Classifier

plt.figure(figsize=(4, 4))

sns.heatmap(conf\_matrix\_rf, annot=True, fmt="d", cmap="Blues", cbar=False,

            xticklabels=['Class 0', 'Class 1'], yticklabels=['Class 0', 'Class 1'])

plt.title("Confusion Matrix - Random Forest Classifier")

plt.xlabel("Predicted Labels")

plt.ylabel("True Labels")

plt.show()

A screenshot of a computer

Description automatically generated

3Use Use *AdaBoostClassifier* and *RandomForstClassifier* from *sklearn* library and implment both on *Breast Cancer* data set.

from sklearn.ensemble import AdaBoostClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import f1\_score, accuracy\_score

from sklearn.datasets import load\_breast\_cancer

from sklearn.metrics import confusion\_matrix

from sklearn.ensemble import RandomForestClassifier

import seaborn as sns

import matplotlib.pyplot as plt

cancer = load\_breast\_cancer()

X = cancer.data

y = cancer.target

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=2020)

# Ada Boost classifier

clf\_boosting = AdaBoostClassifier( base\_estimator=DecisionTreeClassifier(max\_depth=1), n\_estimators=200

)

# Fit the model

clf\_boosting.fit(X\_train, y\_train)

# Make predictions

predictions = clf\_boosting.predict(X\_test)

# Calculate and print F1 Score and Accuracy

print("For Boosting: F1 Score {}, Accuracy {}".format(

    round(f1\_score(y\_test, predictions), 2),

    round(accuracy\_score(y\_test, predictions), 2)

))

# Confusion Matrix

conf\_mat = confusion\_matrix(y\_test, predictions)

# Plot Confusion Matrix

plt.figure(figsize=(8, 6))

sns.heatmap(conf\_mat, annot=True, fmt="d", cmap="Blues", xticklabels=["Benign", "Malignant"], yticklabels=["Benign", "Malignant"])

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix')

plt.show()

# Random forest classifier

# Model training using RandomForestClassifier as an example

rf\_model = RandomForestClassifier(n\_estimators=100, random\_state=42)

rf\_model.fit(X\_train, y\_train)

# Make predictions

rf\_predictions = rf\_model.predict(X\_test)

# Calculate and print F1 Score and Accuracy

print("For Random Forest: F1 Score {}, Accuracy {}".format(

    round(f1\_score(y\_test, predictions), 2),

    round(accuracy\_score(y\_test, predictions), 2)

))

# Confusion Matrix

conf\_mat = confusion\_matrix(y\_test, rf\_predictions)

# Plot Confusion Matrix

plt.figure(figsize=(8, 6))

sns.heatmap(conf\_mat, annot=True, fmt="d", cmap="Blues", xticklabels=["Benign", "Malignant"], yticklabels=["Benign", "Malignant"])

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix')

plt.show()

A screenshot of a computer

Description automatically generated

4Compare the results of Task - 3 classification tasks using accuracy, F1, and confusion matrix.

from sklearn.ensemble import AdaBoostClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import f1\_score, accuracy\_score

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import confusion\_matrix

from sklearn.ensemble import RandomForestClassifier

import seaborn as sns

import matplotlib.pyplot as plt

X, y = make\_classification(n\_samples=100, n\_features=20, n\_classes=2, random\_state=2020)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=2020)

clf\_boosting = AdaBoostClassifier( base\_estimator=DecisionTreeClassifier(max\_depth=1), n\_estimators=200

)

# Fit the model

clf\_boosting.fit(X\_train, y\_train)

# Make predictions

predictions = clf\_boosting.predict(X\_test)

# Calculate and print F1 Score and Accuracy

print("For Boosting: F1 Score {}, Accuracy {}".format(

    round(f1\_score(y\_test, predictions), 2),

    round(accuracy\_score(y\_test, predictions), 2)

))

# Confusion Matrix

conf\_mat = confusion\_matrix(y\_test, predictions)

# Plot Confusion Matrix

plt.figure(figsize=(8, 6))

sns.heatmap(conf\_mat, annot=True, fmt="d", cmap="Blues", xticklabels=["Benign", "Malignant"], yticklabels=["Benign", "Malignant"])

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix')

plt.show()

# Random forest classifier

# Model training using RandomForestClassifier as an example

rf\_model = RandomForestClassifier(n\_estimators=100, random\_state=42)

rf\_model.fit(X\_train, y\_train)

# Make predictions

rf\_predictions = rf\_model.predict(X\_test)

# Calculate and print F1 Score and Accuracy

print("For Random Forest: F1 Score {}, Accuracy {}".format(

    round(f1\_score(y\_test, predictions), 2),

    round(accuracy\_score(y\_test, predictions), 2)

))

# Confusion Matrix

conf\_mat = confusion\_matrix(y\_test, rf\_predictions)

# Plot Confusion Matrix

plt.figure(figsize=(8, 6))

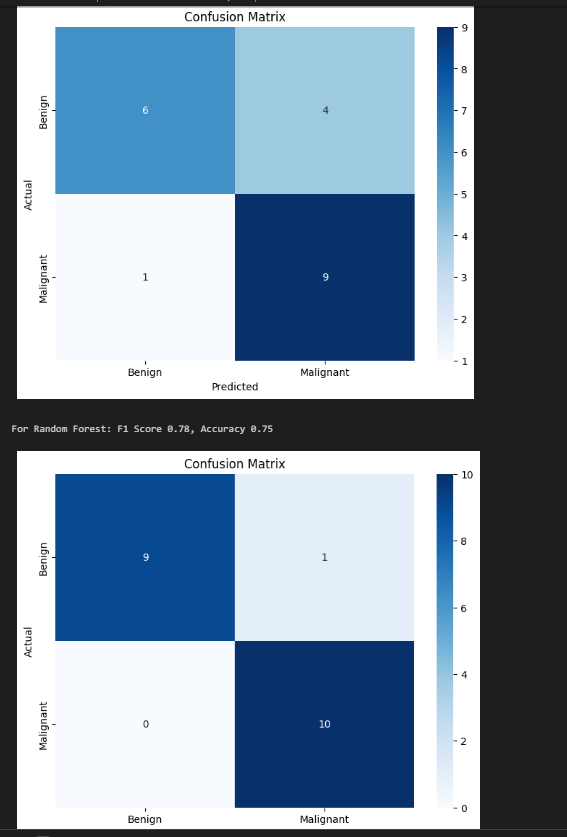
sns.heatmap(conf\_mat, annot=True, fmt="d", cmap="Blues", xticklabels=["Benign", "Malignant"], yticklabels=["Benign", "Malignant"])

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix')

plt.show()



5To check and compare the performance of *Bagging, Boosting, and Stacking* classification al- gorithms, implement *AdaBoostClassifier, RandomForestClassifier, and LogisticRegrassion* (for Stacking) on *Breast Cancer* data set.

import numpy as np

from sklearn.ensemble import AdaBoostClassifier, RandomForestClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import f1\_score, accuracy\_score

from sklearn.datasets import load\_breast\_cancer

from sklearn.metrics import confusion\_matrix

from sklearn.ensemble import RandomForestClassifier

import seaborn as sns

import matplotlib.pyplot as plt

class NumberOfClassifierException(Exception):

    pass

class Stacking():

    def \_\_init\_\_(self, classifiers):

        if len(classifiers) < 2:

            raise numberOfClassifierException (" You must fit your classifier with 2 classifiers at least ");

        else:

            self.\_classifiers = classifiers

    def fit(self, data\_x, data\_y):

        stacked\_data\_x = data\_x.copy()

        for classifier in self.\_classifiers[:-1]:

            classifier.fit(data\_x, data\_y)

            stacked\_data\_x = np.column\_stack((stacked\_data\_x ,classifier.predict\_proba(data\_x)))

        last\_classifier = self.\_classifiers[-1]

        last\_classifier.fit(stacked\_data\_x, data\_y)

    def predict(self, data\_x):

        stacked\_data\_x = data\_x.copy()

        for classifier in self.\_classifiers[:-1]:

            prob\_predictions = classifier.predict\_proba(data\_x)

            stacked\_data\_x = np.column\_stack ((stacked\_data\_x , prob\_predictions))

        last\_classifier = self.\_classifiers[-1]

        return last\_classifier.predict(stacked\_data\_x)

cancer = load\_breast\_cancer()

X = cancer.data

y = cancer.target

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=2020)

# Creating classifiers

boosting\_clf\_ada\_boost = AdaBoostClassifier( base\_estimator=DecisionTreeClassifier(max\_depth=1), n\_estimators=3)

clf\_rf = RandomForestClassifier( n\_estimators=200, max\_depth=1, random\_state=2020)

clf\_adaboost = AdaBoostClassifier( base\_estimator=DecisionTreeClassifier(max\_depth=1, random\_state=2020), n\_estimators=3)

clf\_logistic\_reg = LogisticRegression(solver='liblinear', random\_state=2020)

# Customizing and Exception message

classifiers\_list = [clf\_rf, clf\_adaboost, clf\_logistic\_reg]

clf\_stacking = Stacking(classifiers\_list)

# Fit models

clf\_rf.fit(X\_train, y\_train)

boosting\_clf\_ada\_boost.fit(X\_train, y\_train)

clf\_stacking.fit(X\_train, y\_train)

# Make predictions

predictions\_bagging = clf\_rf.predict(X\_test)

predictions\_boosting = boosting\_clf\_ada\_boost.predict(X\_test)

predictions\_stacking = clf\_stacking.predict(X\_test)

# Print results

print("For Bagging: F1 Score {}, Accuracy {}".format(

    round(f1\_score(y\_test, predictions\_bagging), 2),

    round(accuracy\_score(y\_test, predictions\_bagging), 2)

))

# Confusion Matrix for Bagging

conf\_mat\_bagging = confusion\_matrix(y\_test, predictions\_bagging)

# Plot Confusion Matrix for Bagging

plt.figure(figsize=(8, 6))

sns.heatmap(conf\_mat\_bagging, annot=True, fmt="d", cmap="Blues", xticklabels=["Benign", "Malignant"], yticklabels=["Benign", "Malignant"])

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix - Bagging Classifier')

plt.show()

print("For Boosting: F1 Score {}, Accuracy {}".format(

    round(f1\_score(y\_test, predictions\_boosting), 2),

    round(accuracy\_score(y\_test, predictions\_boosting), 2)

))

# Confusion Matrix for Bagging

conf\_mat\_bagging = confusion\_matrix(y\_test, predictions\_boosting)

# Plot Confusion Matrix for Bagging

plt.figure(figsize=(8, 6))

sns.heatmap(conf\_mat\_bagging, annot=True, fmt="d", cmap="Blues", xticklabels=["Benign", "Malignant"], yticklabels=["Benign", "Malignant"])

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix - Boosting Classifier')

plt.show()

print("For Stacking: F1 Score {}, Accuracy {}".format(

    round(f1\_score(y\_test, predictions\_stacking), 2),

    round(accuracy\_score(y\_test, predictions\_stacking), 2)

))

# Confusion Matrix for Bagging

conf\_mat\_bagging = confusion\_matrix(y\_test, predictions\_stacking)

# Plot Confusion Matrix for Bagging

plt.figure(figsize=(8, 6))

sns.heatmap(conf\_mat\_bagging, annot=True, fmt="d", cmap="Blues", xticklabels=["Benign", "Malignant"], yticklabels=["Benign", "Malignant"])

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix - Stacking Classifier')

plt.show()

A screenshot of a graph

Description automatically generated A screenshot of a computer screen

Description automatically generated

6Compare the results of Task - 5 classification tasks using accuracy, F1, and confusion matrix.

import numpy as np

from sklearn.ensemble import AdaBoostClassifier, RandomForestClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import f1\_score, accuracy\_score

from sklearn.datasets import load\_breast\_cancer

from sklearn.metrics import confusion\_matrix

from sklearn.ensemble import RandomForestClassifier

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.datasets import make\_classification

class NumberOfClassifierException(Exception):

    pass

class Stacking():

    def \_\_init\_\_(self, classifiers):

        if len(classifiers) < 2:

            raise numberOfClassifierException (" You must fit your classifier with 2 classifiers at least ");

        else:

            self.\_classifiers = classifiers

    def fit(self, data\_x, data\_y):

        stacked\_data\_x = data\_x.copy()

        for classifier in self.\_classifiers[:-1]:

            classifier.fit(data\_x, data\_y)

            stacked\_data\_x = np.column\_stack((stacked\_data\_x ,classifier.predict\_proba(data\_x)))

        last\_classifier = self.\_classifiers[-1]

        last\_classifier.fit(stacked\_data\_x, data\_y)

    def predict(self, data\_x):

        stacked\_data\_x = data\_x.copy()

        for classifier in self.\_classifiers[:-1]:

            prob\_predictions = classifier.predict\_proba(data\_x)

            stacked\_data\_x = np.column\_stack ((stacked\_data\_x , prob\_predictions))

        last\_classifier = self.\_classifiers[-1]

        return last\_classifier.predict(stacked\_data\_x)

X, y = make\_classification(n\_samples=1000, n\_features=20, n\_classes=2, random\_state=2020)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=2020)

# Creating classifiers

boosting\_clf\_ada\_boost = AdaBoostClassifier( base\_estimator=DecisionTreeClassifier(max\_depth=1), n\_estimators=3)

clf\_rf = RandomForestClassifier( n\_estimators=200, max\_depth=1, random\_state=2020)

clf\_adaboost = AdaBoostClassifier( base\_estimator=DecisionTreeClassifier(max\_depth=1, random\_state=2020), n\_estimators=3)

clf\_logistic\_reg = LogisticRegression(solver='liblinear', random\_state=2020)

# Customizing and Exception message

classifiers\_list = [clf\_rf, clf\_adaboost, clf\_logistic\_reg]

clf\_stacking = Stacking(classifiers\_list)

# Fit models

clf\_rf.fit(X\_train, y\_train)

boosting\_clf\_ada\_boost.fit(X\_train, y\_train)

clf\_stacking.fit(X\_train, y\_train)

# Make predictions

predictions\_bagging = clf\_rf.predict(X\_test)

predictions\_boosting = boosting\_clf\_ada\_boost.predict(X\_test)

predictions\_stacking = clf\_stacking.predict(X\_test)

# Print results

print("For Bagging: F1 Score {}, Accuracy {}".format(

    round(f1\_score(y\_test, predictions\_bagging), 2),

    round(accuracy\_score(y\_test, predictions\_bagging), 2)

))

# Confusion Matrix for Bagging

conf\_mat\_bagging = confusion\_matrix(y\_test, predictions\_bagging)

# Plot Confusion Matrix for Bagging

plt.figure(figsize=(8, 6))

sns.heatmap(conf\_mat\_bagging, annot=True, fmt="d", cmap="Blues", xticklabels=["Benign", "Malignant"], yticklabels=["Benign", "Malignant"])

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix - Bagging Classifier')

plt.show()

print("For Boosting: F1 Score {}, Accuracy {}".format(

    round(f1\_score(y\_test, predictions\_boosting), 2),

    round(accuracy\_score(y\_test, predictions\_boosting), 2)

))

# Confusion Matrix for Bagging

conf\_mat\_bagging = confusion\_matrix(y\_test, predictions\_boosting)

# Plot Confusion Matrix for Bagging

plt.figure(figsize=(8, 6))

sns.heatmap(conf\_mat\_bagging, annot=True, fmt="d", cmap="Blues", xticklabels=["Benign", "Malignant"], yticklabels=["Benign", "Malignant"])

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix - Boosting Classifier')

plt.show()

print("For Stacking: F1 Score {}, Accuracy {}".format(

    round(f1\_score(y\_test, predictions\_stacking), 2),

    round(accuracy\_score(y\_test, predictions\_stacking), 2)

))

# Confusion Matrix for Bagging

conf\_mat\_bagging = confusion\_matrix(y\_test, predictions\_stacking)

# Plot Confusion Matrix for Bagging

plt.figure(figsize=(8, 6))

sns.heatmap(conf\_mat\_bagging, annot=True, fmt="d", cmap="Blues", xticklabels=["Benign", "Malignant"], yticklabels=["Benign", "Malignant"])

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix - Stacking Classifier')

plt.show()

A screenshot of a computer

Description automatically generated A screenshot of a graph

Description automatically generated

Student Name: sawera fazal Roll No: 21A-026-seSection: 21A

CS334 - Machine Learning

Lab 10 - Probability Based Learning

Instructor: Ms. Maham Ashraf E-mail: mashraf@uit.edu

Semester: Fall, 2023

# Objective

The purpose of this lab session is to introduce probability based learning models, such as Naive-Bayes classifier for categorical and continues features, and Naive Bayes classifier with Laplace Smoothing.

