**PRACTICAL JOURNAL IN**

**APPLIED ARTIFICIAL INTELLIGENCE**

**MACHINE LEARNING**

**SUBMITTED BY:**

**VINAYAK SHANKAR PATADE**

**ROLL NO: 2414564**

**IN PARTIAL FULLFILMENT FOR THE DEGREE OF**

**MASTER OF SCIENCE IN INFORMATION TECHNOLOGY PART – II SEMESTER III**

**ACADEMIC YEAR**

**2024-202**Logo

Description automatically generated**5**

**PARLE TILAK VIDYALAYA ASSOCIATION’S**

**MULUND COLLEGE OF COMMERCE(AUTONOMOUS)**

***(AFFILIATED TO UNIVERSITY OF MUMBAI)***

***NAAC RE-ACCREDITED A GRADE – III CYCLE***

**MULUND WEST, MUMBAI 400080**

**MAHARASHTRA, INDIA**

**2024-25**

Parle Tilak Vidyalaya Association’s

MULUND COLLEGE OF COMMERCE (AUTONOMOUS)

***(Affiliated to University of Mumbai)***

**MULUND WEST, MUMBAI 400080**

**MAHARASHTRA, INDIA**

DEPARTMENT OF INFORMATION TECHNOLOGY

CERTIFICATE

This is to certify that **Vinayak S Patade** of **M.Sc. I.T. Part II** Roll No **2414564** has successfully completed the practical work in **Applied Artificial Intelligence** in partial fulfilment of the requirements for the Semester III of **M.Sc. I.T. Part II** during the academic year **2024-25**.

Teacher In-charge and Coordinator

Examiner

Date:

College Seal

Parle Tilak Vidyalaya Association’s

MULUND COLLEGE OF COMMERCE (AUTONOMOUS)

***(Affiliated to University of Mumbai)***

**MULUND WEST, MUMBAI 400080**

**MAHARASHTRA, INDIA**

DEPARTMENT OF INFORMATION TECHNOLOGY

CERTIFICATE

This is to certify that **Vinayak S Patade** of **M.Sc. I.T. Part II** Roll No **2414564** has successfully completed the practical work in **Machine Learning** in partial fulfilment of the requirements for the Semester III of **M.Sc. I.T. Part II** during the academic year **2024-25**.

Teacher In-charge and Coordinator

Examiner

Date:

College Seal

Table of Contents

[Practical 1: Design a bot using AIML 2](#_Toc180618006)

[Practical 2: Design an Expert system using AIML 5](#_Toc180618007)

[Practical 3: Implement Bayes Theorem using Python 9](#_Toc180618008)

[Practical 4: Implement Conditional Probability and joint probability using Python. 10](#_Toc180618009)

[Practical 5: Write a program to implement Rule based system. (Prolog) 11](#_Toc180618010)

[Practical 6: Design a Fuzzy based application using Python / R 16](#_Toc180618011)

[Practical 7A: Write an application to stimulate supervised learning model. 20](#_Toc180618012)

[7B: Write an application to stimulate unsupervised learning model. 21](#_Toc180618013)

[Practical 8: Write an application to implement clustering algorithm. 23](#_Toc180618014)

[Practical 9: Write an application to implement support vector machine algorithm. 25](#_Toc180618015)

[Practical 10: Simulate artificial neural network model with both feedforward and backpropagation approach. 28](#_Toc180618016)

## 

## Practical 1: Design a bot using AIML

**Std-startup.xml**

<aiml version="1.0.1" encoding="UTF-8">

    <!-- std-startup.xml -->

    <!-- Category is an atomic AIML unit -->

    <category>

        <!-- Pattern to match in user input -->

        <!-- If user enters "LOAD AIML B" -->

        <pattern>LOAD AIML B</pattern>

        <!-- Template is the response to the pattern -->

        <!-- This learn an aiml file -->

        <template>

            <learn>basic\_chat.aiml</learn>

            <!-- You can add more aiml files here -->

            <!--<learn>more\_aiml.aiml</learn>-->

        </template>

    </category>

</aiml>

**Basic\_chat.aiml**

<aiml version="1.0.1" encoding="UTF-8">

    <!-- basic\_chat.aiml -->

    <category>

        <pattern>HELLO</pattern>

        <template>

            Well, hello!

        </template>

    </category>

    <category>

        <pattern>WHAT ARE YOU</pattern>

        <template> I'm a bot, silly! </template>

    </category>

    <category>

<pattern>MY NAME IS \*</pattern>

        <template>

            <set name="username">

                <star />

            </set> is the nice name. </template>

    </category>

    <category>

        <pattern>I LIKE \*</pattern>

        <template>

            <set name="liking">

                <star />

            </set> is also my favourite. </template>

    </category>

    <category>

        <pattern>MY DOG NAME IS \*</pattern>

        <template> THAT IS INTERESTING THAT YOU HAVE A DOG NAMED <set name="dog">

                <star />

            </set> . </template>

    </category>

    <category>

        <pattern>BYE</pattern>

        <template> Bye!!! <get name="username" /> Thanks for talking with me. </template>

    </category>

</aiml>

**Chatbot.py**

import aiml # Create the kernel and learn AIML files

kernel = aiml.Kernel()

kernel.learn("std-startup.xml")

kernel.respond("load aiml b") # Press CTRL-C to break this loop

while True:

message = input("Enter your message to the bot: ")

if message == "quit":

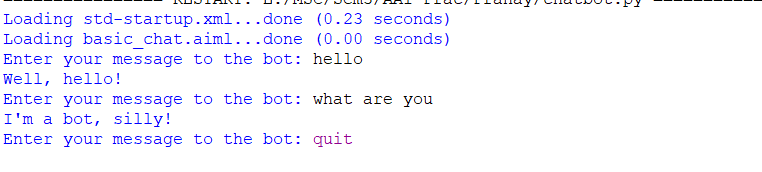
break

else:

bot\_response = kernel.respond(message)

print(bot\_response)

Output:



## 

## Practical 2: Design an Expert system using AIML

**Std\_startup.xml**

<aiml version="1.0.1" encoding="UTF-8">

    <!-- std-startup.xml -->

    <!-- Category is an atomic AIML unit -->

    <category>

        <!-- Pattern to match in user input -->

        <!-- If user enters "LOAD AIML B" -->

        <pattern>LOAD AIML B</pattern>

        <!-- Template is the response to the pattern -->

        <!-- This learn an aiml file -->

        <template>

            <learn>basic\_chat.aiml</learn>

            <!-- You can add more aiml files here -->

            <!--<learn>more\_aiml.aiml</learn>-->

        </template>

    </category>

</aiml>

**Basic\_chat.aiml**

<aiml version="1.0.1" encoding="UTF-8">

    <!-- basic\_chat.aiml -->

    <category>

    <pattern>HELLO</pattern>

    <template>

    WHAT WOULD YOU LIKE TO DISCUSS? : HEALTH, MOVIES

    </template>

    </category>

    <category>

    <pattern>MOVIES</pattern>

    <template>

        YES <set name = "topic">MOVIES</set>

        </template>

</category>

        <category>

        <pattern>HEALTH</pattern>

        <template> YES <set name = "topic">HEALTH</set> </template>

        </category>

        <topic name ="MOVIES">

        <category>

        <pattern>\*</pattern>

        <template>

        DO YOU LIKE COMEDY MOVIES?

        </template>

        </category>

        <category> <pattern>YES</pattern>

        <template>

        I TOO LIKE COMEDY MOVIES

        </template>

        </category>

        <category>

        <pattern>NO</pattern>

        <template>

        BUT I LIKE COMEDY MOVIES

        </template>

        </category>

        </topic>

        <topic name ="HEALTH">

        <category>

        <pattern>\*</pattern>

        <template>

        DO YOU HAVE FEVER?

        </template>

        </category>

        <category>

        <pattern>YES</pattern>

        <template>

        PLEASE TAKE MEDICINES AND PROPER REST

        </template></category>

        <category>

        <pattern>NO</pattern>

        <template>

        GO OUT FOR A WALK AND LISTEN MUSIC

        </template>

        </category>

        </topic>

        <category>

        <pattern>NICE TALKING TO YOU</pattern>

        <template>

        SAME HERE...!!

        </template>

        </category>

        </aiml>

**Chatbot.py**

import aiml

# Create the kernel and learn AIML files

kernel = aiml.Kernel()

kernel.learn("std-startup.xml")

kernel.respond("LOAD AIML B")

# Press CTRL-C to break this loop

while True:

message = input("Enter your message to the bot: ")

if message == "quit":

break

else:

bot\_response = kernel.respond(message)

print(bot\_response)

Output:

A screenshot of a computer program

Description automatically generated

## 

## Practical 3: Implement Bayes Theorem using Python

# calculate the probability of cancer patient and diagnostic test

# calculate P(A|B) given P(A), P(B|A), P(B|not A)

def bayes\_theorem(p\_a, p\_b\_given\_a, p\_b\_given\_not\_a):

# calculate P(not A)

not\_a = 1 - p\_a

# calculate P(B)

p\_b = p\_b\_given\_a \* p\_a + p\_b\_given\_not\_a \* not\_a

# calculate P(A|B)

p\_a\_given\_b = (p\_b\_given\_a \* p\_a) / p\_b

return p\_a\_given\_b

# P(A)

p\_a = 0.0002

# P(B|A)

p\_b\_given\_a = 0.85

# P(B|not A)

p\_b\_given\_not\_a = 0.05

# calculate P(A|B)

result = bayes\_theorem(p\_a, p\_b\_given\_a, p\_b\_given\_not\_a)

# summarize

print('P(A|B) = %.3f%%' % (result \* 100))

Output:



## 

## Practical 4: Implement Conditional Probability and joint probability using Python.

import enum, random

class Kid(enum.Enum):

BOY = 0

GIRL = 1

def random\_kid() -> Kid:

return random.choice([Kid.BOY, Kid.GIRL])

both\_girls = 0

older\_girl = 0

either\_girl = 0

random.seed(0)

for \_ in range(10000):

younger = random\_kid()

older = random\_kid()

if older == Kid.GIRL:

older\_girl += 1

if older == Kid.GIRL and younger == Kid.GIRL:

both\_girls += 1

if older == Kid.GIRL or younger == Kid.GIRL:

either\_girl += 1

print("older girl: ", older\_girl)

print("both girl: ", both\_girls)

print("either girl: ", either\_girl)

print("P(both | older):", both\_girls / older\_girl)

print("P(both | either):", both\_girls / either\_girl)

Output:

A close up of a number

Description automatically generated

## 

## Practical 5: Write a program to implement Rule based system. (Prolog)

go:-

hypothesis(Disease),

write('I believe that the patient have '),

write(Disease),

nl,

write('TAKE CARE '),

undo.

/\*Hypothesis that should be tested\*/

hypothesis(cold) :- cold, !.

hypothesis(flu) :- flu, !.

hypothesis(typhoid) :- typhoid, !.

hypothesis(measles) :- measles, !.

hypothesis(malaria) :- malaria, !.

hypothesis(unknown). /\* no diagnosis\*/

/\*Hypothesis Identification Rules\*/

cold :-

verify(headache),

verify(runny\_nose),

verify(sneezing),

verify(sore\_throat),

write('Advices and Sugestions:'),

nl,

write('1: Tylenol/tab'),

nl,

write('2: panadol/tab'),

nl,

write('3: Nasal spray'),

nl,

write('Please wear warm cloths Because'),

nl.

flu :-

verify(fever),

verify(headache),

verify(chills),

verify(body\_ache),

write('Advices and Sugestions:'),

nl,

write('1: Tamiflu/tab'),

nl,

write('2: panadol/tab'),

nl,

write('3: Zanamivir/tab'),

nl,

write('Please take a warm bath and do salt gargling Because'),

nl.

typhoid :-

verify(headache),

verify(abdominal\_pain),

verify(poor\_appetite),

verify(fever),

write('Advices and Sugestions:'),

nl,

write('1: Chloramphenicol/tab'),

nl,

write('2: Amoxicillin/tab'),

nl,

write('3: Ciprofloxacin/tab'),

nl,

write('4: Azithromycin/tab'),

nl,

write('Please do complete bed rest and take soft Diet Because'),

nl.

measles :-

verify(fever),

verify(runny\_nose),

verify(rash),

verify(conjunctivitis),

write('Advices and Sugestions:'),

nl,

write('1: Tylenol/tab'),

nl,

write('2: Aleve/tab'),

nl,

write('3: Advil/tab'),

nl,

write('4: Vitamin A'),

nl,

write('Please Get rest and use more liquid Because'),

nl.

malaria :-

verify(fever),

verify(sweating),

verify(headache),

verify(nausea),

verify(vomiting),

verify(diarrhea),

write('Advices and Sugestions:'),

nl,

write('1: Aralen/tab'),

nl,

write('2: Qualaquin/tab'),

nl,

write('3: Plaquenil/tab'),

nl,

write('4: Mefloquine'),

nl,

write('Please do not sleep in open air and cover your full skin Because'),

nl.