# Instructions

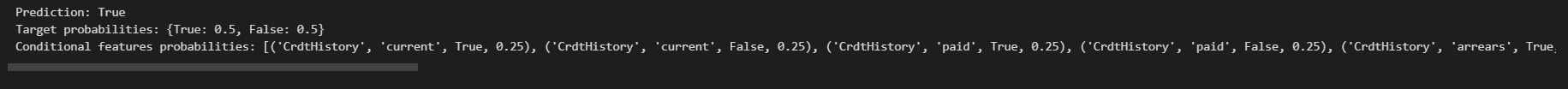
You have to perform the following tasks yourselves. Raise your hand if you face any difficulty in understanding and solving these tasks. **Plagiarism** is an abhorrent practice and you should not engage in it.

# How to Submit

* Submit lab work in a single .py file on Microsoft Teams. (No other format will be accepted)
* Lab work file name should be saved with your roll number (e.g. 19a-001-SE\_LW01.py)
* Submit home work in a single .py file on Microsoft Teams. (No other format will be accepted)
* Home work file name should be saved with your roll number (e.g. 19a-001-SE\_HW01.py)

### 3 Lab Tasks

1. Use the hints as provided in Section -2 for implementing Naive Bayes classifier for categorical variable and implement a class NBClassifier.
2. from sklearn.preprocessing import LabelEncoder
3. import pandas as pd
4. class NaiveBayesClassifier:
5. def \_\_init\_\_(self, data, target):
6. # Constructor to initialize the classifier with input data and target variable
7. self.data = data
8. self.target = target
9. self.target\_prob\_dict = {}  # Dictionary to store target class probabilities
10. self.cond\_prob\_list = []  # List to store conditional probabilities
11. self.smoothing = False  # Flag to enable/disable Laplace smoothing
12. self.k = 1  # Laplace Smoothing parameter
13. def compute\_target\_probabilities(self):
14. # Compute probabilities of each target class in the dataset
15. total\_rows = len(self.data)
16. target\_counts = self.data[self.target].value\_counts()
17. for level, count in target\_counts.items():
18. self.target\_prob\_dict[level] = count / total\_rows
19. def compute\_conditional\_probabilities(self):
20. # Compute conditional probabilities for each feature given the target class
21. for feature in self.data.columns:
22. if feature != self.target:
23. feature\_levels = self.data[feature].unique()
24. for level in feature\_levels:
25. for target\_level in self.target\_prob\_dict.keys():
26. count\_f\_v\_t = len(self.data[(self.data[feature] == level) & (self.data[self.target] == target\_level)])
27. count\_f\_t = len(self.data[self.data[self.target] == target\_level])
28. if self.smoothing:
29. # Laplace Smoothing
30. prob = (count\_f\_v\_t + self.k) / (count\_f\_t + self.k \* len(feature\_levels))
31. else:
32. prob = count\_f\_v\_t / count\_f\_t
33. # Store the computed probability in the list
34. self.cond\_prob\_list.append((feature, level, target\_level, round(prob, 3)))
35. def fit(self, smoothing=False, k=1):
36. # Method to train the Naive Bayes classifier
37. self.smoothing = smoothing
38. self.k = k
39. self.compute\_target\_probabilities()
40. self.compute\_conditional\_probabilities()
41. def predict\_instance(self, instance):
42. # Predict the target class for a given instance
43. prob\_dict = {}
44. for target\_level in self.target\_prob\_dict.keys():
45. prob\_prod = 1
46. for feature, level, t\_level, prob in self.cond\_prob\_list:
47. if feature in instance and instance[feature] == level and t\_level == target\_level:
48. prob\_prod \*= prob
49. # Multiply the conditional probability with the prior probability
50. prob\_dict[target\_level] = round(prob\_prod \* self.target\_prob\_dict[target\_level], 4)
51. # Return the predicted target class with the highest probability
52. return max(prob\_dict, key=prob\_dict.get)
53. def get\_probabilities(self):
54. # Display the computed target and conditional probabilities
55. print("Target probabilities:", self.target\_prob\_dict)
56. print("Conditional features probabilities:", self.cond\_prob\_list)
57. # Example usage:
58. data = {
59. 'CrdtHistory': ['current', 'paid', 'arrears', 'none', 'current', 'paid', 'arrears', 'none'],
60. 'GCoApplicant': ['none', 'guarantor', 'coapplicant', 'none', 'none', 'guarantor', 'coapplicant', 'none'],
61. 'Accommodation': ['own', 'rent', 'free', 'own', 'rent', 'free', 'own', 'rent'],
62. 'Target': [True, True, True, True, False, False, False, False]
63. }
64. df = pd.DataFrame(data)
65. nbc = NaiveBayesClassifier(df, 'Target')
66. nbc.fit(smoothing=True, k=1)
67. # Prediction for a new instance
68. query\_instance = {'CrdtHistory': 'current', 'GCoApplicant': 'none', 'Accommodation': 'own'}
69. prediction = nbc.predict\_instance(query\_instance)
70. print("Prediction:", prediction)
71. # Display all probabilities
72. nbc.get\_probabilities()



2Modify your NBClassifier that can work on continues descriptive and target variables.

import pandas as pd

import numpy as np

from scipy.stats import norm

from sklearn.preprocessing import LabelEncoder

class NaiveBayesClassifier:

    def \_\_init\_\_(self, data, target):

        # Constructor to initialize the classifier with input data and target variable

        self.data = data

        self.target = target

        self.target\_prob\_dict = {}  # Dictionary to store target class probabilities

        self.cond\_prob\_dict = {}  # Dictionary to store conditional probabilities

        self.continuous\_features = []  # List to store names of continuous features

        self.k = 1  # Laplace Smoothing parameter

    def compute\_target\_probabilities(self):

        # Compute probabilities of each target class in the dataset

        total\_rows = len(self.data)

        target\_counts = self.data[self.target].value\_counts()

        for level, count in target\_counts.items():

            self.target\_prob\_dict[level] = count / total\_rows

    def compute\_conditional\_probabilities(self):

        # Compute conditional probabilities for each feature given the target class

        for feature in self.data.columns:

            if feature != self.target:

                if self.data[feature].dtype == np.float64 or self.data[feature].dtype == np.int64:

                    # Continuous variable

                    self.continuous\_features.append(feature)

                else:

                    # Categorical variable

                    feature\_levels = self.data[feature].unique()

                    for level in feature\_levels:

                        for target\_level in self.target\_prob\_dict.keys():

                            count\_f\_v\_t = len(self.data[(self.data[feature] == level) & (self.data[self.target] == target\_level)])

                            count\_f\_t = len(self.data[self.data[self.target] == target\_level])

                            prob = count\_f\_v\_t / count\_f\_t

                            # Store the conditional probability in the dictionary

                            self.cond\_prob\_dict[(feature, level, target\_level)] = prob

    def fit(self, smoothing=False, k=1):

        # Method to train the Naive Bayes classifier

        self.k = k

        self.compute\_target\_probabilities()

        self.compute\_conditional\_probabilities()

    def predict\_instance(self, instance):

        # Predict the target class for a given instance

        prob\_dict = {}

        for target\_level in self.target\_prob\_dict.keys():

            prob\_prod = 1

            for feature, level, t\_level in self.cond\_prob\_dict.keys():

                if feature in instance:

                    if feature in self.continuous\_features:

                        # Use Gaussian distribution for continuous variables

                        mean = self.data[self.data[self.target] == t\_level][feature].mean()

                        std = self.data[self.data[self.target] == t\_level][feature].std()

                        prob\_density = norm.pdf(instance[feature], mean, std)

                        prob\_prod \*= prob\_density

                    else:

                        # Categorical variable

                        if instance[feature] == level and t\_level == target\_level:

                            prob\_prod \*= self.cond\_prob\_dict[(feature, level, target\_level)]

            # Multiply the conditional probability with the prior probability

            prob\_dict[target\_level] = round(prob\_prod \* self.target\_prob\_dict[target\_level], 4)

        # Return the predicted target class with the highest probability

        return max(prob\_dict, key=prob\_dict.get)

    def get\_probabilities(self):

        # Display the computed target and conditional probabilities

        print("Target probabilities:", self.target\_prob\_dict)

        print("Conditional features probabilities:", self.cond\_prob\_dict)

# Read your dataset

data = pd.read\_csv('loan.csv')

# Initialize a LabelEncoder for encoding categorical variables

label\_encoder = LabelEncoder()

# Convert 'Education', 'Self\_Employed', and 'Loan\_Status' columns to numerical labels

data['Education'] = label\_encoder.fit\_transform(data['Education'])

data['Self\_Employed'] = label\_encoder.fit\_transform(data['Self\_Employed'])

data['Loan\_Status'] = label\_encoder.fit\_transform(data['Loan\_Status'])

# Assuming 'Credit\_History' is your target variable

# Create an instance of the NaiveBayesClassifier class with the preprocessed data and target variable

nbc = NaiveBayesClassifier(data, 'Loan\_Status')

# Fit the Naive Bayes classifier to the data with Laplace smoothing (smoothing=True) and Laplace Smoothing parameter k=1

nbc.fit(smoothing=True, k=1)

# Prediction for a new instance

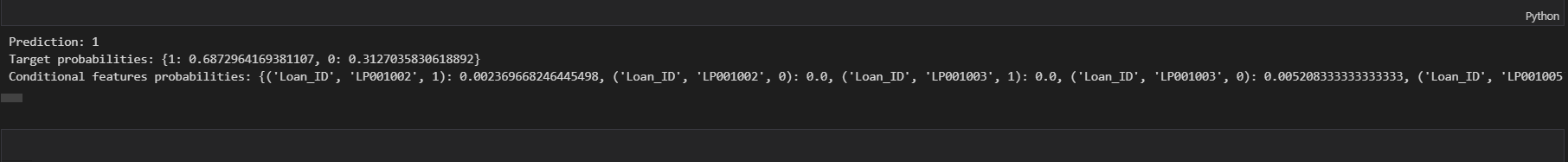
query\_instance = {'Education': 0, 'Self\_Employed': 0, 'Credit\_History': 1, 'ApplicantIncome': 5000, 'LoanAmount': 128}

prediction = nbc.predict\_instance(query\_instance)

print("Prediction:", prediction)

# Display all target and conditional probabilities computed by the classifier

nbc.get\_probabilities()



Student Name: sawera fazal Roll No: 21A-026-seSection: 21A

# CS334 - Machine Learning

Lab 11 - Error Based Learning

Instructor: Ms. Maham Ashraf E-mail: mashraf@uit.edu Semester: Fall, 2023

## Objective

The purpose of this lab session is to introduce Error based learning models, such as Gradient Descent algorithm, Regression based classifier for categorical and continues features, and Support Vector Machine (SVM).

## Instructions

You have to perform the following tasks yourselves. Raise your hand if you face any difficulty in understanding and solving these tasks. **Plagiarism** is an abhorrent practice and you should not engage in it.

## How to Submit

* Submit lab work in a single .py file on Microsoft Teams. (No other format will be accepted)
* Lab work file name should be saved with your roll number (e.g. 19a-001-SE\_LW01.py)
* Submit home work in a single .py file on Microsoft Teams. (No other format will be accepted)
* Home work file name should be saved with your roll number (e.g. 19a-001-SE\_HW01.py)
* **Lab Tasks**

1. Write a complete Linear Regression class as defined in Section 2.

* # Importing necessary libraries
* import pandas as pd
* import numpy as np
* # Loading the training dataset from 'train.csv' into a Pandas DataFrame
* df\_train = pd.read\_csv('train.csv')
* # Loading the testing dataset from 'test.csv' into a Pandas DataFrame
* df\_test = pd.read\_csv('test.csv')
* # Extracting the 'x' column (features) from the training dataset
* x\_train = df\_train['x']
* # Extracting the 'y' column (labels) from the training dataset
* y\_train = df\_train['y']
* # Extracting the 'x' column (features) from the testing dataset
* x\_test = df\_test['x']
* # Extracting the 'y' column (labels) from the testing dataset
* y\_test = df\_test['y']
* # Converting the 'x\_train' and 'y\_train' data to NumPy arrays
* x\_train = np.array(x\_train)
* y\_train = np.array(y\_train)
* # Converting the 'x\_test' and 'y\_test' data to NumPy arrays
* x\_test = np.array(x\_test)
* y\_test = np.array(y\_test)
* # Reshaping the 'x\_train' array to have a single feature column (-1 indicates that the size of that dimension is inferred)
* x\_train = x\_train.reshape(-1, 1)
* # Reshaping the 'x\_test' array to have a single feature column (-1 indicates that the size of that dimension is inferred)
* x\_test = x\_test.reshape(-1, 1)

A screenshot of a computer program

Description automatically generated A screenshot of a computer program

Description automatically generated A screen shot of a computer program

Description automatically generated A screen shot of a computer program

Description automatically generated

A screen shot of a graph

Description automatically generated

Student Name: Sawera Fazal Roll No: 21A-026-se2Section: 21A

# CS334 - Machine Learning

Lab 12 - Error Based Learning - Part - 2

Instructor: Ms. Maham Ashraf E-mail: mashraf@uit.edu

Semester: Fall, 2023

## Objective

The purpose of this lab session is to introduce Error based learning models, such as Gradient Descent algorithm, Regression based classifier for categorical and continues features, and Support Vector Machine (SVM).

## Instructions

You have to perform the following tasks yourselves. Raise your hand if you face any difficulty in understanding and solving these tasks. **Plagiarism** is an abhorrent practice and you should not engage in it.

## How to Submit

* Submit lab work in a single .py file on Microsoft Teams. (No other format will be accepted)
* Lab work file name should be saved with your roll number (e.g. 19a-001-SE\_LW01.py)
* Submit home work in a single .py file on Microsoft Teams. (No other format will be accepted)
* Home work file name should be saved with your roll number (e.g. 19a-001-SE\_HW01.py)

### 5 Lab Tasks

1. Use SGD regressor on datasets. Use food.csv data set and implement SGD regressor, print the learning graph and accuracy of the model.



import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import SGDRegressor

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import mean\_squared\_error

from sklearn.preprocessing import LabelEncoder

from sklearn.metrics import r2\_score

# Load the dataset (assuming it's in the same directory as the script)

data = pd.read\_csv("food.csv")

# Assume that 'target\_column' is the column you want to predict

target\_column = "score"

X = data.drop(target\_column, axis=1)

y = data[target\_column]

# Label encode 'title' and 'id' columns

le = LabelEncoder()

X['title'] = le.fit\_transform(X['title'])

X['id'] = le.fit\_transform(X['id'])

X['url'] = le.fit\_transform(X['url'])

X['body'] = le.fit\_transform(X['body'])

X['timestamp'] = le.fit\_transform(X['timestamp'])

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Feature scaling (important for SGD)

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

# Create an SGDRegressor instance

regressor = SGDRegressor(max\_iter=1000, tol=1e-3, random\_state=42)

# Manually capture the loss values during training

loss\_values = []

for epoch in range(1000):  # Adjust the number of epochs as needed

    regressor.partial\_fit(X\_train\_scaled, y\_train)

    y\_pred = regressor.predict(X\_train\_scaled)

    mse = mean\_squared\_error(y\_train, y\_pred)

    loss\_values.append(mse)

# Predict on the test set

y\_pred = regressor.predict(X\_test\_scaled)

# Calculate and print the Mean Squared Error (MSE) as a measure of accuracy

mse = mean\_squared\_error(y\_test, y\_pred)

print("Mean Squared Error:", mse)

# Calculate R-squared score

score = r2\_score(y\_test, y\_pred)

print("R-squared Score:", score)

# Plot the learning graph

plt.plot(loss\_values)

plt.title("SGD Regressor Learning Curve")

plt.xlabel("Number of Iterations")

plt.ylabel("Loss")

plt.show()

A screen shot of a graph

Description automatically generated

1. To learn how to handle descriptive categorical variable in Linear Regression, develop the model as described below,
   1. Load flight.csv data set.
   2. Apply pre-processing on data set.
   3. Encode categorical features into numerical values.
   4. Apply linear regression model from sklearn library.
   5. Predict the delay of flight departure.

A screenshot of a computer program

Description automatically generated import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn.preprocessing import OneHotEncoder

from sklearn.compose import ColumnTransformer

from sklearn.pipeline import Pipeline

from sklearn.metrics import mean\_squared\_error

# (a) Load flight.csv data set

data = pd.read\_csv("flight.csv")

# (b) Apply pre-processing on the data set

# Assuming 'DepDelay' is the target column to predict

target\_column = "DepDelay"

X = data.drop(target\_column, axis=1)

y = data[target\_column]

# Assume 'UniqueCarrier' and 'Origin' are categorical features

categorical\_features = ['UniqueCarrier', 'Origin']

X\_categorical = X[categorical\_features]

X\_numerical = X.drop(categorical\_features, axis=1)

# (c) Encode categorical features into numerical values

# Use OneHotEncoder for encoding categorical variables

preprocessor = ColumnTransformer(

    transformers=[

        ('cat', OneHotEncoder(), categorical\_features)

    ],

    remainder='passthrough'

)

# (d) Apply linear regression model from sklearn library

# Create a pipeline with preprocessing and linear regression

model = Pipeline([

    ('preprocessor', preprocessor),

    ('regressor', LinearRegression())

])

# (e) Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Fit the model on the training data

model.fit(X\_train, y\_train)

# Predict the delay of flight departure on the test set

y\_pred = model.predict(X\_test)

# Evaluate the model

mse = mean\_squared\_error(y\_test, y\_pred)

print("Mean Squared Error:", mse)

A screenshot of a computer program

Description automatically generated A diagram with blue dots

Description automatically generated

1. To learn how to handle continues target variable in Linear Regression Model, Read the following example and develop the proper model.

The ALCustomers.csv data contains data of airline customer. The main purpose of this dataset is to predict whether a future customer would be satisfied with their service given the details of the other parameters values.

Also the airlines need to know on which aspect of the services offered by them have to be emphasized more to generate more satisfied customers.

* 1. Develop a prediction model that predict that either customer would be satisfy with the current services or not.
  2. Predict the factors (e.g. descriptive variables) that need improvements to increase customer satisfaction rate.

1. import pandas as pd
2. from sklearn.model\_selection import train\_test\_split
3. from sklearn.linear\_model import LogisticRegression
4. from sklearn.impute import SimpleImputer
5. from sklearn.preprocessing import StandardScaler, OneHotEncoder
6. from sklearn.compose import ColumnTransformer
7. from sklearn.pipeline import Pipeline
8. from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix
9. # Load the dataset
10. data = pd.read\_csv("Alcustomer.csv")
11. # Assume 'satisfaction' is the target variable
12. target\_column = "satisfaction"
13. X = data.drop(target\_column, axis=1)
14. y = data[target\_column]
15. # Split the data into training and testing sets
16. X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)
17. # Define numerical and categorical features
18. numeric\_features = X.select\_dtypes(include=['int64', 'float64']).columns
19. categorical\_features = X.select\_dtypes(include=['object']).columns
20. # Create a preprocessor using ColumnTransformer
21. preprocessor = ColumnTransformer(
22. transformers=[
23. ('num', StandardScaler(), numeric\_features),
24. ('cat', OneHotEncoder(), categorical\_features)
25. ],
26. remainder='passthrough'
27. )
28. # Create a Logistic Regression model pipeline with imputation
29. model = Pipeline([
30. ('preprocessor', preprocessor),
31. ('imputer', SimpleImputer(strategy='mean')),  # Choose imputation strategy
32. ('classifier', LogisticRegression(random\_state=42))
33. ])
34. # Fit the model on the training data
35. model.fit(X\_train, y\_train)
36. # Predict on the test set
37. y\_pred = model.predict(X\_test)
38. # Evaluate the model
39. accuracy = accuracy\_score(y\_test, y\_pred)
40. conf\_matrix = confusion\_matrix(y\_test, y\_pred)
41. classification\_rep = classification\_report(y\_test, y\_pred)
42. print("Accuracy:", accuracy)
43. print("\nConfusion Matrix:")
44. print(conf\_matrix)
45. print("\nClassification Report:")
46. print(classification\_rep)
47. # Get feature names after one-hot encoding (if applicable)
48. feature\_names = X.columns.tolist()

A screenshot of a computer program

Description automatically generated

Student Name: Sawera Fazal Roll No: 21A-026-seSection: 21A

# CS334 - Machine Learning

Lab 13

Instructor: Ms. Maham Ashraf E-mail: mashraf@uit.edu Semester: Fall, 2023

# Objective

Open Ended Lab to perform all the concepts of previous lab and work on a dataset to implement Analysis, process, feature Selection , Model Implementation and model Evaluation with Visualization

# Instructions

You have to perform the following tasks yourselves. Raise your hand if you face any difficulty in understanding and solving these tasks. **Plagiarism** is an abhorrent practice and you should not engage in it.

# How to Submit

* Submit lab work in a single .py file on Microsoft Teams. (No other format will be accepted)
* Lab work file name should be saved with your roll number (e.g. 19a-001SE\_LW01.py)
* Submit home work in a single .py file on Microsoft Teams. (No other format will be accepted)
* Home work file name should be saved with your roll number (e.g. 19a-001SE\_HW01.py)

Lab task:

1)Explore Hyper parameter tuning on any dataset from Kaggle.

2)Explore other hyper parameter tuning techniques and use them in your code on your own dataset.

3)Evaluate the performance of different hyperparameter configurations on the validation set.

A screenshot of a computer

Description automatically generated

A screen shot of a computer program

Description automatically generated

A black and white sign with white text

Description automatically generated from sklearn.model\_selection import GridSearchCV, RandomizedSearchCV

param\_grid = {'criterion': ['gini', 'entropy'],

              'splitter': ['best', 'random'],

              'max\_depth': [None, 10, 20, 30, 40, 50],

              'min\_samples\_split': [2, 5, 10],

              'min\_samples\_leaf': [1, 2, 4]}

param\_dist = {'criterion': ['gini', 'entropy'],

              'splitter': ['best', 'random'],

              'max\_depth': [None, 10, 20, 30, 40, 50],

              'min\_samples\_split': [2, 5, 10],

              'min\_samples\_leaf': [1, 2, 4]}

# Grid Search

grid\_search = GridSearchCV(model, param\_grid, cv=5)

grid\_search.fit(X\_train, y\_train)

# Random Search

random\_search = RandomizedSearchCV(model, param\_distributions=param\_dist, n\_iter=10, cv=5)

random\_search.fit(X\_train, y\_train)

# Evaluate models on the testing set

y\_pred\_grid = grid\_search.predict(X\_test)

y\_pred\_random = random\_search.predict(X\_test)

# Calculate accuracy scores

accuracy\_grid = accuracy\_score(y\_test, y\_pred\_grid)

accuracy\_random = accuracy\_score(y\_test, y\_pred\_random)

# Print accuracy scores

print("Accuracy with Grid Search:", accuracy\_grid)

print("Accuracy with Random Search:", accuracy\_random)

A screenshot of a computer

Description automatically generated

Student Name: Sawera fazal Roll No: 21A-026-seSection: 21A

# CS334 - Machine Learning

Lab 14

Instructor: Ms. Maham Ashraf E-mail: mashraf@uit.edu Semester: Fall, 2023

# Objective

Open Ended Lab to perform all the concepts of previous lab and work on a dataset to implement Analysis, process, feature Selection , Model Implementation and model Evaluation with Visualization

# Instructions

You have to perform the following tasks yourselves. Raise your hand if you face any difficulty in understanding and solving these tasks. **Plagiarism** is an abhorrent practice and you should not engage in it.

# How to Submit

* Submit lab work in a single .py file on Microsoft Teams. (No other format will be accepted)
* Lab work file name should be saved with your roll number (e.g. 19a-001SE\_LW01.py)
* Submit home work in a single .py file on Microsoft Teams. (No other format will be accepted)
* Home work file name should be saved with your roll number (e.g. 19a-001SE\_HW01.py)

**Objective:**

This open-ended lab project aims to consolidate and apply the diverse range of AI concepts learned in previous labs. The project encourages students to work on a real-world dataset, performing tasks such as analysis, data processing, feature selection, model implementation, and evaluation. Visualization techniques should be employed to enhance the understanding and interpretation of the results.

Project Tasks:

1. **Data Exploration and Understanding:** 
   * + Begin by thoroughly exploring the contents of the provided dataset.
     + Examine the structure, data types, and key attributes within the dataset.
     + Document any initial observations or patterns identified during exploration.
2. **Analysis and Visualization:** 
   * + Conduct a comprehensive analysis of the dataset, exploring statistical measures, and correlations.
     + Utilize visualization libraries to create insightful charts, graphs, or plots.
     + Provide visualizations that enhance understanding and reveal patterns within the data.
3. **Data Preprocessing and Cleaning:** 
   * + Implement necessary preprocessing steps to clean and organize the dataset.
     + Handle missing values, outliers, or any other data imperfections.
     + Discuss the impact of preprocessing on the overall quality of the data.

1. **Feature Selection and Engineering:** 
   * + Explore feature selection techniques to identify and retain the most relevant features.
     + Consider the creation of new features through feature engineering.
     + Discuss the rationale behind feature selection and engineering decisions.
2. **Model Implementation:** 
   * + Choose a suitable machine learning or deep learning model for the task at hand.
     + Implement the selected model using appropriate libraries or frameworks.
     + Justify the choice of the model architecture based on the characteristics of the dataset.
3. **Training and Evaluation:** 
   * + Train the implemented model using the preprocessed dataset.
     + Evaluate the model's performance using relevant metrics.
     + Discuss the significance of the evaluation results in the context of the project.
4. **Results Visualization and Analysis:** 
   * Visualize the predictions of the model on sample data.
   * Analyze the results, emphasizing the strengths and limitations of the implemented model.
   * Consider the interpretability of the model and its implications.

Submission Guidelines:

* + Submit in a well-documented Jupyter notebook containing the implemented code, explanations, and visualizations.
  + Include comments and markdown cells for clarity and understanding.
  + Ensure the file provides a clear narrative of the project, from data exploration to model evaluation.

**Sawera Fazal 21A-026-SE Shaheer Zaman 21A-003-SE**



**EDA AND VISUALISATION**

In [52]:

**import pandas as pd**

data = pd.read\_csv("loan\_approval\_dataset.csv") dataFrame = pd.DataFrame(data)

dataFrame.head()

Out[52]:



**loan\_id no\_of\_dependents education self\_employed income\_annum loan\_amount loan\_term cibil\_score residential\_assets\_value**

**0** 1 2 Graduate No 9600000 29900000 12 778 2400000

**1**

2

0

Not Graduate

Yes

4100000 12200000

8

417

2700000

**2**

3

3 Graduate

No

9100000 29700000

20

506

7100000

**3** 4 3 Graduate No 8200000 30700000 8 467 18200000

**4**

5

5

Not Graduate

Yes

9800000 24200000

20

382

12400000

In [5]:

dataFrame.shape

Out[5]:

(4269, 13)

In [6]:

dataFrame.describe()

Out[6]:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |
| **loan\_id** | **no\_of\_dependents** | **income\_annum** | **loan\_amount** | **loan\_term** | **cibil\_score** | **residential\_assets\_value** | **commercial\_a** |  |
| **count** | 4269.000000 | 4269.000000 | 4.269000e+03 | 4.269000e+03 | 4269.000000 | 4269.000000 | 4.269000e+03 | 4. |  |
| **mean** | 2135.000000 | 2.498712 | 5.059124e+06 | 1.513345e+07 | 10.900445 | 599.936051 | 7.472617e+06 | 4. |  |
| **std** | 1232.498479 | 1.695910 | 2.806840e+06 | 9.043363e+06 | 5.709187 | 172.430401 | 6.503637e+06 | 4. |  |
| **min** | 1.000000 | 0.000000 | 2.000000e+05 | 3.000000e+05 | 2.000000 | 300.000000 | -1.000000e+05 | 0. |  |
| **25%** | 1068.000000 | 1.000000 | 2.700000e+06 | 7.700000e+06 | 6.000000 | 453.000000 | 2.200000e+06 | 1. |  |
| **50%** | 2135.000000 | 3.000000 | 5.100000e+06 | 1.450000e+07 | 10.000000 | 600.000000 | 5.600000e+06 | 3. |  |
| **75%** | 3202.000000 | 4.000000 | 7.500000e+06 | 2.150000e+07 | 16.000000 | 748.000000 | 1.130000e+07 | 7. |  |
| **max** | 4269.000000 | 5.000000 | 9.900000e+06 | 3.950000e+07 | 20.000000 | 900.000000 | 2.910000e+07 | 1 |  |



dataFrame.dtypes

Out[7]:

loan\_id int64

no\_of\_dependents int64

education object

self\_employed object

income\_annum int64

loan\_amount int64

loan\_term int64

cibil\_score int64

residential\_assets\_value int64 commercial\_assets\_value int64 luxury\_assets\_value int64

bank\_asset\_value int64

loan\_status object dtype: object

In [8]:

null = dataFrame.isnull().sum() print(null.count)

|  |  |  |
| --- | --- | --- |
| <bound method Series.count of | loan\_id | 0 |
| no\_of\_dependents | 0 |  |
| education | 0 |  |
| self\_employed | 0 |  |
| income\_annum | 0 |  |
| loan\_amount | 0 |  |
| loan\_term | 0 |  |
| cibil\_score | 0 |  |
| residential\_assets\_value | 0 |  |
| commercial\_assets\_value | 0 |  |
| luxury\_assets\_value | 0 |  |
| bank\_asset\_value | 0 |  |
| loan\_status | 0 |  |
| dtype: int64> |  |  |

In [9]:

dataFrame.columns

Out[9]:

Index(['loan\_id', ' no\_of\_dependents', ' education', ' self\_employed', ' income\_annum', ' loan\_amount', ' loan\_term', ' cibil\_score', ' residential\_assets\_value', ' commercial\_assets\_value',

' luxury\_assets\_value', ' bank\_asset\_value', ' loan\_status'], dtype='object')

In [10]:

*#checking relation btw columns*

dataFrame.corr()

Out[10]:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |
| **loan\_id** | **no\_of\_dependents** | **income\_annum** | **loan\_amount** | **loan\_term** | **cibil\_score** | **residential\_assets\_value** |  |
| **loan\_id** | 1.000000 | 0.005326 | 0.012592 | 0.008170 | 0.009809 | 0.016323 | 0.020936 |  |
| **no\_of\_dependents** | 0.005326 | 1.000000 | 0.007266 | -0.003366 | -0.020111 | -0.009998 | 0.007376 |  |
| **income\_annum** | 0.012592 | 0.007266 | 1.000000 | 0.927470 | 0.011488 | -0.023034 | 0.636841 |  |
| **loan\_amount** | 0.008170 | -0.003366 | 0.927470 | 1.000000 | 0.008437 | -0.017035 | 0.594596 |  |
| **loan\_term** | 0.009809 | -0.020111 | 0.011488 | 0.008437 | 1.000000 | 0.007810 | 0.008016 |  |
| **cibil\_score** | 0.016323 | -0.009998 | -0.023034 | -0.017035 | 0.007810 | 1.000000 | -0.019947 |  |
| **residential\_assets\_value** | 0.020936 | 0.007376 | 0.636841 | 0.594596 | 0.008016 | -0.019947 | 1.000000 |  |
| **commercial\_assets\_value** | 0.018595 | -0.001531 | 0.640328 | 0.603188 | -0.005478 | -0.003769 | 0.414786 |  |
| **luxury\_assets\_value** | -0.000862 | 0.002817 | 0.929145 | 0.860914 | 0.012490 | -0.028618 | 0.590932 |  |
| **bank\_asset\_value** | 0.010765 | 0.011163 | 0.851093 | 0.788122 | 0.017177 | -0.015478 | 0.527418 |  |
|  |  |  |  |  |  |  |  |  |

dataFrame.info()

<class 'pandas.core.frame.DataFrame'> RangeIndex: 4269 entries, 0 to 4268 Data columns (total 13 columns):