/\* how to ask questions \*/

ask(Question) :-

write('Does the patient have following symptom:'),

write(Question),

write('? '),

read(Response),

nl,

( (Response == yes ; Response == y)

->

assert(yes(Question)) ;

assert(no(Question)), fail).

:- dynamic yes/1,no/1.

/\*How to verify something \*/

verify(S) :-

(yes(S)

->

true ;

(no(S)

->

fail ;

ask(S))).

/\* undo all yes/no assertions\*/

undo :- retract(yes(\_)),fail.

undo :- retract(no(\_)),fail.

undo.

Output:

A screenshot of a computer

Description automatically generated

## 

## Practical 6: Design a Fuzzy based application using Python / R

import numpy as np

import skfuzzy as fuzz

import matplotlib.pyplot as plt

from skfuzzy import control as ctrl

from mpl\_toolkits.mplot3d import Axes3D # Required for 3D plotting

# New Antecedent/Consequent objects hold universe variables and membership

# functions

quality = ctrl.Antecedent(np.arange(0, 10, 0.1), 'quality')

service = ctrl.Antecedent(np.arange(0, 10, 0.1), 'service')

tip = ctrl.Consequent(np.arange(0, 25, 0.1), 'tip')

quality['poor'] = fuzz.zmf(quality.universe, 0,5)

quality['average'] = fuzz.gaussmf(quality.universe,5,1)

quality['good'] = fuzz.smf(quality.universe,5,10)

service['poor'] = fuzz.zmf(service.universe, 0,5)

service['average'] = fuzz.gaussmf(service.universe,5,1)

service['good'] = fuzz.smf(service.universe,5,10)

tip['low'] = fuzz.trimf(tip.universe, [0, 0, 13])

tip['medium'] = fuzz.trimf(tip.universe, [0, 13, 25])

tip['high'] = fuzz.trimf(tip.universe, [13, 25, 25])

quality['average'].view()

plt.title('Quality')

service['poor'].view()

plt.title('Service')

tip['medium'].view()

plt.title('Tip Medium')

rule1 = ctrl.Rule(quality['poor'] | service['poor'], tip['low'])

rule2 = ctrl.Rule(service['average'], tip['medium'])

rule3 = ctrl.Rule(service['good'] | quality['good'], tip['high'])

rule1.view()

plt.title('Rule 1')

rule2.view()

plt.title('Rule 2')

rule3.view()

plt.title('Rule 3')

tipping\_ctrl = ctrl.ControlSystem([rule1, rule2, rule3])

tipping = ctrl.ControlSystemSimulation(tipping\_ctrl)

tipping.input['quality'] = 6.5

tipping.input['service'] = 9.8

tipping.compute()

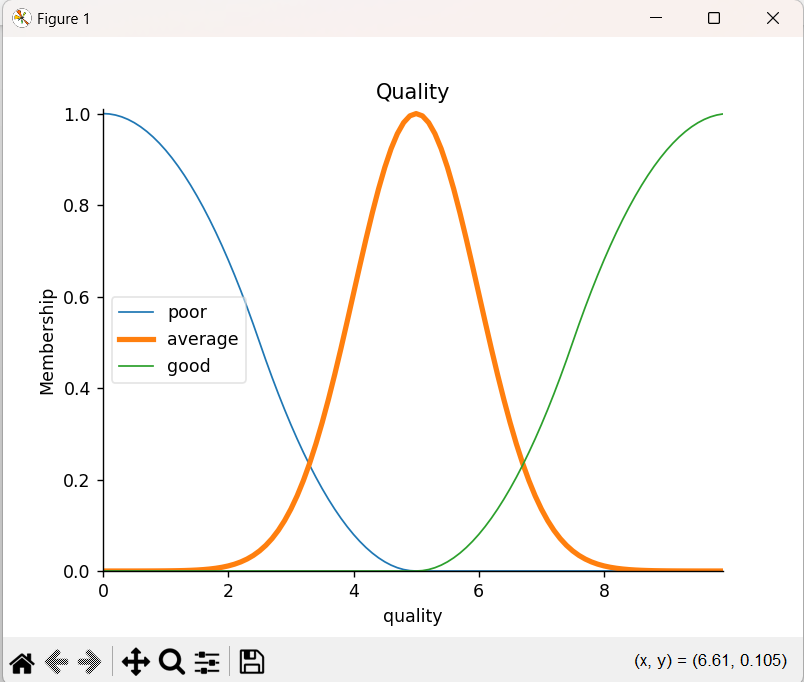
plt.title('Result')

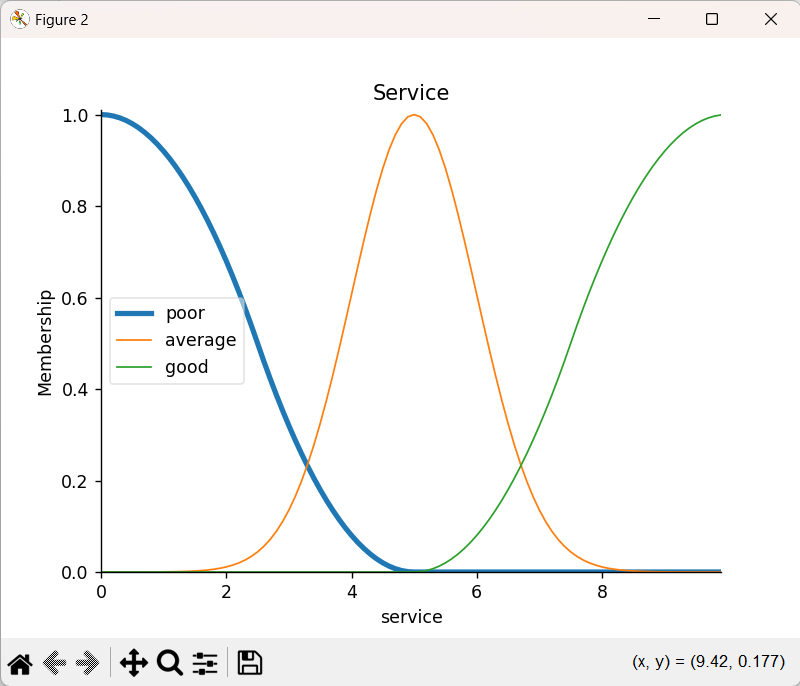
plt.show(block=True)

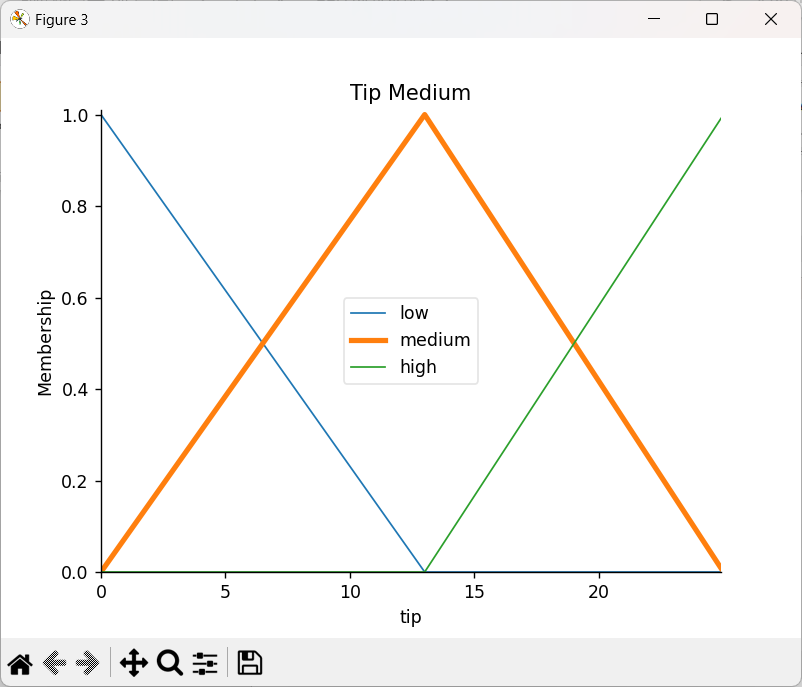
print(tipping.output['tip'])

tip.view(sim=tipping)

Output:







## 

## Practical 7A: Write an application to stimulate supervised learning model.

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

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import classification\_report, confusion\_matrix

from sklearn import datasets

iris=datasets.load\_iris()

x = iris.data

y = iris.target

print ('sepal-length', 'sepal-width', 'petal-length', 'petal-width')

print(x)

print('class: 0-Iris-Setosa, 1- Iris-Versicolour, 2- Iris-Virginica')

print(y)

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x,y,test\_size=0.3)

#To Training the model and Nearest nighbors K=5

classifier = KNeighborsClassifier(n\_neighbors=5)

classifier.fit(x\_train, y\_train)

#To make predictions on our test data

y\_pred=classifier.predict(x\_test)

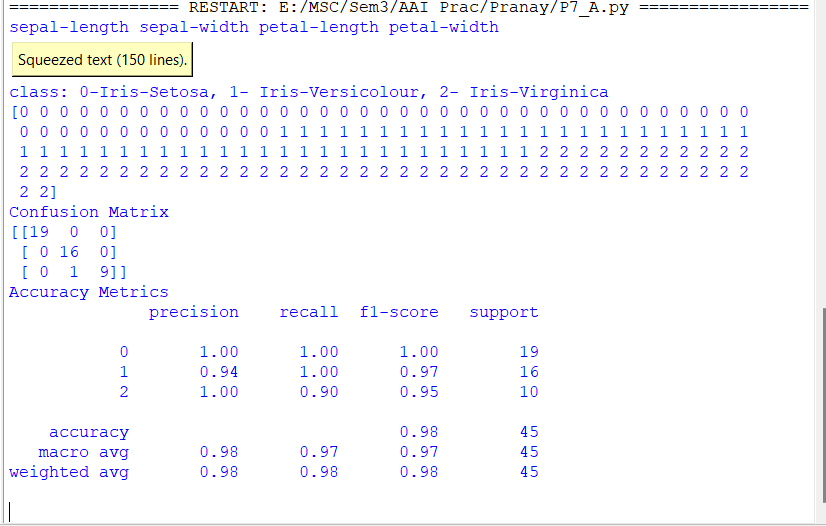
print('Confusion Matrix')

print(confusion\_matrix(y\_test,y\_pred))

print('Accuracy Metrics')

print(classification\_report(y\_test,y\_pred))

Output:



## 7B: Write an application to stimulate unsupervised learning model.

from scipy.cluster.hierarchy import linkage, dendrogram

import matplotlib.pyplot as plt

import pandas as pd

# Reading the DataFrame

seeds\_df = pd.read\_csv("seeds-less-rows.csv")

# Remove the grain species from the DataFrame, save for later

varieties = list(seeds\_df.pop('grain\_variety'))

# Extract the measurements as a NumPy array

samples = seeds\_df.values

"""

Perform hierarchical clustering on samples using the

linkage() function with the method='complete' keyword argument.

Assign the result to mergings.

"""

mergings = linkage(samples, method='complete')

"""

Plot a dendrogram using the dendrogram() function on mergings,

specifying the keyword arguments labels=varieties, leaf\_rotation=90,

and leaf\_font\_size=6.

"""

dendrogram(mergings,

labels=varieties,

leaf\_rotation=90,

leaf\_font\_size=6,

)

plt.show()

Output:

A screenshot of a graph

Description automatically generated

## 

## Practical 8: Write an application to implement clustering algorithm.

import matplotlib.pyplot as plt

from sklearn import datasets

from sklearn.cluster import KMeans

import sklearn.metrics as sm

import pandas as pd

import numpy as np

iris = datasets.load\_iris()

X = pd.DataFrame(iris.data)

X.columns = ['Sepal\_Length','Sepal\_Width','Petal\_Length','Petal\_Width']

y = pd.DataFrame(iris.target)

y.columns = ['Targets']

model = KMeans(n\_clusters=3)

model.fit(X)

plt.figure(figsize=(14,7))

colormap = np.array(['red', 'lime', 'black'])

# Plot the Original Classifications

plt.subplot(1, 2, 1)

plt.scatter(X.Petal\_Length, X.Petal\_Width,

c=colormap[y.Targets], s=40)

plt.title('Real Classification')

plt.xlabel('Petal Length')

plt.ylabel('Petal Width')

# Plot the Models Classifications

plt.subplot(1, 2, 2)

plt.scatter(X.Petal\_Length, X.Petal\_Width,

c=colormap[model.labels\_], s=40)

plt.title('K Mean Classification')

plt.xlabel('Petal Length')

plt.ylabel('Petal Width')

plt.show()

print('The accuracy score of K-Mean: ',sm.accuracy\_score(y, model.labels\_))

print('The Confusion matrix of K-Mean: ',sm.confusion\_matrix(y, model.labels\_))

Output:

A screenshot of a graph

Description automatically generated

## 

## Practical 9: Write an application to implement support vector machine algorithm.

#Import scikit-learn dataset library

from sklearn import datasets

#Import svm model

from sklearn import svm

# Import train\_test\_split function

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

#Import scikit-learn metrics module for accuracy calculation

from sklearn import metrics

#Load dataset

cancer = datasets.load\_breast\_cancer()

# print the names of the 13 features

print("Features: ", cancer.feature\_names)

# print the label type of cancer('malignant' 'benign')

print("Labels: ", cancer.target\_names)

# print data(feature)shape

cancer.data.shape

# print the cancer data features (top 5 records)

print(cancer.data[0:5])

# print the cancer labels (0:malignant, 1:benign)

print(cancer.target)

# Split dataset into training set and test set

X\_train, X\_test, y\_train, y\_test = train\_test\_split(cancer.data, cancer.target, test\_size=0.3,random\_state=109) # 70% training and 30% test

#Create a svm Classifier

clf = svm.SVC(kernel='linear') # Linear Kernel

#Train the model using the training sets

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

#Predict the response for test dataset

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

# Model Accuracy: how often is the classifier correct?

print("Accuracy:",metrics.accuracy\_score(y\_test, y\_pred))

# Model Precision: what percentage of positive tuples are labeled as such?

print("Precision:",metrics.precision\_score(y\_test, y\_pred))

# Model Recall: what percentage of positive tuples are labelled as such?

print("Recall:",metrics.recall\_score(y\_test, y\_pred))

Output:

A screenshot of a computer

Description automatically generated

A number grid with numbers

Description automatically generated with medium confidence

## 

## Practical 10: Simulate artificial neural network model with both feedforward and backpropagation approach.

import numpy as np

X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float) # two inputs [sleep,study]

y = np.array(([92], [86], [89]), dtype=float) # one output [Expected % in Exams]

X = X / np.amax(X, axis=0) # maximum of X array longitudinally

y = y / 100

# Sigmoid Function

def sigmoid(x):

return 1 / (1 + np.exp(-x))

# Derivative of Sigmoid Function

def derivatives\_sigmoid(x):

return x \* (1 - x)

# Variable initialization

epoch = 5000 # Setting training iterations

lr = 0.1 # Setting learning rate

inputlayer\_neurons = 2 # number of features in data set

hiddenlayer\_neurons = 3 # number of hidden layers neurons

output\_neurons = 1 # number of neurons at output layer

# weight and bias initialization

wh = np.random.uniform(size=(inputlayer\_neurons, hiddenlayer\_neurons)) # weight of the link from input node to hidden node

bh = np.random.uniform(size=(1, hiddenlayer\_neurons)) # bias of the link from input node to hidden node

wout = np.random.uniform(size=(hiddenlayer\_neurons, output\_neurons)) # weight of the link from hidden node to output node

bout = np.random.uniform(size=(1, output\_neurons)) # bias of the link from hidden node to output node

# draws a random range of numbers uniformly of dim x\*y

for i in range(epoch):

# Forward Propogation

hinp1 = np.dot(X, wh)

hinp = hinp1 + bh

hlayer\_act = sigmoid(hinp)

outinp1 = np.dot(hlayer\_act, wout)

outinp = outinp1 + bout

output = sigmoid(outinp)

# Backpropagation

EO = y - output

outgrad = derivatives\_sigmoid(output)

d\_output = EO \* outgrad

EH = d\_output.dot(wout.T)

# how much hidden layer weights contributed to error

hiddengrad = derivatives\_sigmoid(hlayer\_act)

d\_hiddenlayer = EH \* hiddengrad

# dotproduct of nextlayererror and currentlayerop

wout += hlayer\_act.T.dot(d\_output) \* lr

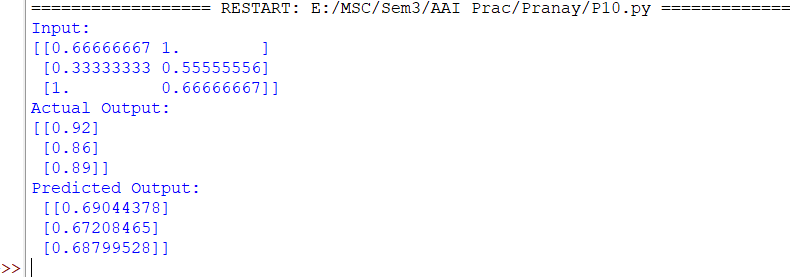
wh += X.T.dot(d\_hiddenlayer) \* lr

print("Input: \n" + str(X))

print("Actual Output: \n" + str(y))

print("Predicted Output: \n", output)

Output:



[Practical 1 3](#_TOC_250020)

* 1. [Load a CSV dataset. Handle missing values, inconsistent formatting, and outliers 3](#_TOC_250019)
  2. [Load a dataset, calculate descriptive summary statistics, create visualizations using different graphs, and identify potential features and target variables 7](#_TOC_250018)
  3. [Create or Explore datasets to use all pre-processing routines like label encoding, scaling, and binarization. 13](#_TOC_250017)

[Practical 2 15](#_TOC_250016)

* 1. [Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a. CSV file and generate the final specific hypothesis. (Create your dataset) ..](#_TOC_250015)

. 15

[Practical 3 16](#_TOC_250014)

* 1. [Simple Linear Regression Fit a linear regression model on a dataset. Interpret coefficients, make predictions, and evaluate performance using metrics like R- squared and MSE 16](#_TOC_250013)
  2. [Multiple Linear Regression Extend linear regression to multiple features. Handle feature selection and potential multicollinearity 19](#_TOC_250012)
  3. [Regularized Linear Models (Ridge, Lasso, ElasticNet) Implement regression variants like LASSO and Ridge on any generated dataset. 26](#_TOC_250011)

[Practical 4 29](#_TOC_250010)

* 1. [Logistic Regression Perform binary classification using logistic regression. Calculate accuracy, precision, recall, and understand the ROC curve. 29](#_TOC_250009)
  2. [Implement and demonstrate k-nearest Neighbor algorithm. Read the training data from a .CSV file and build the model to classify a test sample. Print both correct and wrong predictions 31](#_TOC_250008)
  3. [Build a decision tree classifier or regressor. Control hyperparameters like tree depth to avoid overfitting. Visualize the tree. 34](#_TOC_250007)
  4. [Implement a Support Vector Machine for any relevant dataset. 37](#_TOC_250006)
  5. [Train a random forest ensemble. Experiment with the number of trees and feature sampling. Compare performance to a single decision tree 39](#_TOC_250005)
  6. [Implement a gradient boosting machine (e.g., XGBoost). Tune hyperparameters and explore feature importance. 42](#_TOC_250004)

[Practical 5 45](#_TOC_250003)

* 1. [Implement and demonstrate the working of a Naive Bayesian classifier using a sample data set. Build the model to classify a test sample. 45](#_TOC_250002)
  2. [Implement Hidden Markov Models using hmmlearn 47](#_TOC_250001)

[Practical 6 50](#_TOC_250000)

* 1. Implement Bayesian Linear Regression to explore prior and posterior distribution 50
  2. Implement Gaussian Mixture Models for density estimation and unsupervised clustering 52

Practical 7 54

* 1. Implement cross-validation techniques (k-fold, stratified, etc.) for robust model evaluation 54
  2. Systematically explore combinations of hyperparameters to optimize model performance.(use grid and randomized search) 55

Practical 8 57

* 1. Implement Bayesian Learning using inferences 57

Practical 9 58

* 1. Set up a generator network to produce samples and a discriminator network to distinguish between real and generated data. (Use a simple small dataset) 58

Practical 10 62

* 1. Develop an API to deploy your model and perform predictions 62

# Practical 1

## Load a CSV dataset. Handle missing values, inconsistent formatting, and outliers.

**import** seaborn **as** sns

**import** pandas **as** pd

**import** matplotlib.pyplot **as** plt

**import** numpy **as** np

plt.rcParams["figure.figsize"] = [8,6] sns.set(style="darkgrid")

df = sns.load\_dataset('titanic') print(df.head())

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| \ | survived | pclass | sex | age | sibsp | parch | fare | embarked | class |
| 0 | 0 | 3 | male | 22.0 | 1 | 0 | 7.2500 | S | Third |
| 1 | 1 | 1 | female | 38.0 | 1 | 0 | 71.2833 | C | First |
| 2 | 1 | 3 | female | 26.0 | 0 | 0 | 7.9250 | S | Third |
| 3 | 1 | 1 | female | 35.0 | 1 | 0 | 53.1000 | S | First |
| 4 | 0 | 3 | male | 35.0 | 0 | 0 | 8.0500 | S | Third |

who adult\_male deck embark\_town alive alone

1. man True NaN Southampton no False
2. woman False C Cherbourg yes False
3. woman False NaN Southampton yes True
4. woman False C Southampton yes False
5. man True NaN Southampton no True print(df.isnull().sum())

|  |  |
| --- | --- |
| survived | 0 |
| pclass | 0 |
| sex | 0 |
| age | 177 |
| sibsp | 0 |
| parch | 0 |
| fare | 0 |
| embarked | 2 |
| class | 0 |
| who | 0 |
| adult\_male | 0 |

alone 0

dtype: int64

df = df[["age", "embarked"]] print(df.head())

age embarked

0 22.0 S

1 38.0 C

2 26.0 S

3 35.0 S

4 35.0 S

df.loc[:, 'age'] = df.age.fillna(df.age.median()) df = df.dropna(subset=["embarked"])

print(df.head(20)) age embarked

0 22.0 S

1 38.0 C

2 26.0 S

3 35.0 S

4 35.0 S

5 28.0 Q

6 54.0 S

7 2.0 S

8 27.0 S

9 14.0 C

10 4.0 S

11 58.0 S

12 20.0 S

13 39.0 S

14 14.0 S

15 55.0 S

16 2.0 Q

17 28.0 S

18 31.0 S

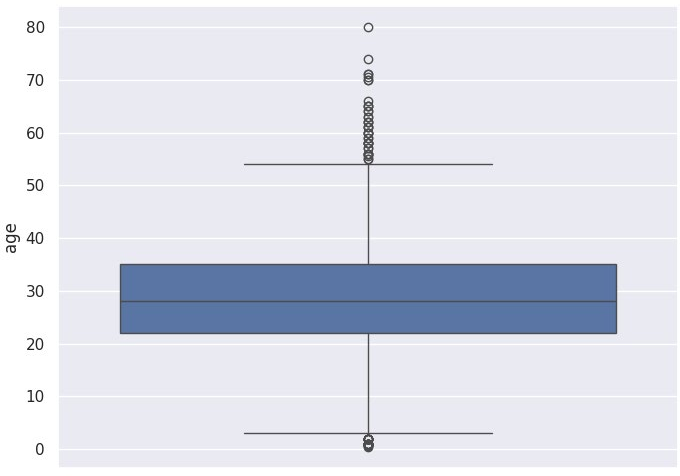
19 28.0 C

df.loc[:, 'embarked'] = df.embarked.str.upper() print(df.embarked.unique())

['S' 'C' 'Q']

sns.boxplot(data=df.age)

<Axes: ylabel='age'>



Q1 = df.age.quantile(0.25) Q3 = df.age.quantile(0.75)