# Column Non-Null Count Dtype

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 loan\_id | 4269 | non-null |  | int64 |
| 1 no\_of\_dependents | 4269 | non-null |  | int64 |
| 2 education | 4269 | non-null |  | object |
| 3 self\_employed | 4269 | non-null |  | object |
| 4 income\_annum | 4269 | non-null |  | int64 |
| 5 loan\_amount | 4269 | non-null |  | int64 |
| 6 loan\_term | 4269 | non-null |  | int64 |
| 7 cibil\_score | 4269 | non-null |  | int64 |
| 8 residential\_assets\_value | 4269 | non-null |  | int64 |
| 9 commercial\_assets\_value | 4269 | non-null |  | int64 |
| 10 luxury\_assets\_value | 4269 | non-null |  | int64 |
| 11 bank\_asset\_value | 4269 | non-null |  | int64 |
| 12 loan\_status | 4269 | non-null |  | object |

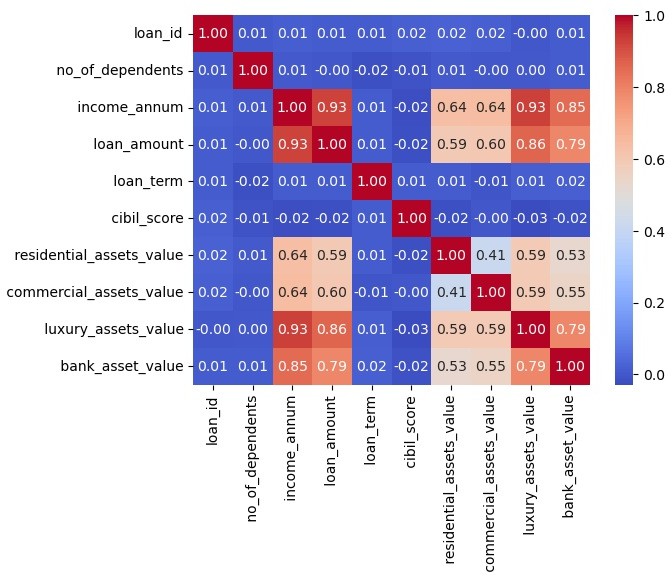
dtypes: int64(10), object(3) memory usage: 433.7+ KB

In [12]:

**import matplotlib.pyplot as plt import seaborn as sns**

**import numpy as np**

numerical\_columns = dataFrame.select\_dtypes(include=np.number).columns dataFrame[numerical\_columns].hist(bins=20, figsize=(15, 10)) plt.show()



correlation\_matrix = dataFrame.corr()

sns.heatmap(correlation\_matrix, annot=**True**, cmap='coolwarm', fmt=".2f") plt.show()

*# Bar plot between 'education', 'self\_employed', and 'loan\_status'*

plt.figure(figsize=(8, 6))

*# Bar plot for 'education'*

plt.subplot(3, 1, 1)

sns.countplot(x=' education', data=dataFrame) plt.title('Bar Plot: Education Distribution')

*# Bar plot for 'self\_employed'*

plt.subplot(3, 1, 2)

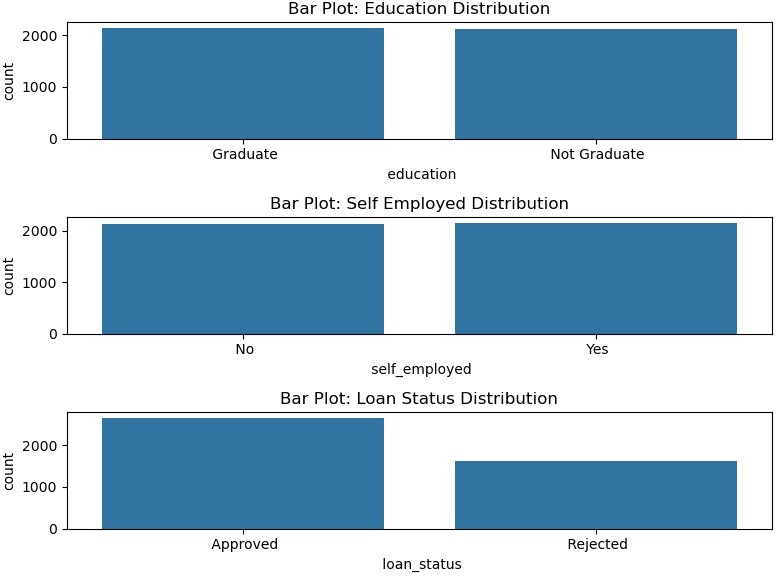
sns.countplot(x=' self\_employed', data=dataFrame) plt.title('Bar Plot: Self Employed Distribution')

*# Bar plot for 'loan\_status'*

plt.subplot(3, 1, 3)

sns.countplot(x=' loan\_status', data=dataFrame) plt.title('Bar Plot: Loan Status Distribution')

plt.tight\_layout() plt.show()

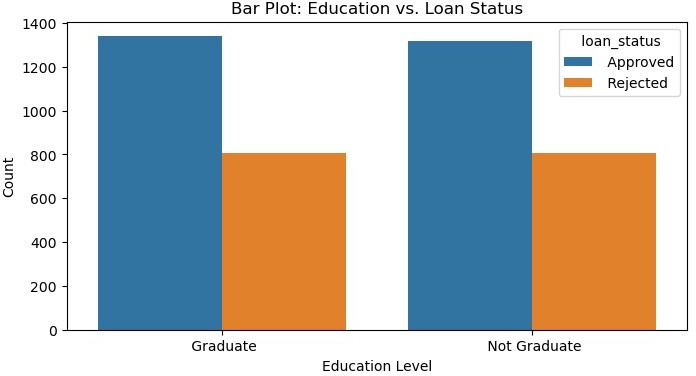


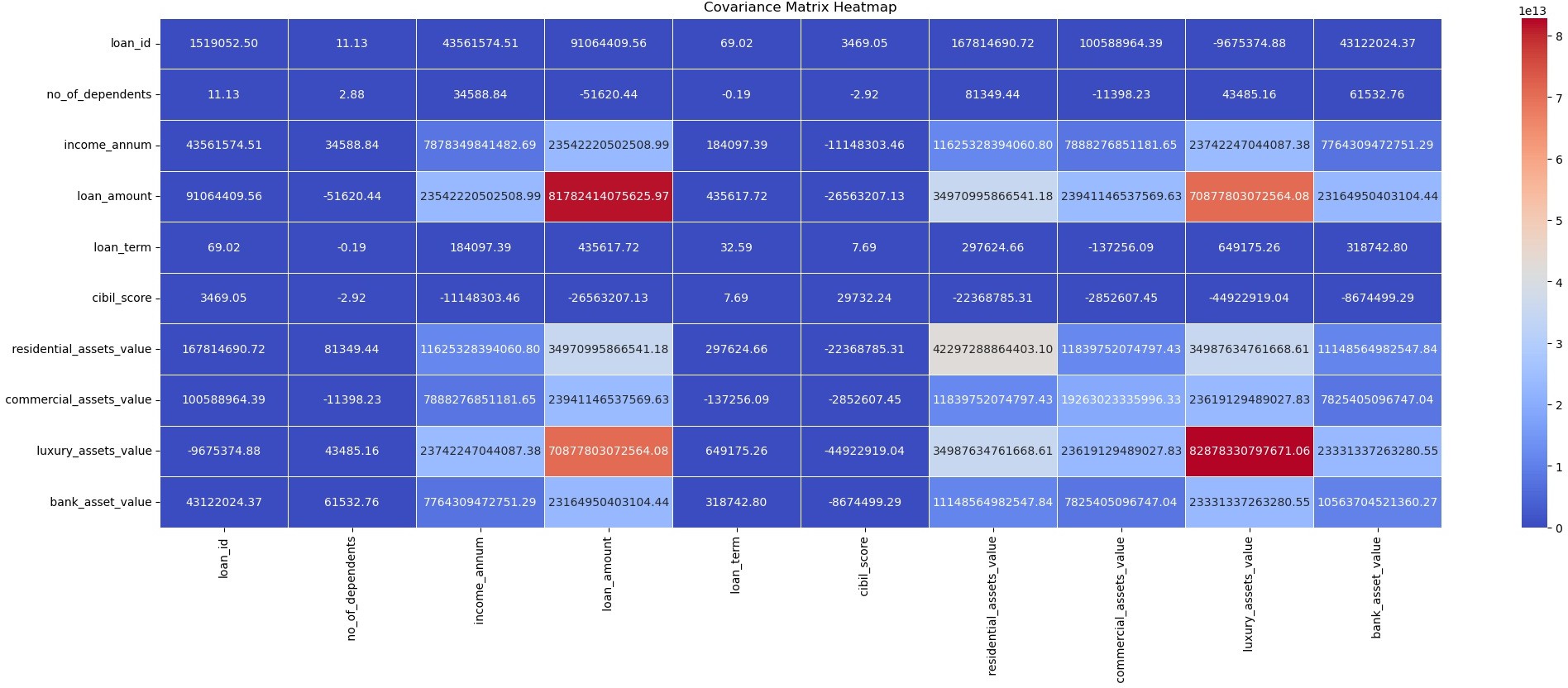
*# Bar plot between 'education' and 'loan\_status'*

plt.figure(figsize=(8, 4))

sns.countplot(x=' education', hue=' loan\_status', data=dataFrame) plt.title('Bar Plot: Education vs. Loan Status') plt.xlabel('Education Level')

plt.ylabel('Count') plt.show()



In [17]:

*# Calculate the covariance matrix*

covariance\_matrix = dataFrame[numerical\_columns].cov()

*# Plot the covariance matrix as a heatmap*

plt.figure(figsize=(25, 8))

sns.heatmap(covariance\_matrix, annot=**True**, cmap='coolwarm', fmt='.2f', linewidths=0.5) plt.title('Covariance Matrix Heatmap')

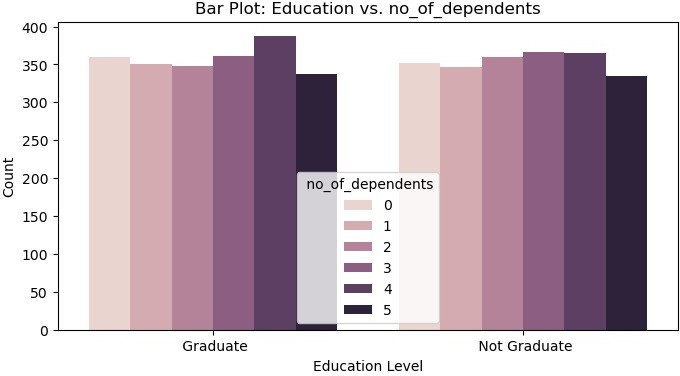
plt.show()

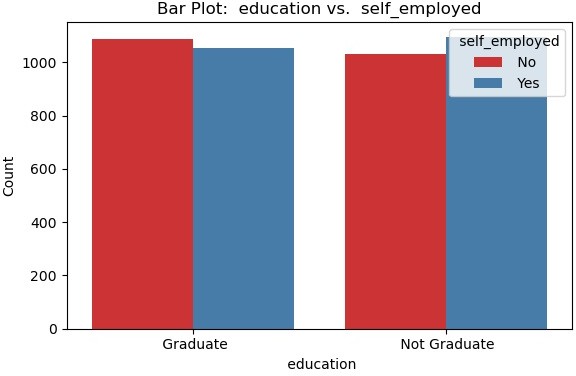
*# Bar plot between 'education' and 'loan\_status'*

plt.figure(figsize=(8, 4))

sns.countplot(x=' education', hue=' no\_of\_dependents', data=dataFrame) plt.title('Bar Plot: Education vs. no\_of\_dependents') plt.xlabel('Education Level')

plt.ylabel('Count') plt.show()



In [58]:

*# Select three categorical columns for the bar plot*

categorical\_columns = [' education', ' self\_employed', ' no\_of\_dependents']

*# Create a bar plot*

plt.figure(figsize=(6, 4))

sns.countplot(x=categorical\_columns[0], hue=categorical\_columns[1], data=dataFrame, palette='Set1') plt.title(f'Bar Plot: **{categorical\_columns[0]}** vs. **{categorical\_columns[1]}**') plt.xlabel(categorical\_columns[0])

plt.ylabel('Count') plt.legend(title=categorical\_columns[1])

plt.tight\_layout() plt.show()

*# List of variables to analyze*

variables\_to\_analyze = [' no\_of\_dependents', ' education', ' self\_employed']

*# Set up subplots*

fig, axes = plt.subplots(nrows=1, ncols=len(variables\_to\_analyze), figsize=(15, 5))

*# Loop through variables and create bar plots*

**for** i, variable **in** enumerate(variables\_to\_analyze): ax = axes[i]

df\_grouped = dataFrame.groupby([variable, ' loan\_status']).size().unstack() df\_grouped.plot(kind='bar', stacked=**True**, ax=ax) ax.set\_title(f'Relationship between **{variable}** and Loan Status') ax.set\_xlabel(variable)

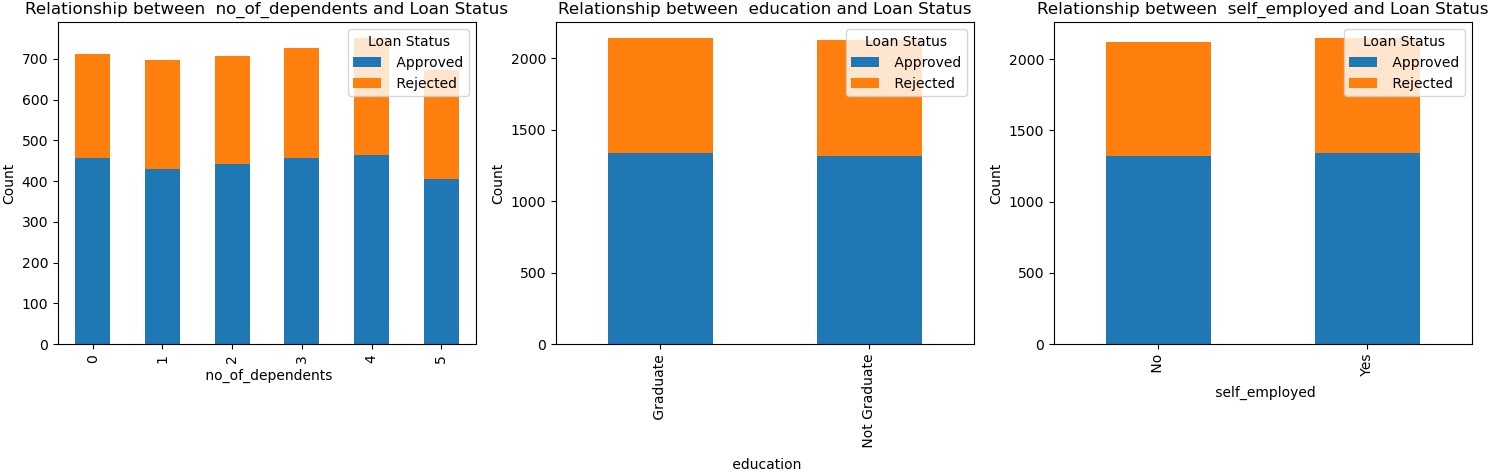
ax.set\_ylabel('Count') ax.legend(title='Loan Status')

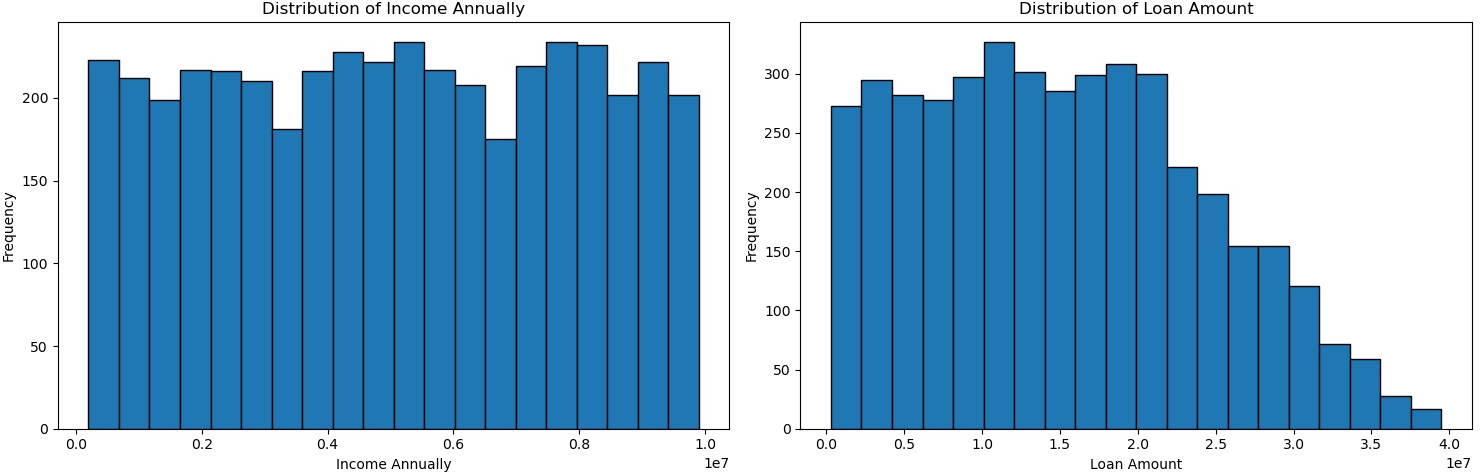
*# Adjust layout*

plt.tight\_layout()