IQR = Q3 - Q1

lower\_bound = Q1 - 1.5 \* IQR upper\_bound = Q3 + 1.5 \* IQR

df = df[(df.age >= lower\_bound) & (df.age <= upper\_bound)] print(df.head())

age embarked

0 22.0 S

1 38.0 C

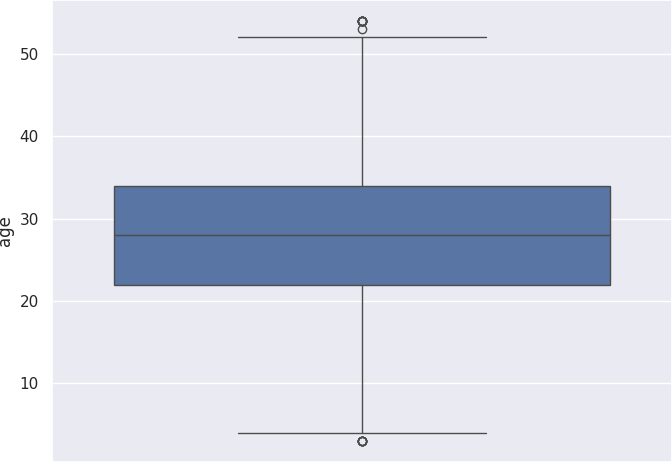
2 26.0 S

3 35.0 S

4 35.0 S

sns.boxplot(data=df.age)

<Axes: ylabel='age'>



## Load a dataset, calculate descriptive summary statistics, create visualizations using different graphs, and identify potential features and target variables

**import** pandas **as** pd **import** numpy **as** np **import** seaborn **as** sns

**import** matplotlib.pyplot **as** plt

plt.rcParams["figure.figsize"] = [8,6] sns.set(style="darkgrid")

df = sns.load\_dataset('iris') print(df.head())

sepal\_length sepal\_width petal\_length petal\_width species

0 5.1 3.5 1.4 0.2 setosa

1 4.9 3.0 1.4 0.2 setosa

2 4.7 3.2 1.3 0.2 setosa

3 4.6 3.1 1.5 0.2 setosa

4 5.0 3.6 1.4 0.2 setosa

summary\_statistics = df.describe() print(summary\_statistics)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | sepal\_length | sepal\_width | petal\_length | petal\_width |
| count | 150.000000 | 150.000000 | 150.000000 | 150.000000 |
| mean | 5.843333 | 3.057333 | 3.758000 | 1.199333 |
| std | 0.828066 | 0.435866 | 1.765298 | 0.762238 |
| min | 4.300000 | 2.000000 | 1.000000 | 0.100000 |
| 25% | 5.100000 | 2.800000 | 1.600000 | 0.300000 |
| 50% | 5.800000 | 3.000000 | 4.350000 | 1.300000 |
| 75% | 6.400000 | 3.300000 | 5.100000 | 1.800000 |
| max | 7.900000 | 4.400000 | 6.900000 | 2.500000 |

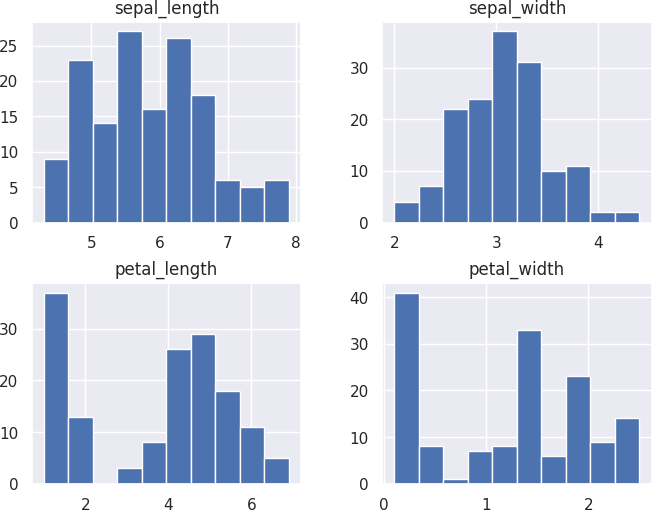
*#Univariate Visualizations*

df.hist()

array([[<Axes: title={'center': 'sepal\_length'}>,

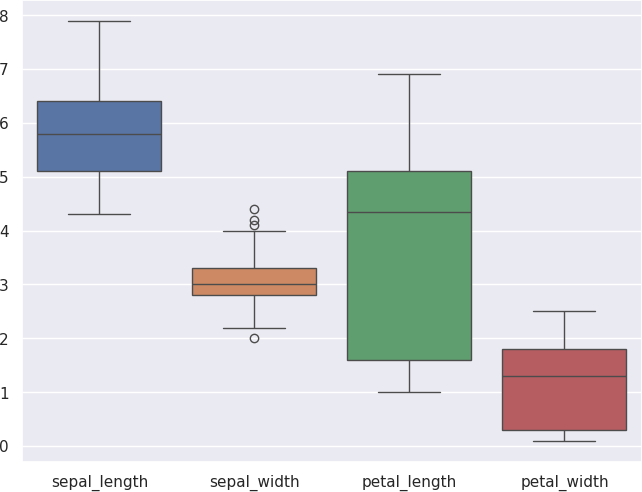
<Axes: title={'center': 'sepal\_width'}>], [<Axes: title={'center': 'petal\_length'}>,

<Axes: title={'center': 'petal\_width'}>]], dtype=object)



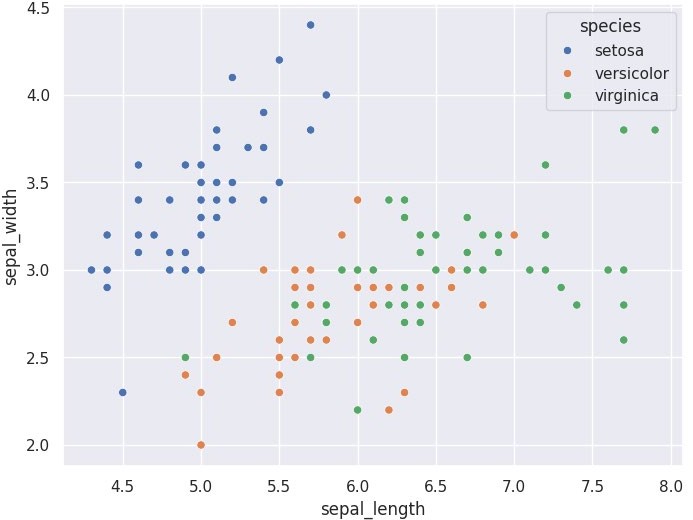
sns.boxplot(data=df)

<Axes: >



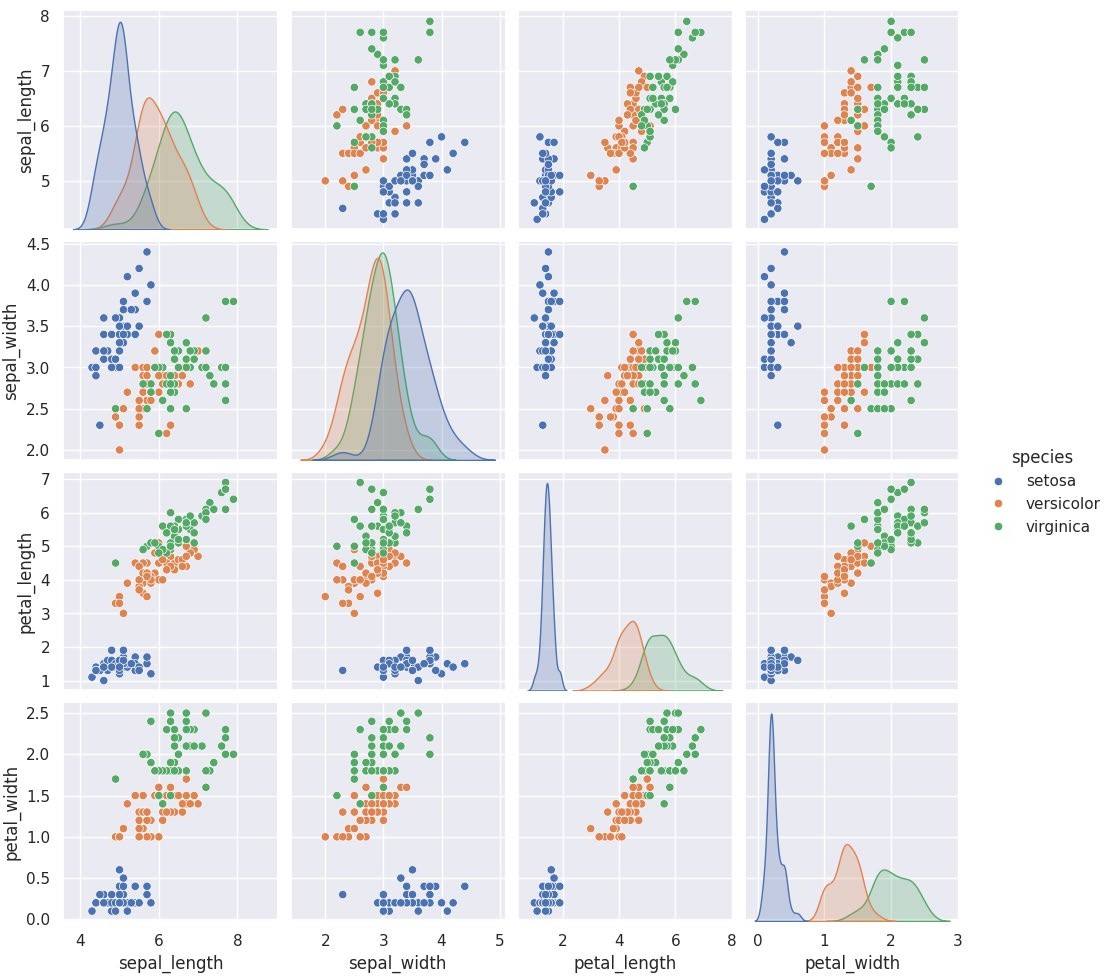
sns.scatterplot(data=df, x='sepal\_length', y='sepal\_width', hue='species')

<Axes: xlabel='sepal\_length', ylabel='sepal\_width'>



sns.pairplot(df, hue="species")

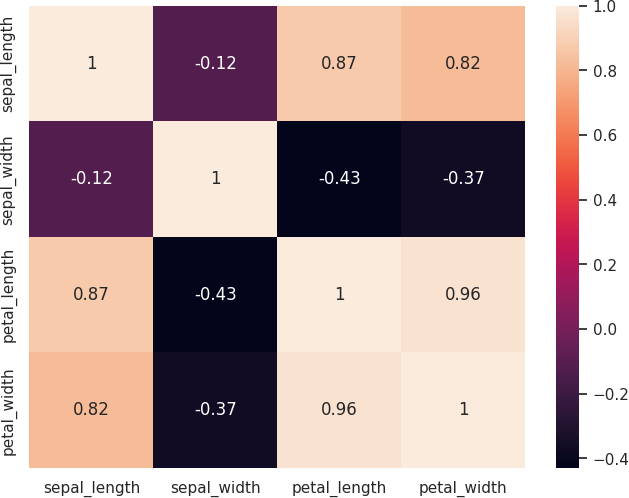
<seaborn.axisgrid.PairGrid at 0x7fc52c528610>



*#Correleation*

numeric\_df = df.select\_dtypes(include=[np.number]) correlation\_matrix = numeric\_df.corr() sns.heatmap(correlation\_matrix, annot=True)

<Axes: >



potential\_features = df.select\_dtypes(include=[np.number]).columns.tolist(

)

target\_variable = 'species'

print("Potential Features: ", potential\_features) print("Target Variable: ", target\_variable)

Potential Features: ['sepal\_length', 'sepal\_width', 'petal\_length', 'peta l\_width']

Target Variable: species

## Create or Explore datasets to use all pre-processing routines like label encoding, scaling, and binarization.

**import** seaborn **as** sns

**import** pandas **as** pd

**from** sklearn.preprocessing **import** LabelEncoder **from** sklearn.preprocessing **import** StandardScaler **from** sklearn.preprocessing **import** Binarizer

df = sns.load\_dataset('titanic') print(df.head())