*# Show the plots*

plt.show()



In [21]:

fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(15, 5))

*# Create histograms*

dataFrame[' income\_annum'].plot(kind='hist', bins=20, edgecolor='black', ax=axes[0]) axes[0].set\_title('Distribution of Income Annually')

axes[0].set\_xlabel('Income Annually') axes[0].set\_ylabel('Frequency')

dataFrame[' loan\_amount'].plot(kind='hist', bins=20, edgecolor='black', ax=axes[1]) axes[1].set\_title('Distribution of Loan Amount')

axes[1].set\_xlabel('Loan Amount') axes[1].set\_ylabel('Frequency')

*# Adjust layout*

plt.tight\_layout()

*# Show the plots*

plt.show()

**import matplotlib.pyplot as plt**

*# Set up subplots*

fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(15, 5))

*# Histogram for income\_annum*

axes[0].hist([dataFrame[dataFrame[' loan\_status'] == ' Approved'][' income\_annum'], dataFrame[dataFrame[' loan\_status'] == ' Rejected'][' income\_annum']], bins=20, alpha=0.5, label=[' Approved', ' Rejected'])

axes[0].set\_title('Distribution of Income\_annum by Loan Status') axes[0].set\_xlabel('Income\_annum') axes[0].set\_ylabel('Frequency')

axes[0].legend()

*# Histogram for loan\_amount*

axes[1].hist([dataFrame[dataFrame[' loan\_status'] == ' Approved'][' loan\_amount'], dataFrame[dataFrame[' loan\_status'] == ' Rejected'][' loan\_amount']], bins=20, alpha=0.5, label=[' Approved', ' Rejected'])

axes[1].set\_title('Distribution of Loan Amount by Loan Status') axes[1].set\_xlabel('Loan Amount') axes[1].set\_ylabel('Frequency')

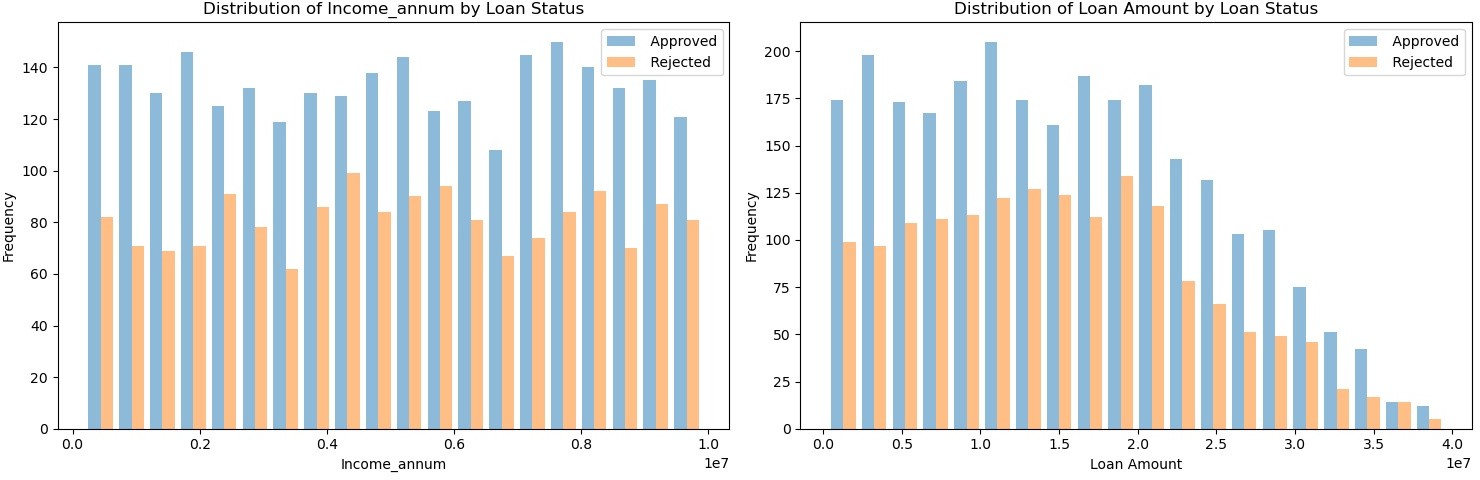
axes[1].legend()

*# Adjust layout*

plt.tight\_layout()

*# Show the plots*

plt.show()



**Handling Outliers**

**import pandas as pd**

*# Assuming df is your DataFrame*

*# Replace with your actual dataset loading mechanism # df = pd.read\_csv('your\_dataset.csv')*

*# Define a function to print and handle outliers using IQR for all numerical columns*

**def** handle\_outliers\_iqr(dataframe):

outliers = pd.DataFrame(columns=['Column', 'Outliers'])

**for** column **in** dataframe.select\_dtypes(include='number').columns: Q1 = dataframe[column].quantile(0.25)

Q3 = dataframe[column].quantile(0.75)

IQR = Q3 - Q1

lower\_bound = Q1 - 1.5 \* IQR upper\_bound = Q3 + 1.5 \* IQR

column\_outliers = dataframe[(dataframe[column] < lower\_bound) | (dataframe[column] > upper\_bound)] outliers = pd.concat([outliers, pd.DataFrame({'Column': [column], 'Outliers': [column\_outliers]})])

*# Clip the values to remove outliers*

dataframe[column] = dataframe[column].clip(lower=lower\_bound, upper=upper\_bound)

**return** outliers

*# Get outliers and print them*

outliers\_info = handle\_outliers\_iqr(dataFrame) outliers\_info

Out[62]:

**Column Outliers**

**0** loan\_id Empty DataFrame Columns: [loan\_id, no\_of\_depe...

**0** no\_of\_dependents Empty DataFrame Columns: [loan\_id, no\_of\_depe...

**0** income\_annum Empty DataFrame Columns: [loan\_id, no\_of\_depe...

**0** loan\_amount Empty DataFrame Columns: [loan\_id, no\_of\_depe...

**0** loan\_term Empty DataFrame Columns: [loan\_id, no\_of\_depe...

**0** cibil\_score Empty DataFrame Columns: [loan\_id, no\_of\_depe...

**0** residential\_assets\_value Empty DataFrame Columns: [loan\_id, no\_of\_depe...

**0** commercial\_assets\_value Empty DataFrame Columns: [loan\_id, no\_of\_depe...

**0** luxury\_assets\_value Empty DataFrame Columns: [loan\_id, no\_of\_depe...

**0** bank\_asset\_value Empty DataFrame Columns: [loan\_id, no\_of\_depe...

In [24]:

dataFrame.shape

Out[24]:

(4269, 13)

**Filter Methods for feature selection**

information gain

first converting categorical columns into numerial ones to apply filter methods

**from sklearn.preprocessing import** LabelEncoder

*# Create a copy of the original DataFrame*

encoded\_df = dataFrame.copy()

*# Initialize LabelEncoder*

label\_encoder = LabelEncoder()

*# Iterate over columns and apply label encoding for categorical columns*

**for** column **in** dataFrame.select\_dtypes(include='object').columns:

**if** len(dataFrame[column].unique()) <= 2:

*# For binary (two-level) categorical columns, use LabelEncoder*

encoded\_df[column] = label\_encoder.fit\_transform(dataFrame[column])

**else**:

*# For multi-level categorical columns, consider other encoding methods (e.g., one-hot encoding)*

print(f"Column '**{column}**' has more than two unique values. Consider other encoding methods.")

print("Encoded DataFrame:") encoded\_df

Encoded DataFrame: Out[65]:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **loan\_id** | **no\_of\_dependents** | **education** | **self\_employed** | **income\_annum** | **loan\_amount** | **loan\_term** | **cibil\_score** | **residential\_assets\_value** |
| **0** | 1 | 2 | 0 | 0 | 9600000 | 29900000 | 12 | 778 | 2400000.0 |
| **1** | 2 | 0 | 1 | 1 | 4100000 | 12200000 | 8 | 417 | 2700000.0 |
| **2** | 3 | 3 | 0 | 0 | 9100000 | 29700000 | 20 | 506 | 7100000.0 |
| **3** | 4 | 3 | 0 | 0 | 8200000 | 30700000 | 8 | 467 | 18200000.0 |
| **4** | 5 | 5 | 1 | 1 | 9800000 | 24200000 | 20 | 382 | 12400000.0 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | .. |
| **4264** | 4265 | 5 | 0 | 1 | 1000000 | 2300000 | 12 | 317 | 2800000.0 |
| **4265** | 4266 | 0 | 1 | 1 | 3300000 | 11300000 | 20 | 559 | 4200000.0 |
| **4266** | 4267 | 2 | 1 | 0 | 6500000 | 23900000 | 18 | 457 | 1200000.0 |
| **4267** | 4268 | 1 | 1 | 0 | 4100000 | 12800000 | 8 | 780 | 8200000.0 |
| **4268** | 4269 | 1 | 0 | 0 | 9200000 | 29700000 | 10 | 607 | 17800000.0 |

4269 rows × 13 columns



In [66]:

x = encoded\_df.drop(' loan\_status', axis=1) *# Assuming 'target\_feature' is the target feature*

y = encoded\_df[' loan\_status']

*#for feature selection based on mutual information for a classification task.*

**from sklearn.feature\_selection import** mutual\_info\_classif

importance = mutual\_info\_classif(x, y)

feat\_importance = pd.Series(importance, encoded\_df.columns[0:len(encoded\_df.columns) - 1]) feat\_importance.plot(kind='barh', color='teal')

plt.show()



Select KBest

**from sklearn.feature\_selection import** SelectKBest, f\_classif

*# Select features using SelectKBest and f\_classif* selector = SelectKBest(score\_func=f\_classif, k=3) X\_new = selector.fit\_transform(x, y)

*# Get the selected features*

mask = selector.get\_support() selected\_features = x.columns[mask]

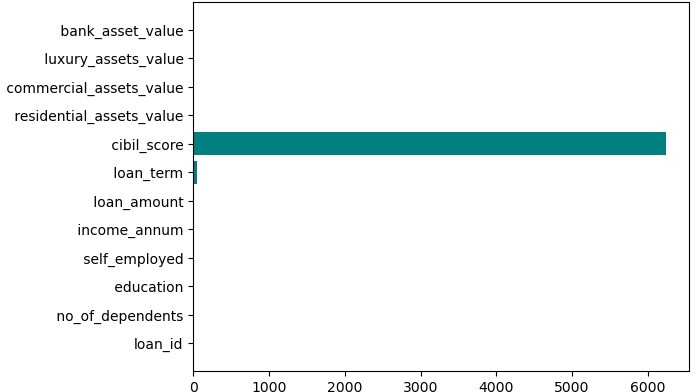
*# Plot a histogram graph*

plt.barh(range(len(selector.scores\_)), selector.scores\_, color='teal') plt.yticks(range(len(x.columns)), x.columns)

plt.show()

*# Print selected features*

print("Selected features:", selected\_features)



Selected features: Index([' no\_of\_dependents', ' loan\_term', ' cibil\_score'], dtype='object')

Fisher Score

**from sklearn.feature\_selection import** f\_classif

*# Perform feature selection using f\_classif*

fisher\_score, p\_value = f\_classif(x, y)

*# Print the results*

**for** i, column **in** enumerate(encoded\_df.columns[:-1]): *# Exclude the target column*

print("Feature:", column)

print("Fisher Score:", fisher\_score[i]) print("p-value:", p\_value[i])

print(" ")

Feature: loan\_id

Fisher Score: 1.3349674319819718

p-value: 0.2479881190731431

Feature: no\_of\_dependents Fisher Score: 1.4006001729702904

p-value: 0.23668903607767336

Feature: education

Fisher Score: 0.10320162169880803

p-value: 0.7480366078913321

Feature: self\_employed

Fisher Score: 0.0005064308399286643

p-value: 0.9820469589094434

Feature: income\_annum

Fisher Score: 0.9846688347948409

p-value: 0.32110512407104685

Feature: loan\_amount

Fisher Score: 1.1131763752970667

p-value: 0.2914522504516055

Feature: loan\_term

Fisher Score: 55.22545796738981 p-value: 1.291185463829176e-13

Feature: cibil\_score

Fisher Score: 6235.054590534256

p-value: 0.0

Feature: residential\_assets\_value Fisher Score: 0.958402864749625

p-value: 0.32764510326774676

Feature: commercial\_assets\_value Fisher Score: 0.33035269381476207

p-value: 0.565481784235544

Feature: luxury\_assets\_value Fisher Score: 1.020728281020977

p-value: 0.3124036068561609

Feature: bank\_asset\_value

Fisher Score: 0.19418768809834083

p-value: 0.6594761809444952

Mean absooute difference

mad = x.mad(axis=0)

print("Mean absolute deviation of columns:") print(mad)

Mean absolute deviation of columns: loan\_id 1.067250e+03

no\_of\_dependents 1.488298e+00

education 4.999901e-01

self\_employed 4.999736e-01

income\_annum 2.424446e+06

loan\_amount 7.580930e+06

loan\_term 4.963065e+00

cibil\_score 1.492619e+02 residential\_assets\_value 5.291511e+06 commercial\_assets\_value 3.588533e+06 luxury\_assets\_value 7.632964e+06

bank\_asset\_value 2.668145e+06 dtype: float64

**Forward feature selection**

In [33]:

**from sklearn.linear\_model import** LogisticRegression

**from mlxtend.feature\_selection import** SequentialFeatureSelector **as** SFS

*# 4. Apply Model*

lr = LogisticRegression(class\_weight='balanced', solver='lbfgs', random\_state=42, n\_jobs=-1, max\_iter=500) lr.fit(x, y)

*# 5. Select best features*

bfs = SFS(lr,

k\_features='best', forward=**True**, floating=**False**, verbose=2, scoring='accuracy', cv=0)

bfs = bfs.fit(x, y)

*# 6. Print feature list*

features = list(bfs.k\_feature\_names\_) print(features)

[Parallel(n\_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers. [Parallel(n\_jobs=1)]: Done 1 out of 1 | elapsed: 0.5s remaining: 0.0s [Parallel(n\_jobs=1)]: Done 12 out of 12 | elapsed: 2.8s finished

[2024-01-02 22:37:05] Features: 1/12 -- score: 0.9510423986882174[Parallel(n\_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

[Parallel(n\_jobs=1)]: Done 1 out of 1 | elapsed: 0.0s remaining: 0.0s [Parallel(n\_jobs=1)]: Done 11 out of 11 | elapsed: 0.7s finished

[2024-01-02 22:37:06] Features: 2/12 -- score: 0.9505739048957601[Parallel(n\_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

[Parallel(n\_jobs=1)]: Done 1 out of 1 | elapsed: 0.0s remaining: 0.0s [Parallel(n\_jobs=1)]: Done 10 out of 10 | elapsed: 0.2s finished

[2024-01-02 22:37:06] Features: 3/12 -- score: 0.9508081517919887[Parallel(n\_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [Parallel(n\_jobs=1)]: | Done | 1 out of | 1 | elapsed: | 0.0s remaining: | 0.0s |
| [Parallel(n\_jobs=1)]: | Done | 9 out of | 9 | elapsed: | 0.6s finished |  |

[2024-01-02 22:37:07] Features: 4/12 -- score: 0.9501054111033029[Parallel(n\_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [Parallel(n\_jobs=1)]: | Done | 1 out of | 1 | elapsed: | 0.0s remaining: | 0.0s |
| [Parallel(n\_jobs=1)]: | Done | 8 out of | 8 | elapsed: | 0.2s finished |  |

[2024-01-02 22:37:07] Features: 5/12 -- score: 0.949402670414617[Parallel(n\_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [Parallel(n\_jobs=1)]: | Done | 1 out of | 1 | elapsed: | 0.0s remaining: | 0.0s |
| [Parallel(n\_jobs=1)]: | Done | 7 out of | 7 | elapsed: | 1.0s finished |  |

[2024-01-02 22:37:08] Features: 6/12 -- score: 0.9226985242445538[Parallel(n\_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [Parallel(n\_jobs=1)]: | Done | 1 out of | 1 | elapsed: | 0.0s remaining: | 0.0s |
| [Parallel(n\_jobs=1)]: | Done | 6 out of | 6 | elapsed: | 0.1s finished |  |

[2024-01-02 22:37:08] Features: 7/12 -- score: 0.7390489576013117[Parallel(n\_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

[Parallel(n\_jobs=1)]: Done 1 out of 1 | elapsed: 0.0s remaining: 0.0s

[Parallel(n\_jobs=1)]: Done 5 out of 5 | elapsed: 0.1s finished

[2024-01-02 22:37:08] Features: 8/12 -- score: 0.7390489576013117[Parallel(n\_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [Parallel(n\_jobs=1)]: | Done | 1 out of | 1 | elapsed: | 0.0s remaining: | 0.0s |
| [Parallel(n\_jobs=1)]: | Done | 4 out of | 4 | elapsed: | 0.0s finished |  |

[2024-01-02 22:37:08] Features: 9/12 -- score: 0.7388147107050832[Parallel(n\_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [Parallel(n\_jobs=1)]: | Done | 1 out of | 1 | elapsed: | 0.0s remaining: | 0.0s |
| [' cibil\_score']  [Parallel(n\_jobs=1)]: | Done | 3 out of | 3 | elapsed: | 0.1s finished |  |

[2024-01-02 22:37:08] Features: 10/12 -- score: 0.7434996486296557[Parallel(n\_jobs=1)]: Using backen d SequentialBackend with 1 concurrent workers.

[Parallel(n\_jobs=1)]: Done 1 out of 1 | elapsed: 0.0s remaining: 0.0s [Parallel(n\_jobs=1)]: Done 2 out of 2 | elapsed: 0.0s finished

[2024-01-02 22:37:08] Features: 11/12 -- score: 0.624033731553057[Parallel(n\_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| [Parallel(n\_jobs=1)]: Done | 1 out of | 1 | elapsed: | 0.0s remaining: | 0.0s |
| [Parallel(n\_jobs=1)]: Done | 1 out of | 1 | elapsed: | 0.0s finished |  |

[2024-01-02 22:37:08] Features: 12/12 -- score: 0.5427500585617241

**Exhaustive Feature Selection:**

**from sklearn.neighbors import** KNeighborsClassifier

**from mlxtend.feature\_selection import** ExhaustiveFeatureSelector **as** EFS

x = encoded\_df.drop(' loan\_status', axis=1) *# Assuming 'target\_feature' is the target feature*

y = encoded\_df[' loan\_status']

knn = KNeighborsClassifier(n\_neighbors=3) efs1 = EFS(knn,

min\_features=1, max\_features=4, scoring='accuracy', print\_progress=**True**, cv=5)

efs1 = efs1.fit(x, y)

print('Best accuracy score: **%.2f**' % efs1.best\_score\_) print('Best subset (indices): ', efs1.best\_idx\_)

print('Best subset (corresponding names): ', efs1.best\_feature\_names\_)

Features: 793/793

Best accuracy score: 0.95

Best subset (indices): (2, 3, 6, 7)

Best subset (corresponding names): (' education', ' self\_employed', ' loan\_term', ' cibil\_score')

**Random Forest**

In [36]:

*# Apply Random Forest regression algorithm*

**from sklearn.ensemble import** RandomForestRegressor

rf = RandomForestRegressor(random\_state=0) rf.fit(X\_train, y\_train)

*# Print selected features*

print(rf.feature\_importances\_) print(np.array(features)[rf.feature\_importances\_ > 0.1])

[8.34715611e-03 3.46046217e-03 3.96891375e-04 8.49391812e-04 2.19010605e-02 2.84329259e-02 7.66550462e-02 8.32448350e-01

8.28227062e-03 6.78039033e-03 7.91008880e-03 4.53596631e-03]

[' cibil\_score']

**Genetic algorithm**

**import random**

**from sklearn.metrics import** accuracy\_score

**from sklearn.linear\_model import** LogisticRegression **from sklearn.model\_selection import** train\_test\_split **from sklearn.preprocessing import** StandardScaler

**import pandas as pd**

logmodel = LogisticRegression(max\_iter=1000) scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(x)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.33, random\_state=42)

*# defining various steps required for the genetic algorithm*

**def** initialization\_of\_population(size, n\_feat): population = []

**for** i **in** range(size):

chromosome = np.ones(n\_feat, dtype=bool) chromosome[:int(0.3 \* n\_feat)] = **False** np.random.shuffle(chromosome) population.append(chromosome)

**return** population

**def** fitness\_score(population, X\_train, X\_test, y\_train, y\_test, logmodel): scores = []

**for** chromosome **in** population: logmodel.fit(X\_train[:, chromosome], y\_train)

predictions = logmodel.predict(X\_test[:, chromosome]) scores.append(accuracy\_score(y\_test, predictions))

scores, population = np.array(scores), np.array(population) inds = np.argsort(scores)

**return** list(scores[inds][::-1]), list(population[inds, :][::-1])

**def** selection(pop\_after\_fit, n\_parents): population\_next\_gen = []

**for** i **in** range(n\_parents): population\_next\_gen.append(pop\_after\_fit[i])

**return** population\_next\_gen

**def** crossover(pop\_after\_sel): population\_next\_gen = pop\_after\_sel.copy() **for** i **in** range(len(pop\_after\_sel)):

child = pop\_after\_sel[i].copy()

child[3:7] = pop\_after\_sel[(i + 1) % len(pop\_after\_sel)][3:7] population\_next\_gen.append(child)

**return** population\_next\_gen

**def** mutation(pop\_after\_cross, mutation\_rate): population\_next\_gen = []

**for** i **in** range(0, len(pop\_after\_cross)): chromosome = pop\_after\_cross[i].copy() **for** j **in** range(len(chromosome)):

**if** random.random() < mutation\_rate: chromosome[j] = **not** chromosome[j]

population\_next\_gen.append(chromosome)

**return** population\_next\_gen

**def** generations(size, n\_feat, n\_parents, mutation\_rate, n\_gen, X\_train, X\_test, y\_train, y\_test, logmodel): best\_chromo = []

best\_score = []

population\_next\_gen = initialization\_of\_population(size, n\_feat)

**for** i **in** range(n\_gen):

scores, pop\_after\_fit = fitness\_score(population\_next\_gen, X\_train, X\_test, y\_train, y\_test, logmodel) print(scores[:2])

pop\_after\_sel = selection(pop\_after\_fit, n\_parents) pop\_after\_cross = crossover(pop\_after\_sel)

population\_next\_gen = mutation(pop\_after\_cross, mutation\_rate) best\_chromo.append(pop\_after\_fit[0]) best\_score.append(scores[0])

**return** best\_chromo, best\_score

*# Implementing GA*

*# Replace the following placeholders with your actual data # X\_train, X\_test, y\_train, y\_test, logmodel*

chromo, score = generations(size=200, n\_feat=12, n\_parents=100, mutation\_rate=0.10, n\_gen=5, X\_train=X\_train, X\_test=X\_test, y\_train=y\_train, y\_test=y\_test, logmodel=logmodel)

logmodel.fit(X\_train[:, chromo[-1]], y\_train) predictions = logmodel.predict(X\_test[:, chromo[-1]])

print("Accuracy score after genetic algorithm is = " + str(accuracy\_score(y\_test, predictions))) print('List of important features:', chromo[-1])

[0.9247693399574166, 0.9240596167494677]

[0.9254790631653655, 0.9247693399574166]

[0.9254790631653655, 0.9247693399574166]

[0.9254790631653655, 0.9247693399574166]

[0.9254790631653655, 0.9254790631653655]

Accuracy score after genetic algorithm is = 0.9254790631653655

List of important features: [ True False False True False True False True True True True True

]

In [38]:

x.columns

Out[38]:

Index(['loan\_id', ' no\_of\_dependents', ' education', ' self\_employed', ' income\_annum', ' loan\_amount', ' loan\_term', ' cibil\_score', ' residential\_assets\_value', ' commercial\_assets\_value',

' luxury\_assets\_value', ' bank\_asset\_value'], dtype='object')

so according to all the feature selection method cibil\_score is best one for prediction dropping those columns which have zero impact on the prediction of loan\_status

In [39]:

*# Drop specified columns*

columns\_to\_drop = ['loan\_id', ' income\_annum', ' commercial\_assets\_value'] final\_df = encoded\_df.drop(columns=columns\_to\_drop)

*# Display the resulting DataFrame*

final\_df

Out[39]:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | | |  |
|  | **no\_of\_dependents** | **education** | **self\_employed** | **loan\_amount** | **loan\_term** | **cibil\_score** | **residential\_assets\_value** | **luxury\_assets\_value** |  |
| **0** | 2 | 0 | 0 | 29900000 | 12 | 778 | 2400000.0 | 22700000 |  |
| **1** | 0 | 1 | 1 | 12200000 | 8 | 417 | 2700000.0 | 8800000 |  |
| **2** | 3 | 0 | 0 | 29700000 | 20 | 506 | 7100000.0 | 33300000 |  |
| **3** | 3 | 0 | 0 | 30700000 | 8 | 467 | 18200000.0 | 23300000 |  |
| **4** | 5 | 1 | 1 | 24200000 | 20 | 382 | 12400000.0 | 29400000 |  |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... |  |
| **4264** | 5 | 0 | 1 | 2300000 | 12 | 317 | 2800000.0 | 3300000 |  |
| **4265** | 0 | 1 | 1 | 11300000 | 20 | 559 | 4200000.0 | 11000000 |  |
| **4266** | 2 | 1 | 0 | 23900000 | 18 | 457 | 1200000.0 | 18100000 |  |
| **4267** | 1 | 1 | 0 | 12800000 | 8 | 780 | 8200000.0 | 14100000 |  |
| **4268** | 1 | 0 | 0 | 29700000 | 10 | 607 | 17800000.0 | 35700000 |  |

**Decision Tree**



4269 rows × 10 columns

In [70]:

x = final\_df.drop(' loan\_status', axis=1) *# Assuming 'target\_feature' is the target feature*

y = final\_df[' loan\_status']

In [71]:

*# Import necessary libraries*

**from sklearn.tree import** DecisionTreeClassifier

**from sklearn.model\_selection import** train\_test\_split

*# Perform train and test split*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(x, y, random\_state=0)

y\_train\_reshaped = y\_train.values.reshape(-1, 1)

*# Build Decision Tree classifier*

clf = DecisionTreeClassifier(criterion='gini', max\_depth=4, random\_state=1) clf.fit(X\_train, y\_train\_reshaped)

Out[71]:

▾

DecisionTreeClassifier

DecisionTreeClassifier(max\_depth=4, random\_state=1)

In [44]:

*# Extract relevant attributes from the tree structure*

n\_nodes = clf.tree\_.node\_count children\_left = clf.tree\_.children\_left children\_right = clf.tree\_.children\_right feature = clf.tree\_.feature

threshold = clf.tree\_.threshold

node\_depth = np.zeros(shape=n\_nodes, dtype=np.int64) is\_leaves = np.zeros(shape=n\_nodes, dtype=bool)

stack = [(0, 0)] *# start with the root node id (0) and its depth (0)*

*# Traverse the tree structure to compute depth and identify leaves*

**while** len(stack) > 0:

*# 'pop' ensures each node is only visited once* node\_id, depth = stack.pop() node\_depth[node\_id] = depth

*# If the left and right child of a node is not the same, we have a split node*

is\_split\_node = children\_left[node\_id] != children\_right[node\_id]

*# If a split node, append left and right children and depth to 'stack' # so we can loop through them*

**if** is\_split\_node: stack.append((children\_left[node\_id], depth + 1)) stack.append((children\_right[node\_id], depth + 1))

**else**:

is\_leaves[node\_id] = **True**

*# Print the binary tree structure*

print("The binary tree structure has **{n}** nodes and has the following tree structure:**\n**".format(n=n\_nodes))

**for** i **in** range(n\_nodes):

**if** is\_leaves[i]:

print("**{space}**node=**{node}** is a leaf node.".format(space=node\_depth[i] \* "**\t**", node=i))

**else**:

print("**{space}**node=**{node}** is a split node: "

"go to node **{left}** if X[:, **{feature}**] <= **{threshold}** " "else to node **{right}**.".format(space=node\_depth[i] \* "**\t**",

node=i, left=children\_left[i], feature=feature[i], threshold=threshold[i], right=children\_right[i]))

The binary tree structure has 23 nodes and has the following tree structure:

node=0 is a split node: go to node 1 if X[:, 5] <= 549.5 else to node 10. node=1 is a split node: go to node 2 if X[:, 4] <= 5.0 else to node 9.

node=2 is a split node: go to node 3 if X[:, 3] <= 26250000.0 else to node 6.

node=3 is a split node: go to node 4 if X[:, 7] <= 20350000.0 else to node 5

.

7.

21.

node=4 is a leaf node. node=5 is a leaf node.

node=6 is a split node: go to node 7 if X[:, 6] <= 400000.0 else to node 8. node=7 is a leaf node.

node=8 is a leaf node. node=9 is a leaf node.

node=10 is a split node: go to node 11 if X[:, 6] <= 250000.0 else to node 18. node=11 is a split node: go to node 12 if X[:, 5] <= 693.5 else to node 15.

node=12 is a split node: go to node 13 if X[:, 0] <= 0.5 else to node 14. node=13 is a leaf node.

node=14 is a leaf node.

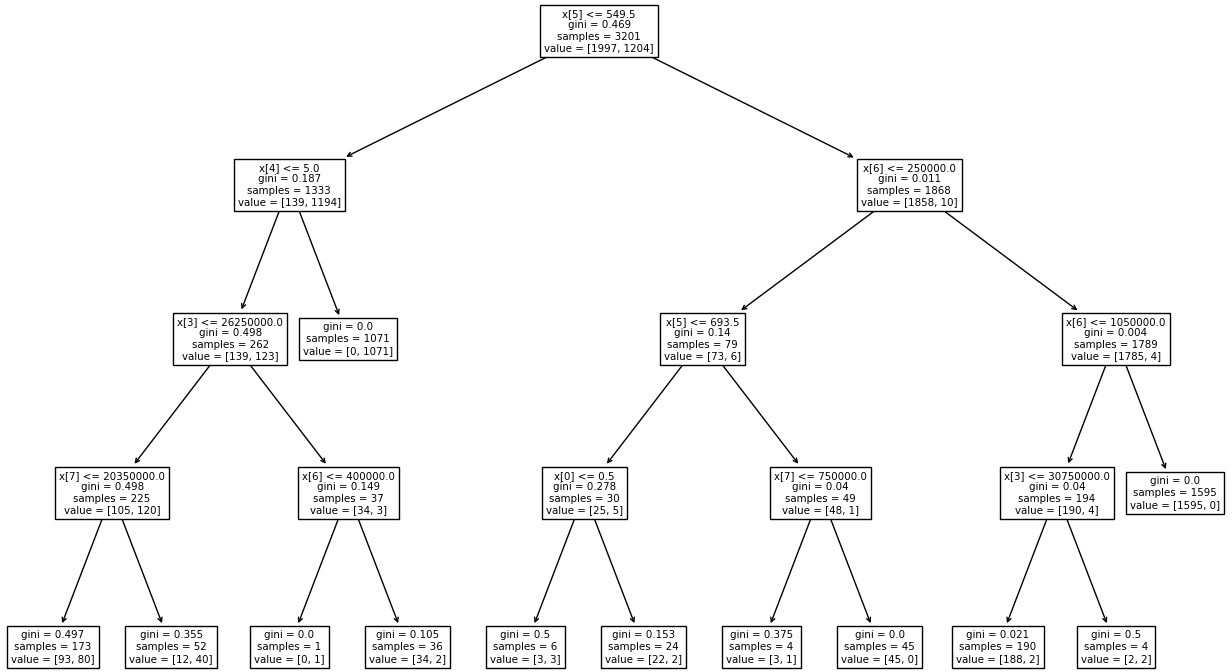
node=15 is a split node: go to node 16 if X[:, 7] <= 750000.0 else to node 1

node=16 is a leaf node. node=17 is a leaf node.

node=18 is a split node: go to node 19 if X[:, 6] <= 1050000.0 else to node 22. node=19 is a split node: go to node 20 if X[:, 3] <= 30750000.0 else to node

node=20 is a leaf node. node=21 is a leaf node.

node=22 is a leaf node.