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| \ | survived | pclass | sex | age | sibsp | parch | fare | embarked | class |
| 0 | 0 | 3 | male | 22.0 | 1 | 0 | 7.2500 | S | Third |
| 1 | 1 | 1 | female | 38.0 | 1 | 0 | 71.2833 | C | First |
| 2 | 1 | 3 | female | 26.0 | 0 | 0 | 7.9250 | S | Third |
| 3 | 1 | 1 | female | 35.0 | 1 | 0 | 53.1000 | S | First |
| 4 | 0 | 3 | male | 35.0 | 0 | 0 | 8.0500 | S | Third |

who adult\_male deck embark\_town alive alone

1. man True NaN Southampton no False
2. woman False C Cherbourg yes False
3. woman False NaN Southampton yes True
4. woman False C Southampton yes False
5. man True NaN Southampton no True

label\_encoder = LabelEncoder()

df['sex'] = label\_encoder.fit\_transform(df['sex']) print(df.sex.head())

|  |  |
| --- | --- |
| 0 | 1 |
| 1 | 0 |
| 2 | 0 |
| 3 | 0 |
| 4 | 1 |

Name: sex, dtype: int64 scaler = StandardScaler()

df[['age', 'fare']] = scaler.fit\_transform(df[['age', 'fare']]) print(df[['age', 'fare']].head())

|  |  |
| --- | --- |
| age | fare |
| 0 -0.530377 | -0.502445 |
| 1 0.571831 | 0.786845 |
| 2 -0.254825 | -0.488854 |

binarizer = Binarizer(threshold=0)

df[['fare']] = binarizer.fit\_transform(df[['fare']]) print(df.fare.head())

|  |  |
| --- | --- |
| 0 | 0.0 |
| 1 | 1.0 |
| 2 | 0.0 |
| 3 | 1.0 |
| 4 | 0.0 |

Name: fare, dtype: float64

# Practical 2

## Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a. CSV file and generate the final specific hypothesis. (Create your dataset)

**import** pandas **as** pd

data = {

'Material': ['Plastic', 'Metal', 'Glass', 'Metal'],

'Color': ['White', 'Silver', 'Green', 'Grey'],

'Size': ['Small', 'Large', 'Small', 'Large'],

'Recyclable': ['Yes', 'Yes', 'Yes', 'No'],

'E-Waste': ['No', 'Yes', 'No', 'Yes']

}

df = pd.DataFrame(data) df.to\_csv('training\_data.csv', index=False)

data = pd.read\_csv('training\_data.csv')

X = data.iloc[:, :-1]

y = data.iloc[:, -1] hypothesis = ['0'] \* X.shape[1]

**for** i **in** range(len(X)):

**if** y[i] == 'Yes':

**for** j **in** range(X.shape[1]):

**if** hypothesis[j] == '0': hypothesis[j] = X.iloc[i, j]

**elif** hypothesis[j] != X.iloc[i, j]: hypothesis[j] = '?'

print(hypothesis)

['Metal', '?', 'Large', '?']

# Practical 3

## Simple Linear Regression Fit a linear regression model on a dataset. Interpret coefficients, make predictions, and evaluate performance using metrics like R-squared and MSE

**import** numpy **as** np

**import** pandas **as** pd

**import** matplotlib.pyplot **as** plt

**import** seaborn **as** sns

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

**from** sklearn.linear\_model **import** LinearRegression

**from** sklearn.metrics **import** mean\_squared\_error, r2\_score

np.random.seed(42)

X = 2.5 \* np.random.rand(100, 1)

y = 5 + 2 \* X + np.random.randn(100, 1)

data = pd.DataFrame({'Feature': X.flatten(), 'Target': y.flatten()}) print(data.head())

Feature Target

0 0.936350 6.959748

1 2.376786 9.454564

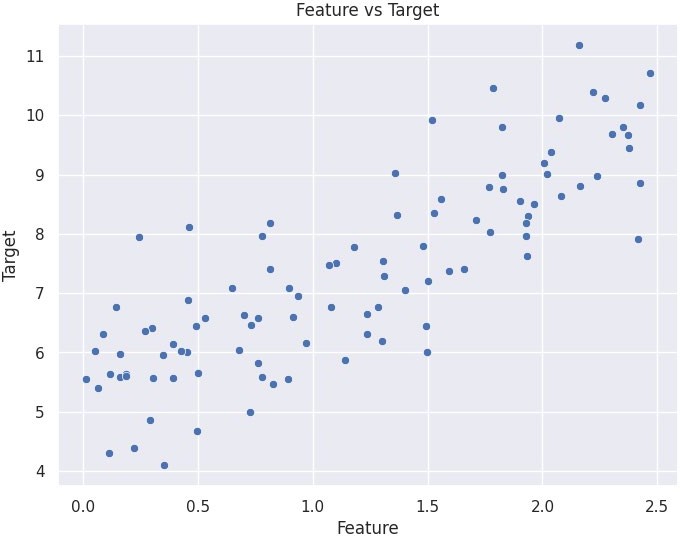
2 1.829985 8.751730

3 1.496646 6.005724

4 0.390047 5.560421

plt.figure(figsize=(8, 6)) sns.scatterplot(x='Feature', y='Target', data=data) plt.title('Feature vs Target')

plt.show()



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

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

print(f"Intercept: {model.intercept\_[0]:.2f}") print(f"Coefficient: {model.coef\_[0][0]:.2f}")

Intercept: 5.14

Coefficient: 1.84

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

pred\_df = pd.DataFrame({'Actual': y\_test.flatten(), 'Predicted': y\_pred.fl atten()})

print(pred\_df.head())

Actual Predicted

0 5.974345 5.435196

1 8.970661 9.257909

2 7.624273 8.694195

3 7.403224 8.189620

4 7.084932 6.332951

mse = mean\_squared\_error(y\_test, y\_pred) print(f"Mean Squared Error (MSE): {mse:.2f}")

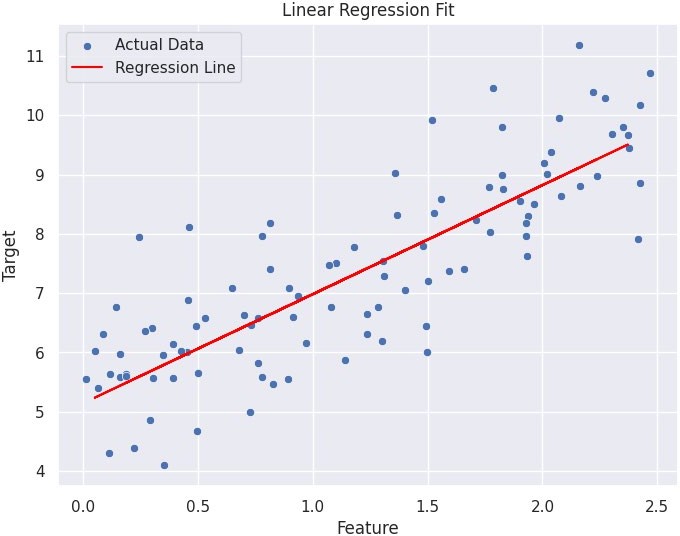
r2 = r2\_score(y\_test, y\_pred) print(f"R-squared: {r2:.2f}")

Mean Squared Error (MSE): 0.65 R-squared: 0.73

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

sns.scatterplot(x='Feature', y='Target', data=data, label='Actual Data') plt.plot(X\_test, y\_pred, color='red', label='Regression Line') plt.title('Linear Regression Fit')

plt.legend() plt.show()



## Multiple Linear Regression Extend linear regression to multiple features. Handle feature selection and potential multicollinearity.

**import** seaborn **as** sns **import** pandas **as** pd **import** numpy **as** np

**import** matplotlib.pyplot **as** plt

**import** statsmodels.api **as** sm

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

**from** sklearn.linear\_model **import** LinearRegression

**from** sklearn.metrics **import** mean\_squared\_error, r2\_score

**from** statsmodels.stats.outliers\_influence **import** variance\_inflation\_factor df = sns.load\_dataset('diamonds')

print("Missing values in the dataset:") print(df.isnull().sum())

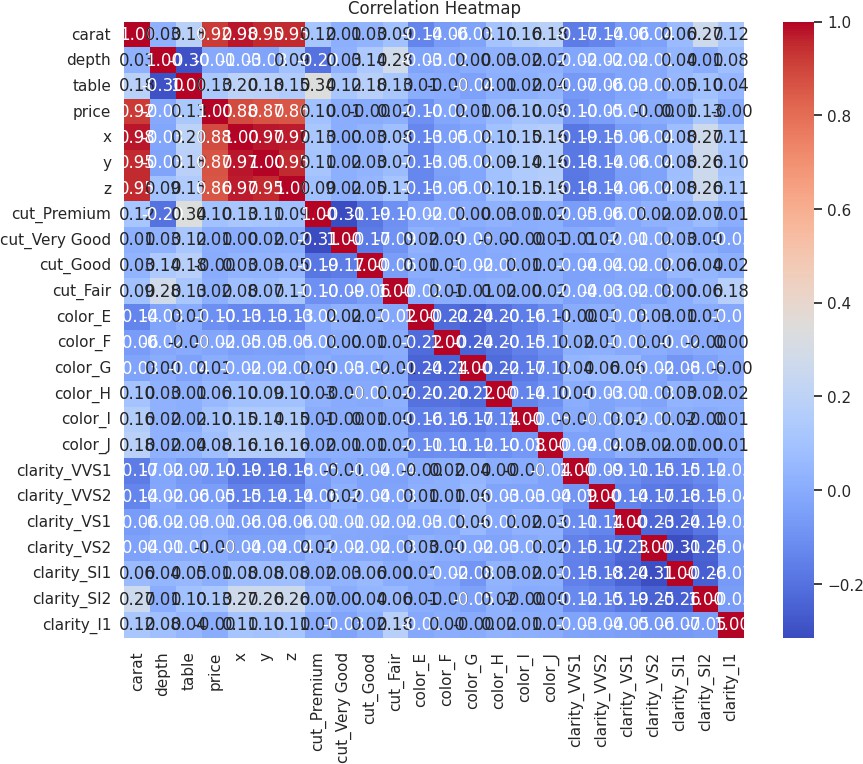
df = pd.get\_dummies(df, drop\_first=True)

|  |  |
| --- | --- |
| Missing | values in the dataset: |
| carat | 0 |
| cut | 0 |
| color | 0 |
| clarity | 0 |
| depth | 0 |
| table | 0 |
| price | 0 |
| x | 0 |
| y | 0 |
| z | 0 |
| dtype: | int64 |

plt.figure(figsize=(10, 8))

sns.heatmap(df.corr(), annot=True, cmap='coolwarm', fmt='.2f') plt.title("Correlation Heatmap")

plt.show()



X = df[['carat', 'depth', 'table', 'x', 'y', 'z', 'cut\_Premium', 'cut\_Good', 'cut\_Very Good', 'color\_E', 'color\_F', 'clarity\_VVS2', 'clarity\_VS1']]

y = df['price']

y = y.astype(float)

X\_with\_constant = sm.add\_constant(X)

X\_with\_constant = X\_with\_constant.astype(int) vif = pd.DataFrame()

vif['Features'] = X.columns

vif['VIF'] = [variance\_inflation\_factor(X\_with\_constant.values, i+1) **for** i

**in** range(len(X.columns))] print(vif)

|  |  |  |
| --- | --- | --- |
|  | Features | VIF |
| 0 | carat | 3.613538 |
| 1 | depth | 1.211270 |

2 table 1.530672

3 x 19.224267

4 y 15.677513

5 z 5.789510

6 cut\_Premium 1.548643

7 cut\_Good 1.295429

8 cut\_Very Good 1.346362

9 color\_E 1.079848

10 color\_F 1.060367

11 clarity\_VVS2 1.051655

12 clarity\_VS1 1.031049

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, r andom\_state=42)

print("Data types of X\_train:") print(X\_train.dtypes)

print("Data type of y\_train:", y\_train.dtype)

Data types of X\_train:

carat float64

depth float64

table float64

1. float64
2. float64
3. float64

cut\_Premium bool

cut\_Good bool

cut\_Very Good bool

color\_E bool

color\_F bool

clarity\_VVS2 bool

clarity\_VS1 bool dtype: object

Data type of y\_train: float64

X\_train = X\_train.astype(float) y\_train = y\_train.astype(float)

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

print("Intercept:", model.intercept\_) print("Coefficients:", model.coef\_)

Intercept: 17520.480548853404

Coefficients: [ 1.06799558e+04 -1.74848962e+02 -8.87411825e+01 -1.17618393 e+03

3.03543071e+01 8.16330490e+00 -3.94552416e+01 -1.98436785e+02

-1.87877044e+01 4.36586414e+02 4.74099129e+02 1.02842658e+03 6.62232418e+02]

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

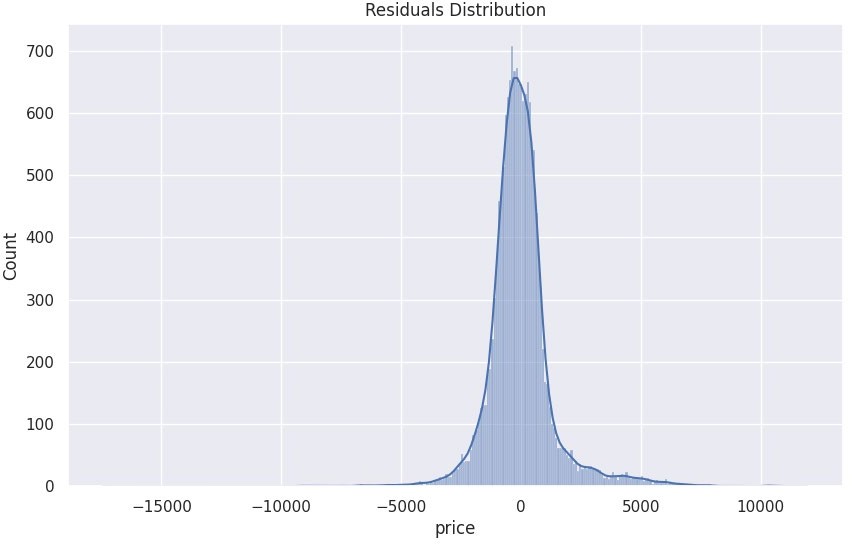
mse = mean\_squared\_error(y\_test, y\_pred) r2 = r2\_score(y\_test, y\_pred)

print("Mean Squared Error:", mse) print("R-squared:", r2)

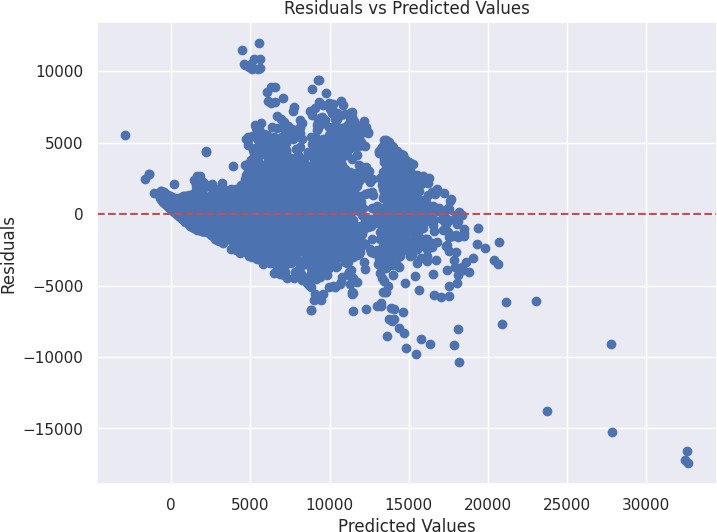
residuals = y\_test - y\_pred

Mean Squared Error: 2018911.748442661 R-squared: 0.8705490836162249

plt.figure(figsize=(10, 6)) sns.histplot(residuals, kde=True) plt.title("Residuals Distribution") plt.show()



plt.scatter(y\_pred, residuals) plt.axhline(y=0, color='r', linestyle='--') plt.xlabel("Predicted Values") plt.ylabel("Residuals") plt.title("Residuals vs Predicted Values") plt.show()



X\_train\_sm = sm.add\_constant(X\_train) ols\_model = sm.OLS(y\_train, X\_train\_sm).fit() print(ols\_model.summary())

OLS Regression Results

==========================================================================

====

Dep. Variable: price R-squared: 0

.870

Model: OLS Adj. R-squared: 0

.869

Method: Least Squares F-statistic: 1.935

e+04

Date: Thu, 24 Oct 2024 Prob (F-statistic): 0.00

Time: 07:44:35 Log-Likelihood: -3.2835 e+05

No. Observations: 37758 AIC: 6.567

e+05

Df Residuals: 37744 BIC: 6.569

e+05

Df Model: 13

Covariance Type: nonrobust

==========================================================================

coef std err t P>|t| [0.025

0.975]

const 1.752e+04 537.976 32.567 0.000 1.65e+04 1

.86e+04

carat 1.068e+04 72.935 146.431 0.000 1.05e+04 1

.08e+04

depth -174.8490 6.336 -27.595 0.000 -187.268 -

162.430

table -88.7412 4.144 -21.414 0.000 -96.864

-80.619

x -1176.1839 46.936 -25.059 0.000 -1268.179 -1

084.188

y 30.3543 27.897 1.088 0.277 -24.325

85.034

z 8.1633 43.889 0.186 0.852 -77.861

94.188

cut\_Premium -39.4552 21.233 -1.858 0.063 -81.073

2.163

cut\_Good -198.4368 29.625 -6.698 0.000 -256.502 -

140.372

cut\_Very Good -18.7877 20.712 -0.907 0.364 -59.383

21.808

color\_E 436.5864 20.108 21.712 0.000 397.174

475.999

color\_F 474.0991 20.084 23.606 0.000 434.735

513.463

clarity\_VVS2 1028.4266 26.287 39.123 0.000 976.903 1

079.950

clarity\_VS1 662.2324 21.075 31.423 0.000 620.925

703.540

==========================================================================

====

Omnibus: 9105.593 Durbin-Watson: 1

.992

Prob(Omnibus): 0.000 Jarque-Bera (JB): 327369

.326

Skew: 0.453 Prob(JB):

0.00

Kurtosis: 17.397 Cond. No. 6.14

e+03

==========================================================================

====

Notes:

1. Standard Errors assume that the covariance matrix of the errors is cor rectly specified.

## Regularized Linear Models (Ridge, Lasso, ElasticNet) Implement regression variants like LASSO and Ridge on any generated dataset.

**import** numpy **as** np

**import** pandas **as** pd

**import** matplotlib.pyplot **as** plt

**from** sklearn.datasets **import** make\_regression

**from** sklearn.model\_selection **import** train\_test\_split **from** sklearn.linear\_model **import** Ridge, Lasso, ElasticNet **from** sklearn.metrics **import** mean\_squared\_error

X, y, coef = make\_regression(n\_samples=100, n\_features=10, noise=0.1, coef

=True, random\_state=42)

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

ridge\_model = Ridge(alpha=1.0) ridge\_model.fit(X\_train, y\_train) ridge\_pred = ridge\_model.predict(X\_test)

lasso\_model = Lasso(alpha=0.1) lasso\_model.fit(X\_train, y\_train) lasso\_pred = lasso\_model.predict(X\_test)

elastic\_model = ElasticNet(alpha=0.1, l1\_ratio=0.5) elastic\_model.fit(X\_train, y\_train)

elastic\_pred = elastic\_model.predict(X\_test)

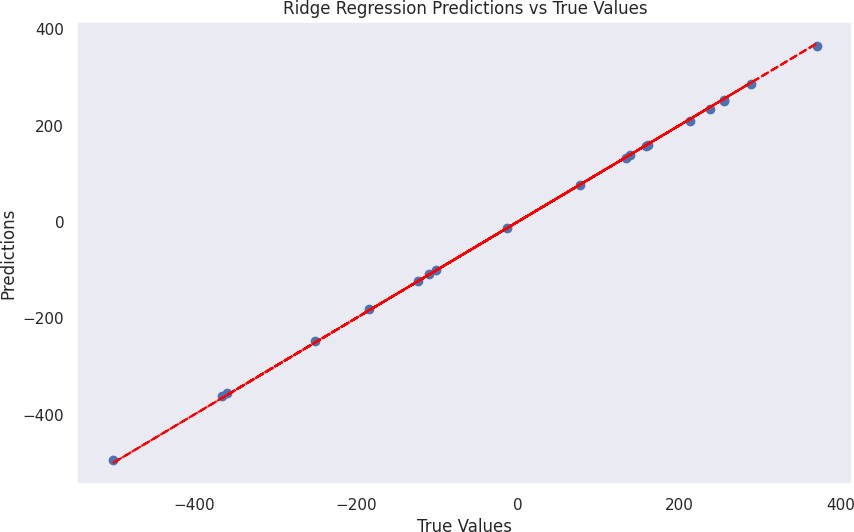
**def** plot\_results(y\_test, predictions, model\_name): plt.figure(figsize=(10, 6)) plt.scatter(y\_test, predictions)

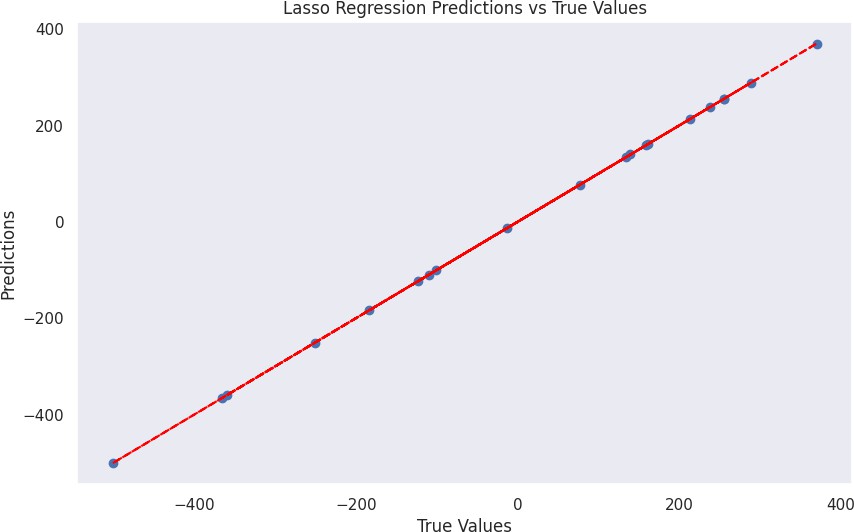
plt.plot(y\_test, y\_test, color='red', linestyle='--') *# y=x line* plt.title(f'{model\_name} Predictions vs True Values') plt.xlabel('True Values')

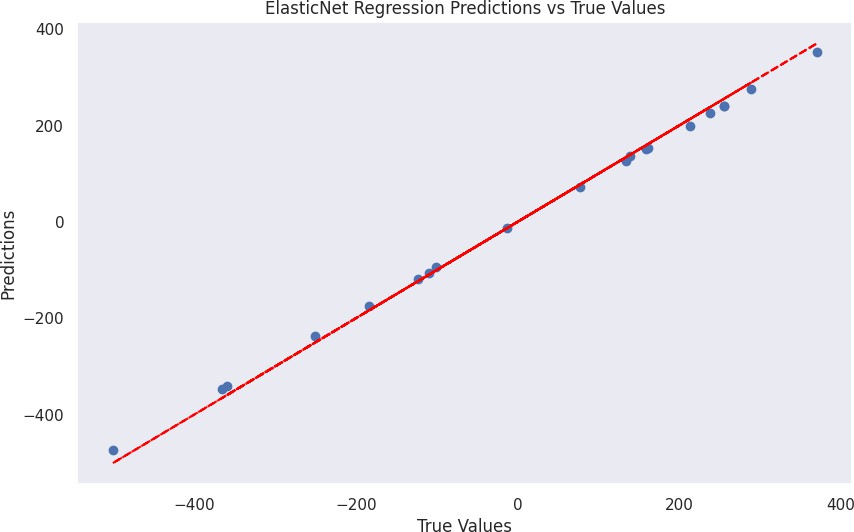
plt.ylabel('Predictions') plt.grid()

plt.show()

plot\_results(y\_test, ridge\_pred, 'Ridge Regression') plot\_results(y\_test, lasso\_pred, 'Lasso Regression') plot\_results(y\_test, elastic\_pred, 'ElasticNet Regression')







print("Mean Squared Error (MSE):")

print(f"Ridge: {mean\_squared\_error(y\_test, ridge\_pred):.2f}") print(f"Lasso: {mean\_squared\_error(y\_test, lasso\_pred):.2f}") print(f"ElasticNet: {mean\_squared\_error(y\_test, elastic\_pred):.2f}")

Mean Squared Error (MSE): Ridge: 11.84

Lasso: 0.18

ElasticNet: 176.03

# Practical 4

## Logistic Regression Perform binary classification using logistic regression. Calculate accuracy, precision, recall, and understand the ROC curve.