In [45]:

**from sklearn.tree import** plot\_tree

**import matplotlib.pyplot as plt**

*# Plot the decision tree* plt.figure(figsize=(16, 10)) plot\_tree(clf)

plt.show()

**KNN Algorithm**

**import pandas as pd**

**from sklearn.model\_selection import** train\_test\_split

**from sklearn.neighbors import** KNeighborsClassifier

**from sklearn.metrics import** accuracy\_score, classification\_report, confusion\_matrix

*# Initialize and train the KNN classifier* knn\_classifier = KNeighborsClassifier(n\_neighbors=3) knn\_classifier.fit(X\_train, y\_train)

*# Make predictions with KNN on the test set*

knn\_predictions = knn\_classifier.predict(X\_test)

*#Evaluate the performane of KNN*

knn\_accuracy = accuracy\_score(y\_test, knn\_predictions) knn\_classification\_report = classification\_report(y\_test, knn\_predictions) knn\_confusion\_matrix = confusion\_matrix(y\_test, knn\_predictions)

*# Display results*

print("KNN Accuracy:", knn\_accuracy) print("**\n**KNN Classification Report:") print(knn\_classification\_report) print("**\n**KNN Confusion Matrix:") print(knn\_confusion\_matrix)

KNN Accuracy: 0.5430711610486891

KNN Classification Report:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | precision | recall | f1-score | support |
| 0 | 0.62 | 0.69 | 0.65 | 659 |
| 1 | 0.38 | 0.30 | 0.34 | 409 |
| accuracy |  |  | 0.54 | 1068 |
| macro avg | 0.50 | 0.50 | 0.49 | 1068 |
| weighted avg | 0.52 | 0.54 | 0.53 | 1068 |

KNN Confusion Matrix: [[456 203]

[285 124]]

**Base models and voting**

*# evaluate standard models on the synthetic dataset*

**from numpy import** mean

**from numpy import** std

**from sklearn.datasets import** make\_classification

**from sklearn.model\_selection import** cross\_val\_score

**from sklearn.model\_selection import** RepeatedStratifiedKFold

**from sklearn.linear\_model import** LogisticRegression **from sklearn.neighbors import** KNeighborsClassifier **from sklearn.tree import** DecisionTreeClassifier **from sklearn.svm import** SVC

**from sklearn.naive\_bayes import** GaussianNB

**from matplotlib import** pyplot

**from sklearn.ensemble import** VotingClassifier

*# get the dataset*

*# get a list of models to evaluate*

**def** get\_models (): models = list()

models.append(('lr', LogisticRegression())) models.append(('tree', DecisionTreeClassifier())) models.append (('nb', GaussianNB())) models.append(('svm', SVC(probability=**True**))) **return** models

*# evaluate a give model using cross - validation*

**def** evaluate\_model (model , X, y):

*# define the model evaluation procedure*

cv = RepeatedStratifiedKFold( n\_splits=10 , n\_repeats=3 , random\_state=1)

*# evaluate the model*

scores = cross\_val\_score(model, X, y, scoring='accuracy', cv=cv, n\_jobs= -1)

**return** scores

*# get the models to evaluate*

models = get\_models()

*# evaluate the models and store results*

results , names = list() , list()

**for** name , model **in** models :

*# evaluate model*

scores =evaluate\_model ( model ,x , y)

*# Store results* results.append(scores) names.append(name) *#summarize result*

print('>**%s %.3f** (**%.3f**)' % (name, mean(scores), std(scores)))

*# plot model performance for comparison*

pyplot.boxplot (results , labels=names , showmeans=**True** ) pyplot.show ()

*#create the ensemble*

ensemble = VotingClassifier( estimators = models , voting = 'soft')

*# define the evaluation procedure*

cv = RepeatedStratifiedKFold ( n\_splits =10 , n\_repeats =3 , random\_state =1)

*# evaluate the ensemble*

scores = cross\_val\_score (ensemble , x, y, scoring ='accuracy', cv=cv , n\_jobs=-1)

*# summarize the result*

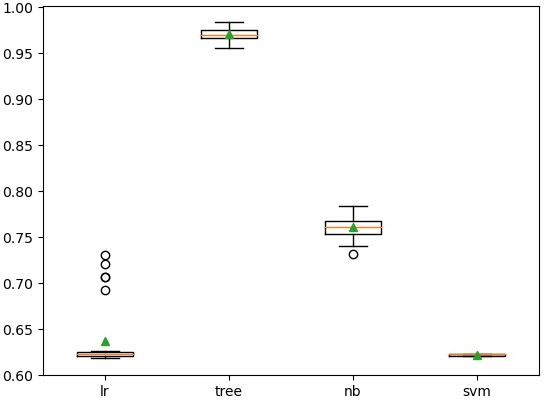
print('Mean Accuracy: **%.3f** (**%.3f**)' % (mean(scores), std(scores)))

>lr 0.637 (0.034)

>tree 0.971 (0.007)

>nb 0.761 (0.011)

>svm 0.622 (0.001)



Mean Accuracy: 0.966 (0.012)

**Ensemble pruning**

*# evaluate a list of models*

**def** evaluate\_ensemble (models , X, y):

*# check for no models*

**if** len(models) == 0:

**return** 0.0

*# create the ensemble*

ensemble = VotingClassifier(estimators = models , voting = 'soft')

*# define the evaluation procedure*

cv = RepeatedStratifiedKFold ( n\_splits=10 , n\_repeats=3 , random\_state=1)

*# evaluate the ensemble*

scores = cross\_val\_score (ensemble , X, y, scoring ='accuracy', cv=cv ,n\_jobs= -1)

*# return mean score*

**return** mean(scores)

*# perform a single round of pruning the ensemble*

**def** prune\_round (models\_in , X, y):

*# establish a baseline*

baseline = evaluate\_ensemble (models\_in , X, y) best\_score , removed = baseline , **None**

*# enumerate removing each candidate and see if we can improve performance*

**for** m **in** models\_in :

*# copy the list of chosen models*

dup = models\_in.copy() *# remove this model* dup.remove(m) *#evaluate new ensemble*

result = evaluate\_ensemble (dup, X , y)

*# check for new best*

**if** result > best\_score :

*# store the new best*

best\_score , removed = result , m

**return** best\_score , removed

*# prune an ensemble from scratch*

**def** prune\_ensemble ( models , X , y): best\_score = 0.0

*# prune ensemble until no further improvement*

**while True** :

*# remove one model to the ensemble*

score , removed = prune\_round(models ,X, y)

*# check for no improvement*

**if** removed **is None** :

print ('>no further improvement')

**break**

*# keep track of best score*

best\_score = score

*# remove model from the list*

models.remove( removed )

*# report results along the way*

print('>**%.3f** (removed: **%s**)' % (score, removed[0]))

**return** best\_score , models

*# get the models to evaluate*

models = get\_models ()

*# prune the ensemble*

score , model\_list = prune\_ensemble(models,x, y) names = ','. join ([ n **for** n , \_ **in** model\_list ]) print ('Models : **%s** '% names )

print ('Final Mean Accuracy : **%.3f**' % score )

>0.972 (removed: svm) Models : lr,tree,nb

Final Mean Accuracy : 0.972

**Bagging and Random Forest classifier**

**from sklearn.model\_selection import** KFold, cross\_val\_score

**from sklearn.ensemble import** BaggingClassifier

**from sklearn.tree import** DecisionTreeClassifier

**from sklearn.model\_selection import** train\_test\_split **from sklearn** . ensemble **import RandomForestClassifier from sklearn.datasets import** load\_diabetes

**from sklearn.metrics import** f1\_score, accuracy\_score

**from sklearn.metrics import** confusion\_matrix

**import seaborn as sns**

**import matplotlib.pyplot as plt import pandas as pd**

max\_features = 3

kfold = KFold(n\_splits=10, shuffle=**True**, random\_state=2020) decision\_tree = DecisionTreeClassifier(max\_features = max\_features) num\_trees = 100

*# Decision Tree base estimator*

bagging\_model = BaggingClassifier(base\_estimator=decision\_tree, n\_estimators = num\_trees, random\_state=2020)

results = cross\_val\_score(bagging\_model, X\_train, y\_train, cv=kfold)

print("Bagging classifier Accuracy: **%0.2f** (+/- **%0.2f**)" % (results.mean(), results.std()))

*# Bagging Classifier with Decision Tree base estimator*

bagging\_model.fit(X\_train, y\_train) y\_pred\_bagging = bagging\_model.predict(X\_test)

accuracy\_bagging = accuracy\_score(y\_test, y\_pred\_bagging) f1\_bagging = f1\_score(y\_test, y\_pred\_bagging)

print("Bagging Classifier Accuracy: **%0.2f**" % accuracy\_bagging) print("Bagging Classifier F1 Score: **%0.2f**" % f1\_bagging)

*# Bagging Classifier with Decision Tree base estimator*

conf\_matrix\_bagging = confusion\_matrix(y\_test, y\_pred\_bagging)

*# Plot confusion matrix for Bagging Classifier*

plt.figure(figsize=(4, 4))

sns.heatmap(conf\_matrix\_bagging, annot=**True**, fmt="d", cmap="Blues", cbar=**False**, xticklabels=['Class 0', 'Class 1'], yticklabels=['Class 0', 'Class 1'])

plt.title("Confusion Matrix - Bagging Classifier") plt.xlabel("Predicted Labels")

plt.ylabel("True Labels") plt.show()

*# Random Forest Classifier*

num\_trees\_rf = 100

max\_features\_rf = 3

kfold\_rf = KFold(n\_splits=10, shuffle=**True**, random\_state=2020)

rf\_model = RandomForestClassifier(n\_estimators=num\_trees\_rf, max\_features= max\_features)

*# Cross-validation*

results\_rf = cross\_val\_score(rf\_model, X\_train, y\_train, cv=kfold\_rf)

print("Random Forest classifier Accuracy: **%0.2f** (+/- **%0.2f**)" % (results\_rf.mean(),results\_rf.std()))

*#accuracy and F1 score* rf\_model.fit(X\_train, y\_train) y\_pred\_rf = rf\_model.predict(X\_test)

accuracy\_rf = accuracy\_score(y\_test, y\_pred\_rf) f1\_rf = f1\_score(y\_test, y\_pred\_rf)

print("Random Forest Classifier Accuracy: **%0.2f**" % accuracy\_rf) print("Random Forest Classifier F1 Score: **%0.2f**" % f1\_rf)

*# Random Forest Classifier*

conf\_matrix\_rf = confusion\_matrix(y\_test, y\_pred\_rf)

*# Plot confusion matrix for Random Forest Classifier*

plt.figure(figsize=(4, 4))

sns.heatmap(conf\_matrix\_rf, annot=**True**, fmt="d", cmap="Blues", cbar=**False**, xticklabels=['Class 0', 'Class 1'], yticklabels=['Class 0', 'Class 1'])

plt.title("Confusion Matrix - Random Forest Classifier") plt.xlabel("Predicted Labels")

plt.ylabel("True Labels") plt.show()

/opt/anaconda3/envs/myenv/lib/python3.9/site-packages/sklearn/ensemble/\_base.py:156: FutureWarning:

`base\_estimator` was renamed to `estimator` in version 1.2 and will be removed in 1.4. warnings.warn(

/opt/anaconda3/envs/myenv/lib/python3.9/site-packages/sklearn/ensemble/\_base.py:156: FutureWarning:

`base\_estimator` was renamed to `estimator` in version 1.2 and will be removed in 1.4. warnings.warn(

/opt/anaconda3/envs/myenv/lib/python3.9/site-packages/sklearn/ensemble/\_base.py:156: FutureWarning:

`base\_estimator` was renamed to `estimator` in version 1.2 and will be removed in 1.4. warnings.warn(

/opt/anaconda3/envs/myenv/lib/python3.9/site-packages/sklearn/ensemble/\_base.py:156: FutureWarning:

`base\_estimator` was renamed to `estimator` in version 1.2 and will be removed in 1.4. warnings.warn(

/opt/anaconda3/envs/myenv/lib/python3.9/site-packages/sklearn/ensemble/\_base.py:156: FutureWarning:

`base\_estimator` was renamed to `estimator` in version 1.2 and will be removed in 1.4. warnings.warn(

/opt/anaconda3/envs/myenv/lib/python3.9/site-packages/sklearn/ensemble/\_base.py:156: FutureWarning:

`base\_estimator` was renamed to `estimator` in version 1.2 and will be removed in 1.4. warnings.warn(

/opt/anaconda3/envs/myenv/lib/python3.9/site-packages/sklearn/ensemble/\_base.py:156: FutureWarning:

`base\_estimator` was renamed to `estimator` in version 1.2 and will be removed in 1.4. warnings.warn(

/opt/anaconda3/envs/myenv/lib/python3.9/site-packages/sklearn/ensemble/\_base.py:156: FutureWarning:

`base\_estimator` was renamed to `estimator` in version 1.2 and will be removed in 1.4. warnings.warn(

/opt/anaconda3/envs/myenv/lib/python3.9/site-packages/sklearn/ensemble/\_base.py:156: FutureWarning:

`base\_estimator` was renamed to `estimator` in version 1.2 and will be removed in 1.4. warnings.warn(

/opt/anaconda3/envs/myenv/lib/python3.9/site-packages/sklearn/ensemble/\_base.py:156: FutureWarning:

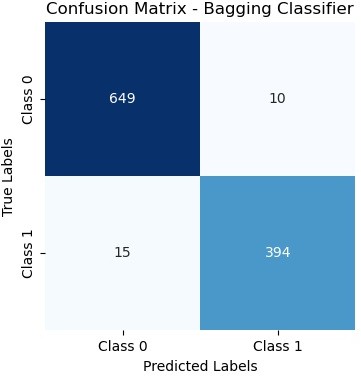
`base\_estimator` was renamed to `estimator` in version 1.2 and will be removed in 1.4. warnings.warn(

Bagging classifier Accuracy: 0.97 (+/- 0.01)

/opt/anaconda3/envs/myenv/lib/python3.9/site-packages/sklearn/ensemble/\_base.py:156: FutureWarning:

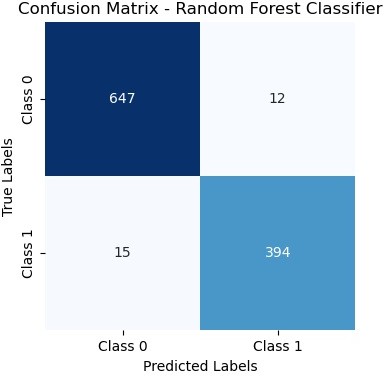
`base\_estimator` was renamed to `estimator` in version 1.2 and will be removed in 1.4. warnings.warn(

Bagging Classifier Accuracy: 0.98 Bagging Classifier F1 Score: 0.97



Random Forest classifier Accuracy: 0.97 (+/- 0.01) Random Forest Classifier Accuracy: 0.97

Random Forest Classifier F1 Score: 0.97



**Stacking**

In [76]:

**import numpy as np**

**from sklearn.ensemble import** AdaBoostClassifier, RandomForestClassifier

**from sklearn.linear\_model import** LogisticRegression

**from sklearn.tree import** DecisionTreeClassifier

**from sklearn.metrics import** f1\_score, accuracy\_score

**from sklearn.datasets import** load\_breast\_cancer

**from sklearn.metrics import** confusion\_matrix

**from sklearn.ensemble import** RandomForestClassifier

**import seaborn as sns**

**import matplotlib.pyplot as plt**

**class NumberOfClassifierException**(**Exception**): **pass**

**class Stacking**():

**def**  init (self, classifiers):

**if** len(classifiers) < 2:

**raise** numberOfClassifierException (" You must fit your classifier with 2 classifiers at least ");

**else**:

self.\_classifiers = classifiers

**def** fit(self, data\_x, data\_y): stacked\_data\_x = data\_x.copy()

**for** classifier **in** self.\_classifiers[:-1]: classifier.fit(data\_x, data\_y)

stacked\_data\_x = np.column\_stack((stacked\_data\_x ,classifier.predict\_proba(data\_x))) last\_classifier = self.\_classifiers[-1]

last\_classifier.fit(stacked\_data\_x, data\_y)