**import** numpy **as** np

**import** matplotlib.pyplot **as** plt

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

**from** sklearn.linear\_model **import** LogisticRegression

**from** sklearn.metrics **import** accuracy\_score, precision\_score, recall\_score, roc\_curve, auc

**from** sklearn.datasets **import** make\_classification

X, y = make\_classification(n\_samples=1000, n\_features=10, n\_classes=2, ran dom\_state=42)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, r andom\_state=42)

logistic\_reg\_model = LogisticRegression() logistic\_reg\_model.fit(X\_train, y\_train) y\_pred = logistic\_reg\_model.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred) precision = precision\_score(y\_test, y\_pred) recall = recall\_score(y\_test, y\_pred)

print(f"Accuracy: {accuracy:.2f}") print(f"Precision: {precision:.2f}") print(f"Recall: {recall:.2f}")

Accuracy: 0.85

Precision: 0.89

Recall: 0.82

y\_prob = logistic\_reg\_model.predict\_proba(X\_test)[:, 1] fpr, tpr, thresholds = roc\_curve(y\_test, y\_prob) roc\_auc = auc(fpr, tpr)

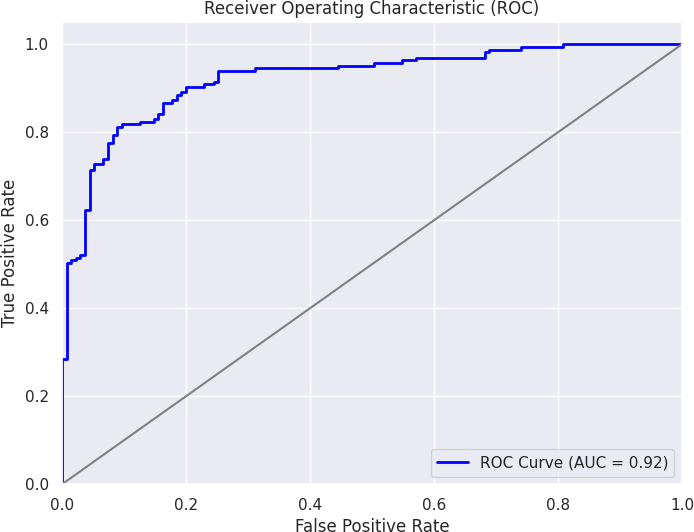
plt.figure()

plt.plot(fpr, tpr, color='blue', lw=2, label=f"ROC Curve (AUC = {roc\_auc:. 2f})")

plt.plot([0, 1], [0, 1], color='gray', linestyle='-') *# Diagonal line for random classifier*

plt.xlim([0.0, 1.0])

plt.legend(loc='lower right') plt.show()



## Implement and demonstrate k-nearest Neighbor algorithm. Read the training data from a .CSV file and build the model to classify a test sample. Print both correct and wrong predictions.

**from** sklearn **import** datasets

**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

iris = datasets.load\_iris()

X = iris.data y = iris.target

df = pd.DataFrame(data=X, columns=iris.feature\_names) df['target'] = y

print(df.head())

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ) | sepal  \ | length | (cm) | sepal | width | (cm) | petal | length | (cm) | petal | width | (cm |
| 0 |  |  | 5.1 |  |  | 3.5 |  |  | 1.4 |  |  | 0. |
| 2 |  |  |  |  |  |  |  |  |  |  |  |  |
| 1 |  |  | 4.9 |  |  | 3.0 |  |  | 1.4 |  |  | 0. |
| 2 |  |  |  |  |  |  |  |  |  |  |  |  |
| 2 |  |  | 4.7 |  |  | 3.2 |  |  | 1.3 |  |  | 0. |
| 2 |  |  |  |  |  |  |  |  |  |  |  |  |
| 3 |  |  | 4.6 |  |  | 3.1 |  |  | 1.5 |  |  | 0. |
| 2 |  |  |  |  |  |  |  |  |  |  |  |  |
| 4 |  |  | 5.0 |  |  | 3.6 |  |  | 1.4 |  |  | 0. |
| 2 |  |  | | | | | | | | | | |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ) | sepal  \ | length | (cm) | sepal | width | (cm) | petal | length | (cm) | petal | width | (cm |
| 0 |  |  | 5.1 |  |  | 3.5 |  |  | 1.4 |  |  | 0. |
| 2 |  |  |  |  |  |  |  |  |  |  |  |  |
| 1 |  |  | 4.9 |  |  | 3.0 |  |  | 1.4 |  |  | 0. |
| 2 |  |  |  |  |  |  |  |  |  |  |  |  |
| 2 |  |  | 4.7 |  |  | 3.2 |  |  | 1.3 |  |  | 0. |
| 2 |  |  |  |  |  |  |  |  |  |  |  |  |
| 3 |  |  | 4.6 |  |  | 3.1 |  |  | 1.5 |  |  | 0. |
| 2 |  |  |  |  |  |  |  |  |  |  |  |  |
| 4 |  |  | 5.0 |  |  | 3.6 |  |  | 1.4 |  |  | 0. |
| 2 |  |  | | | | | | | | | | |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

X = df.iloc[:, :-1]

y = df.iloc[:, -1]

y = y.astype('category').cat.codes

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

k = 3

knn = KNeighborsClassifier(n\_neighbors=k) knn.fit(X\_train, y\_train)

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

**for** i **in** range(len(y\_test)):

print(f'Predicted: {iris.target\_names[y\_pred[i]]}, Actual: {iris.targe t\_names[y\_test.iloc[i]]}')

Predicted: versicolor, Actual: versicolor Predicted: setosa, Actual: setosa Predicted: virginica, Actual: virginica Predicted: versicolor, Actual: versicolor Predicted: versicolor, Actual: versicolor Predicted: setosa, Actual: setosa Predicted: versicolor, Actual: versicolor Predicted: virginica, Actual: virginica Predicted: versicolor, Actual: versicolor Predicted: versicolor, Actual: versicolor Predicted: virginica, Actual: virginica Predicted: setosa, Actual: setosa Predicted: setosa, Actual: setosa Predicted: setosa, Actual: setosa Predicted: setosa, Actual: setosa Predicted: versicolor, Actual: versicolor Predicted: virginica, Actual: virginica Predicted: versicolor, Actual: versicolor Predicted: versicolor, Actual: versicolor Predicted: virginica, Actual: virginica Predicted: setosa, Actual: setosa Predicted: virginica, Actual: virginica Predicted: setosa, Actual: setosa Predicted: virginica, Actual: virginica Predicted: virginica, Actual: virginica Predicted: virginica, Actual: virginica Predicted: virginica, Actual: virginica Predicted: virginica, Actual: virginica Predicted: setosa, Actual: setosa Predicted: setosa, Actual: setosa

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

print(classification\_report(y\_test, y\_pred, target\_names=iris.target\_names

))

Accuracy: 100.00%

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | precision | recall | f1-score | support |
| setosa | 1.00 | 1.00 | 1.00 | 10 |
| versicolor | 1.00 | 1.00 | 1.00 | 9 |
| virginica | 1.00 | 1.00 | 1.00 | 11 |
| accuracy |  |  | 1.00 | 30 |
| macro avg | 1.00 | 1.00 | 1.00 | 30 |
| weighted avg | 1.00 | 1.00 | 1.00 | 30 |

## Build a decision tree classifier or regressor. Control hyperparameters like tree depth to avoid overfitting. Visualize the tree.

**import** numpy **as** np

**import** pandas **as** pd

**import** matplotlib.pyplot **as** plt

**from** sklearn.datasets **import** load\_iris, fetch\_california\_housing

**from** sklearn.tree **import** DecisionTreeClassifier, DecisionTreeRegressor, pl ot\_tree

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

**from** sklearn.metrics **import** accuracy\_score, mean\_squared\_error

iris = load\_iris()

X = iris.data y = iris.target

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

clf = DecisionTreeClassifier(max\_depth=3, random\_state=42) clf.fit(X\_train, y\_train)

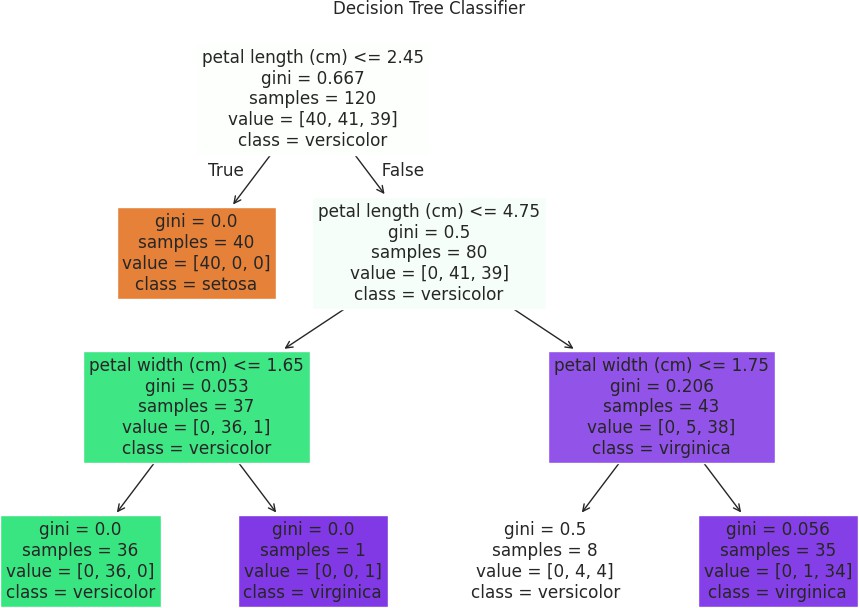
y\_pred = clf.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred) print(f'Accuracy: {accuracy:.2f}')

Accuracy: 1.00 plt.figure(figsize=(12,8))

plot\_tree(clf, filled=True, feature\_names=iris.feature\_names, class\_names= iris.target\_names)

plt.title("Decision Tree Classifier") plt.show()



housing = fetch\_california\_housing()

X\_housing = housing.data y\_housing = housing.target

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

reg = DecisionTreeRegressor(max\_depth=3, random\_state=42) reg.fit(X\_train\_housing, y\_train\_housing)

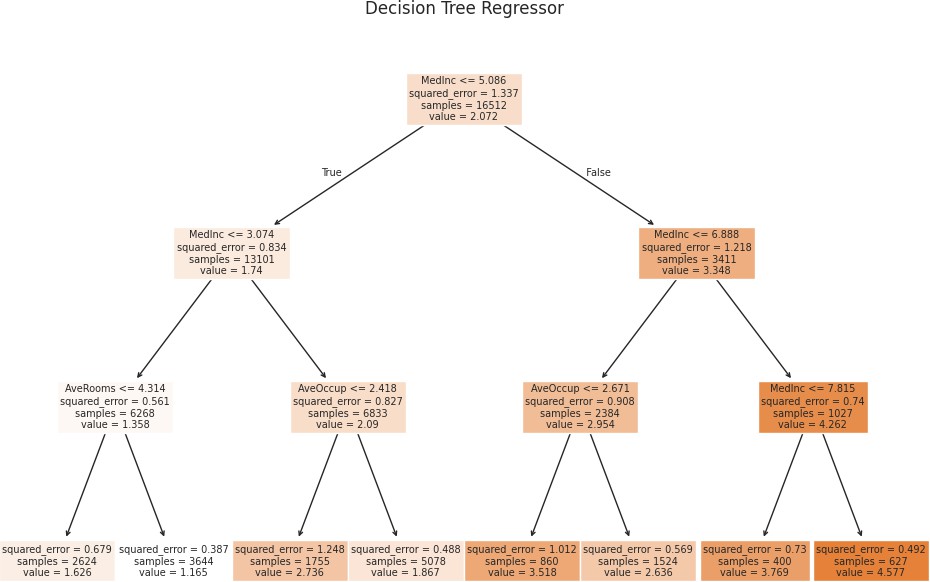
y\_pred\_housing = reg.predict(X\_test\_housing)

mse = mean\_squared\_error(y\_test\_housing, y\_pred\_housing) print(f'Mean Squared Error: {mse:.2f}')

Mean Squared Error: 0.64 plt.figure(figsize=(12,8))

plot\_tree(reg, filled=True, feature\_names=housing.feature\_names) plt.title("Decision Tree Regressor")

plt.show()



## Implement a Support Vector Machine for any relevant dataset.

**import** numpy **as** np

**import** pandas **as** pd

**import** matplotlib.pyplot **as** plt

**from** sklearn **import** datasets

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

**from** sklearn.svm **import** SVC

**from** sklearn.metrics **import** accuracy\_score, confusion\_matrix, classificati on\_report

**import** seaborn **as** sns

iris = datasets.load\_iris()

X = iris.data y = iris.target

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, r andom\_state=42)

svm\_classifier = SVC(kernel='linear', random\_state=42) svm\_classifier.fit(X\_train, y\_train)

y\_pred = svm\_classifier.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred) print(f"Accuracy: {accuracy:.2f}")

conf\_matrix = confusion\_matrix(y\_test, y\_pred) print("\nConfusion Matrix:") print(conf\_matrix)

class\_report = classification\_report(y\_test, y\_pred, target\_names=iris.tar get\_names)

print("\nClassification Report:") print(class\_report)

Accuracy: 1.00 Confusion Matrix:

|  |  |  |
| --- | --- | --- |
| [[19 | 0 | 0] |
| [ 0 | 13 | 0] |
| [ 0 | 0 | 13]] |

Classification Report:

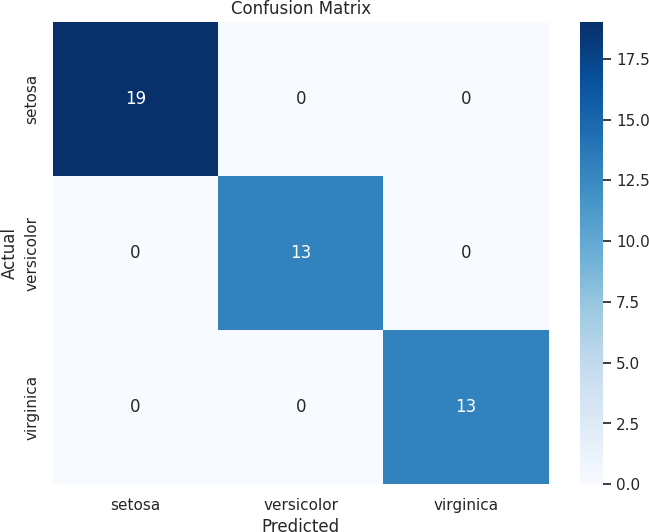
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| precision | | recall | f1-score | support |
| setosa | 1.00 | 1.00 | 1.00 | 19 |
| versicolor | 1.00 | 1.00 | 1.00 | 13 |
| virginica | 1.00 | 1.00 | 1.00 | 13 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| accuracy |  |  | 1.00 | 45 |
| macro avg | 1.00 | 1.00 | 1.00 | 45 |
| weighted avg | 1.00 | 1.00 | 1.00 | 45 |

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

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=ir is.target\_names, yticklabels=iris.target\_names)

plt.title('Confusion Matrix') plt.ylabel('Actual') plt.xlabel('Predicted') plt.show()



## Train a random forest ensemble. Experiment with the number of trees and feature sampling. Compare performance to a single decision tree.

**import** numpy **as** np

**import** pandas **as** pd

**import** matplotlib.pyplot **as** plt

**from** sklearn **import** datasets

**from** sklearn.model\_selection **import** train\_test\_split **from** sklearn.tree **import** DecisionTreeClassifier **from** sklearn.ensemble **import** RandomForestClassifier

**from** sklearn.metrics **import** accuracy\_score, confusion\_matrix, classificati on\_report

**import** seaborn **as** sns

iris = datasets.load\_iris()

X = iris.data y = iris.target

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, r andom\_state=42)

tree\_classifier = DecisionTreeClassifier(random\_state=42) tree\_classifier.fit(X\_train, y\_train)

y\_pred\_tree = tree\_classifier.predict(X\_test) accuracy\_tree = accuracy\_score(y\_test, y\_pred\_tree) print(f"Decision Tree Accuracy: {accuracy\_tree:.2f}\n")

Decision Tree Accuracy: 1.00

n\_trees = [1, 5, 10, 50, 100] accuracy\_forest = []

**for** n **in** n\_trees:

forest\_classifier = RandomForestClassifier(n\_estimators=n, random\_stat e=42)

forest\_classifier.fit(X\_train, y\_train) y\_pred\_forest = forest\_classifier.predict(X\_test) accuracy = accuracy\_score(y\_test, y\_pred\_forest) accuracy\_forest.append(accuracy)

print(f"Random Forest with {n} trees Accuracy: {accuracy:.2f}")

Random Forest with 1 trees Accuracy: 1.00 Random Forest with 5 trees Accuracy: 1.00 Random Forest with 10 trees Accuracy: 1.00 Random Forest with 50 trees Accuracy: 1.00 Random Forest with 100 trees Accuracy: 1.00

plt.figure(figsize=(10, 6))

plt.plot(n\_trees, accuracy\_forest, marker='o', label='Random Forest') plt.axhline(y=accuracy\_tree, color='r', linestyle='-', label='Single Decis ion Tree')

plt.title('Model Accuracy Comparison')

plt.xlabel('Number of Trees') plt.ylabel('Accuracy') plt.xticks(n\_trees) plt.legend()

plt.grid() plt.show()



best\_n = n\_trees[np.argmax(accuracy\_forest)] *# Get the best performing nu mber of trees*

best\_forest\_classifier = RandomForestClassifier(n\_estimators=best\_n, rando m\_state=42)

best\_forest\_classifier.fit(X\_train, y\_train) y\_pred\_best\_forest = best\_forest\_classifier.predict(X\_test)

conf\_matrix = confusion\_matrix(y\_test, y\_pred\_best\_forest) print("\nConfusion Matrix for Random Forest (best model):") print(conf\_matrix)

Class\_report = classification\_report(y\_test, y\_pred\_best\_forest, target\_na mes=iris.target\_names)

print("\nClassification Report for Random Forest (best model):") print(class\_report)

Confusion Matrix for Random Forest (best model):

|  |  |  |
| --- | --- | --- |
| [[19 | 0 | 0] |
| [ 0 | 13 | 0] |
| [ 0 | 0 | 13]] |

Classification Report for Random Forest (best model):

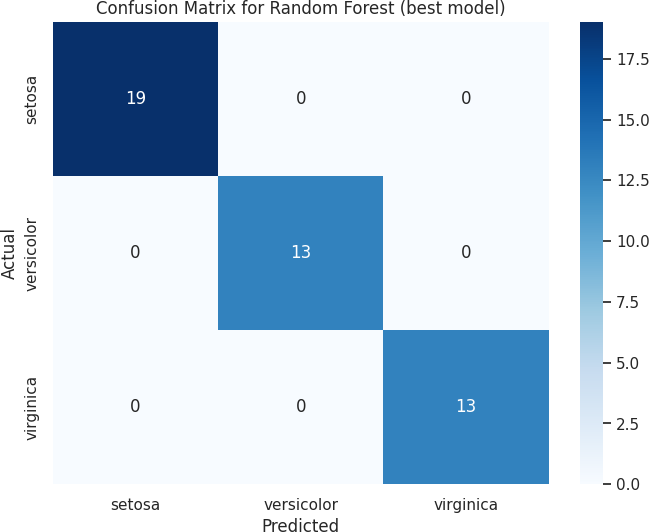
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | precision | recall | f1-score | support |
| setosa | 1.00 | 1.00 | 1.00 | 19 |
| versicolor | 1.00 | 1.00 | 1.00 | 13 |
| virginica | 1.00 | 1.00 | 1.00 | 13 |
| accuracy |  |  | 1.00 | 45 |
| macro avg | 1.00 | 1.00 | 1.00 | 45 |
| weighted avg | 1.00 | 1.00 | 1.00 | 45 |

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

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=ir is.target\_names, yticklabels=iris.target\_names)

plt.title('Confusion Matrix for Random Forest (best model)') plt.ylabel('Actual')

plt.xlabel('Predicted') plt.show()



## Implement a gradient boosting machine (e.g., XGBoost). Tune hyperparameters and explore feature importance.

**import** numpy **as** np

**import** pandas **as** pd

**import** matplotlib.pyplot **as** plt

**import** seaborn **as** sns

**from** sklearn **import** datasets

**from** sklearn.model\_selection **import** train\_test\_split, GridSearchCV

**from** xgboost **import** XGBClassifier

**from** sklearn.metrics **import** accuracy\_score, confusion\_matrix, classificati on\_report

iris = datasets.load\_iris()

X = iris.data y = iris.target

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, r andom\_state=42)

xgb\_model = XGBClassifier(use\_label\_encoder=False, eval\_metric='mlogloss') xgb\_model.fit(X\_train, y\_train)

param\_grid = {

'n\_estimators': [50, 100, 200],

'max\_depth': [3, 4, 5],

'learning\_rate': [0.01, 0.1, 0.2],

'subsample': [0.8, 1.0]

}

grid\_search = GridSearchCV(estimator=xgb\_model, param\_grid=param\_grid, sco ring='accuracy', cv=3, verbose=1, n\_jobs=-1)

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

best\_params = grid\_search.best\_params\_

print("Best parameters from GridSearch:", best\_params)

Fitting 3 folds for each of 54 candidates, totalling 162 fits

Best parameters from GridSearch: {'learning\_rate': 0.01, 'max\_depth': 4, ' n\_estimators': 100, 'subsample': 0.8}

best\_xgb\_model = grid\_search.best\_estimator\_ y\_pred = best\_xgb\_model.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred) print(f"\nXGBoost Accuracy: {accuracy:.2f}\n")

conf\_matrix = confusion\_matrix(y\_test, y\_pred) print("Confusion Matrix:")

print(conf\_matrix)

class\_report = classification\_report(y\_test, y\_pred, target\_names=iris.tar get\_names)

print("\nClassification Report:") print(class\_report)

XGBoost Accuracy: 1.00

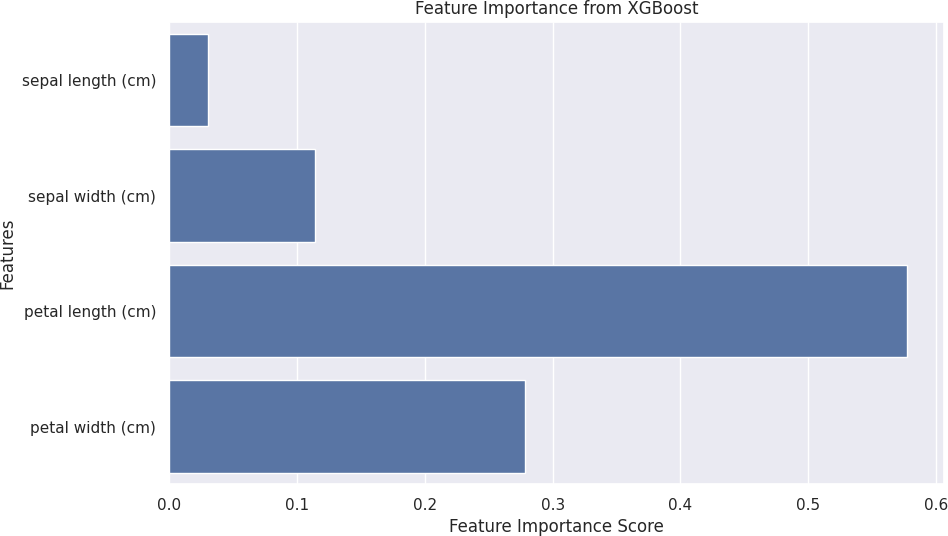
|  |  |  |
| --- | --- | --- |
| Confusion Matrix: | | |
| [[19 | 0 | 0] |
| [ 0 | 13 | 0] |
| [ 0 | 0 | 13]] |
| Classification Report: | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | precision | recall | f1-score | support |
| setosa | 1.00 | 1.00 | 1.00 | 19 |
| versicolor | 1.00 | 1.00 | 1.00 | 13 |
| virginica | 1.00 | 1.00 | 1.00 | 13 |
| accuracy |  |  | 1.00 | 45 |
| macro avg | 1.00 | 1.00 | 1.00 | 45 |
| weighted avg | 1.00 | 1.00 | 1.00 | 45 |

plt.figure(figsize=(10, 6)) sns.barplot(x=best\_xgb\_model.feature\_importances\_, y=iris.feature\_names) plt.title('Feature Importance from XGBoost')

plt.xlabel('Feature Importance Score') plt.ylabel('Features')

plt.show()



# Practical 5

## Implement and demonstrate the working of a Naive Bayesian classifier using a sample data set. Build the model to classify a test sample.

**import** numpy **as** np

**import** pandas **as** pd

**from** sklearn.datasets **import** load\_iris

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

**from** sklearn.naive\_bayes **import** GaussianNB

**from** sklearn.metrics **import** accuracy\_score, classification\_report, confusi on\_matrix

iris = load\_iris()

X = iris.data y = iris.target

df = pd.DataFrame(data=X, columns=iris.feature\_names) df['target'] = y

print(df.head())

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ) | sepal  \ | length | (cm) | sepal | width | (cm) | petal | length | (cm) | petal | width | (cm |
| 0 |  |  | 5.1 |  |  | 3.5 |  |  | 1.4 |  |  | 0. |
| 2 |  |  |  |  |  |  |  |  |  |  |  |  |