**def** predict(self, data\_x): stacked\_data\_x = data\_x.copy()

**for** classifier **in** self.\_classifiers[:-1]: prob\_predictions = classifier.predict\_proba(data\_x)

stacked\_data\_x = np.column\_stack ((stacked\_data\_x , prob\_predictions)) last\_classifier = self.\_classifiers[-1]

**return** last\_classifier.predict(stacked\_data\_x)

*# Creating classifiers*

boosting\_clf\_ada\_boost = AdaBoostClassifier( base\_estimator=DecisionTreeClassifier(max\_depth=1), n\_estimators=3) clf\_rf = RandomForestClassifier( n\_estimators=200, max\_depth=1, random\_state=2020)

clf\_adaboost = AdaBoostClassifier( base\_estimator=DecisionTreeClassifier(max\_depth=1, random\_state=2020), n\_estim ators=3)

clf\_logistic\_reg = LogisticRegression(solver='liblinear', random\_state=2020)

*# Customizing and Exception message*

classifiers\_list = [clf\_rf, clf\_adaboost, clf\_logistic\_reg] clf\_stacking = Stacking(classifiers\_list)

*# Fit models*

clf\_rf.fit(X\_train, y\_train) boosting\_clf\_ada\_boost.fit(X\_train, y\_train)

clf\_stacking.fit(X\_train, y\_train)

*# Make predictions*

predictions\_bagging = clf\_rf.predict(X\_test) predictions\_boosting = boosting\_clf\_ada\_boost.predict(X\_test) predictions\_stacking = clf\_stacking.predict(X\_test)

*# Print results*

print("For Bagging: F1 Score **{}**, Accuracy **{}**".format( round(f1\_score(y\_test, predictions\_bagging), 2),

round(accuracy\_score(y\_test, predictions\_bagging), 2)

))

*# Confusion Matrix for Bagging*

conf\_mat\_bagging = confusion\_matrix(y\_test, predictions\_bagging)

*# Plot Confusion Matrix for Bagging*

plt.figure(figsize=(6, 4))

sns.heatmap(conf\_mat\_bagging, annot=**True**, fmt="d", cmap="Blues", xticklabels=["Benign", "Malignant"], yticklabels

=["Benign", "Malignant"]) plt.xlabel('Predicted') plt.ylabel('Actual')

plt.title('Confusion Matrix - Bagging Classifier') plt.show()

print("For Boosting: F1 Score **{}**, Accuracy **{}**".format( round(f1\_score(y\_test, predictions\_boosting), 2),

round(accuracy\_score(y\_test, predictions\_boosting), 2)

))

*# Confusion Matrix for Bagging*

conf\_mat\_bagging = confusion\_matrix(y\_test, predictions\_boosting)

*# Plot Confusion Matrix for Bagging*

plt.figure(figsize=(6, 4))

sns.heatmap(conf\_mat\_bagging, annot=**True**, fmt="d", cmap="Blues", xticklabels=["Benign", "Malignant"], yticklabels

=["Benign", "Malignant"]) plt.xlabel('Predicted') plt.ylabel('Actual')

plt.title('Confusion Matrix - Boosting Classifier') plt.show()

print("For Stacking: F1 Score **{}**, Accuracy **{}**".format( round(f1\_score(y\_test, predictions\_stacking), 2),

round(accuracy\_score(y\_test, predictions\_stacking), 2)

))

*# Confusion Matrix for Bagging*

conf\_mat\_bagging = confusion\_matrix(y\_test, predictions\_stacking)

*# Plot Confusion Matrix for Bagging*

plt.figure(figsize=(6, 4))

sns.heatmap(conf\_mat\_bagging, annot=**True**, fmt="d", cmap="Blues", xticklabels=["Benign", "Malignant"], yticklabels

=["Benign", "Malignant"]) plt.xlabel('Predicted') plt.ylabel('Actual')

plt.title('Confusion Matrix - Stacking Classifier') plt.show()

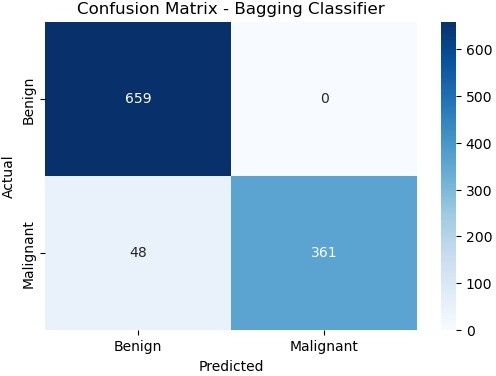
/opt/anaconda3/envs/myenv/lib/python3.9/site-packages/sklearn/ensemble/\_base.py:156: FutureWarning:

`base\_estimator` was renamed to `estimator` in version 1.2 and will be removed in 1.4. warnings.warn(

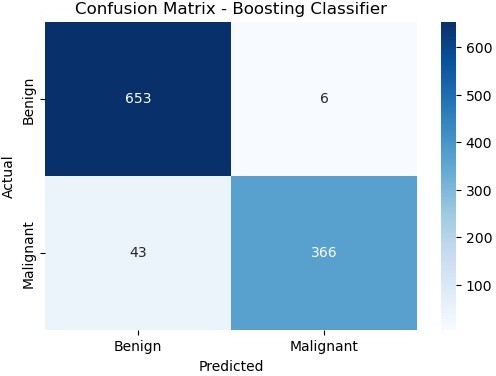
/opt/anaconda3/envs/myenv/lib/python3.9/site-packages/sklearn/ensemble/\_base.py:156: FutureWarning:

`base\_estimator` was renamed to `estimator` in version 1.2 and will be removed in 1.4. warnings.warn(

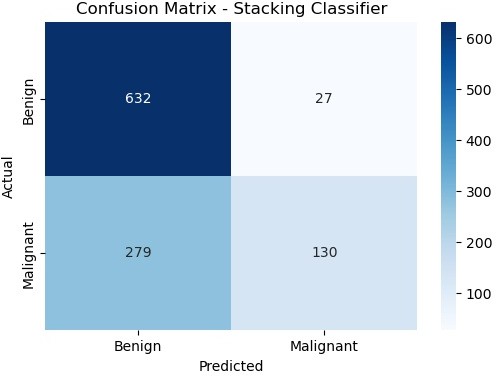
For Bagging: F1 Score 0.94, Accuracy 0.96



For Boosting: F1 Score 0.94, Accuracy 0.95



For Stacking: F1 Score 0.46, Accuracy 0.71



**Naive Bayes Classifier**

In [77]:

**class NaiveBayesClassifier**:

**def**  init (self, data, target):

*# Constructor to initialize the classifier with input data and target variable*

self.data = data self.target = target

self.target\_prob\_dict = {} *# Dictionary to store target class probabilities* self.cond\_prob\_list = [] *# List to store conditional probabilities* self.smoothing = **False** *# Flag to enable/disable Laplace smoothing*

self.k = 1 *# Laplace Smoothing parameter*

**def** compute\_target\_probabilities(self):

*# Compute probabilities of each target class in the dataset*

total\_rows = len(self.data)

target\_counts = self.data[self.target].value\_counts()

**for** level, count **in** target\_counts.items(): self.target\_prob\_dict[level] = count / total\_rows

**def** compute\_conditional\_probabilities(self):

*# Compute conditional probabilities for each feature given the target class*

**for** feature **in** self.data.columns:

**if** feature != self.target:

feature\_levels = self.data[feature].unique()

rget\_level)])

**for** level **in** feature\_levels:

**for** target\_level **in** self.target\_prob\_dict.keys():

count\_f\_v\_t = len(self.data[(self.data[feature] == level) & (self.data[self.target] == ta

count\_f\_t = len(self.data[self.data[self.target] == target\_level])

**if** self.smoothing:

*# Laplace Smoothing*

prob = (count\_f\_v\_t + self.k) / (count\_f\_t + self.k \* len(feature\_levels))

**else**:

prob = count\_f\_v\_t / count\_f\_t

*# Store the computed probability in the list*

self.cond\_prob\_list.append((feature, level, target\_level, round(prob, 3)))

**def** fit(self, smoothing=**False**, k=1):

*# Method to train the Naive Bayes classifier*

self.smoothing = smoothing self.k = k

self.compute\_target\_probabilities() self.compute\_conditional\_probabilities()

**def** predict\_instance(self, instance):

*# Predict the target class for a given instance*

prob\_dict = {}

**for** target\_level **in** self.target\_prob\_dict.keys(): prob\_prod = 1

**for** feature, level, t\_level, prob **in** self.cond\_prob\_list:

**if** feature **in** instance **and** instance[feature] == level **and** t\_level == target\_level: prob\_prod \*= prob

*# Multiply the conditional probability with the prior probability*

prob\_dict[target\_level] = round(prob\_prod \* self.target\_prob\_dict[target\_level], 4)

*# Return the predicted target class with the highest probability*

**return** max(prob\_dict, key=prob\_dict.get)

**def** get\_probabilities(self):

*# Display the computed target and conditional probabilities* print("Target probabilities:", self.target\_prob\_dict) print("Conditional features probabilities:", self.cond\_prob\_list)

*# Example usage:*

data = {

'CrdtHistory': ['current', 'paid', 'arrears', 'none', 'current', 'paid', 'arrears', 'none'], 'GCoApplicant': ['none', 'guarantor', 'coapplicant', 'none', 'none', 'guarantor', 'coapplicant', 'none'], 'Accommodation': ['own', 'rent', 'free', 'own', 'rent', 'free', 'own', 'rent'],

'Target': [**True**, **True**, **True**, **True**, **False**, **False**, **False**, **False**]

}

df = pd.DataFrame(data)

nbc = NaiveBayesClassifier(final\_df, ' loan\_status') nbc.fit(smoothing=**True**, k=1)

*# Prediction for a new instance*

query\_instance = {' no\_of\_dependents': 5, ' education': 1, ' self\_employed': 1, ' loan\_amount': 24300000, ' loan\_ term': 12, ' cibil\_score': 317, ' residential\_assests\_value': 18200000.0, ' luxury\_assests\_value': 29400000, ' ba

nk\_asset\_value': 800000.0}

prediction = nbc.predict\_instance(query\_instance) print("Prediction:", prediction)

Prediction: 0

**Linear Regression Model**

In [90]:

**import numpy as np**

**import matplotlib.pyplot as plt**

**from sklearn.metrics import** r2\_score

**def** gradient\_descent(X\_train, y\_train, learning\_rate=0.0001, epochs=1000): n = len(X\_train)

a\_0 = 0.0

a\_1 = 0.0

**for** epoch **in** range(epochs): y\_pred = a\_0 + a\_1 \* X\_train error = y\_pred - y\_train

mean\_sq\_err = np.sum(error \*\* 2) / n

a\_0 = a\_0 - learning\_rate \* 2 \* np.sum(error) / n

a\_1 = a\_1 - learning\_rate \* 2 \* np.sum(error \* X\_train) / n

**if** epoch % 100 == 0:

print(f'Mean Squared Error after **{epoch}** epochs:', mean\_sq\_err)

**return** a\_0, a\_1

**def** linear\_regression(X\_train, y\_train, X\_test): a\_0, a\_1 = gradient\_descent(X\_train, y\_train)

y\_pred\_train = a\_0 + a\_1 \* X\_train y\_pred\_test = a\_0 + a\_1 \* X\_test

**return** y\_pred\_train, y\_pred\_test, a\_0, a\_1

*# Performing linear regression*

y\_pred\_train, y\_pred\_test, a\_0, a\_1 = linear\_regression(X\_train, y\_train, X\_test)

*# Combine training and test sets for plotting* X\_combined = np.concatenate((X\_train, X\_test)) y\_combined = np.concatenate((y\_train, y\_pred\_test))

*# Calculating R2 Score*

r2 = r2\_score(y\_train, y\_pred\_train) print('R2 Score:', r2)

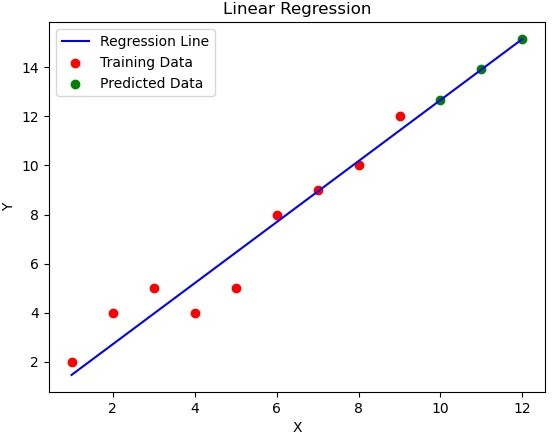
*# Plotting*

plt.scatter(X\_train, y\_train, color='red', label='Training Data') plt.plot(X\_combined, a\_0 + a\_1 \* X\_combined, color='blue', label='Regression Line') plt.scatter(X\_test, y\_pred\_test, color='green', label='Predicted Data') plt.legend()

plt.xlabel('X')

plt.ylabel('Y') plt.title('Linear Regression') plt.show()

Mean Squared Error after 0 epochs: 52.77777777777778 Mean Squared Error after 100 epochs: 14.917031978341372 Mean Squared Error after 200 epochs: 4.62610266991101 Mean Squared Error after 300 epochs: 1.8286803492572576 Mean Squared Error after 400 epochs: 1.0680036202264156 Mean Squared Error after 500 epochs: 0.8609191543258855 Mean Squared Error after 600 epochs: 0.8043043436911523 Mean Squared Error after 700 epochs: 0.7885900306638032 Mean Squared Error after 800 epochs: 0.7839951845567169 Mean Squared Error after 900 epochs: 0.7824252690486773 R2 Score: 0.9202568099499092



**SDG Regressor**

In [63]:

**from sklearn.linear\_model import** SGDRegressor **from sklearn.preprocessing import** StandardScaler **from sklearn.metrics import** mean\_squared\_error **from sklearn.metrics import** r2\_score

*# Feature scaling (important for SGD)*

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train) X\_test\_scaled = scaler.transform(X\_test)

*# Create an SGDRegressor instance*

regressor = SGDRegressor(max\_iter=1000, tol=1e-3, random\_state=42)

*# Manually capture the loss values during training*

loss\_values = []

**for** epoch **in** range(1000): *# Adjust the number of epochs as needed*

regressor.partial\_fit(X\_train\_scaled, y\_train) y\_pred = regressor.predict(X\_train\_scaled)

mse = mean\_squared\_error(y\_train, y\_pred) loss\_values.append(mse)

*# Predict on the test set*

y\_pred = regressor.predict(X\_test\_scaled)

*# Calculate and print the Mean Squared Error (MSE) as a measure of accuracy*

mse = mean\_squared\_error(y\_test, y\_pred) print("Mean Squared Error:", mse)

*# Calculate R-squared score* score = r2\_score(y\_test, y\_pred) print("R-squared Score:", score)

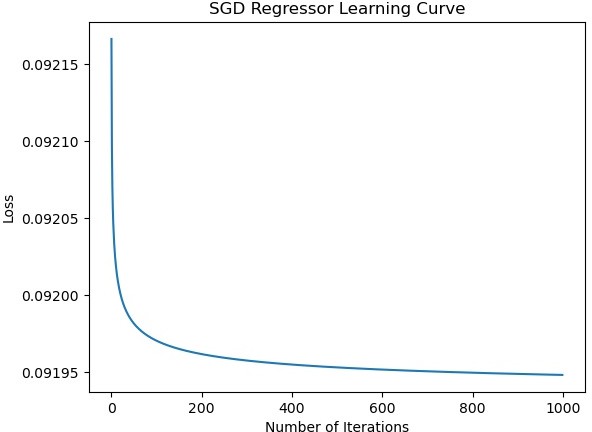
*# Plot the learning graph*

plt.plot(loss\_values)

plt.title("SGD Regressor Learning Curve") plt.xlabel("Number of Iterations") plt.ylabel("Loss")

plt.show()

Mean Squared Error: 0.09042501884190475 R-squared Score: 0.6173317848725053



In [ ]: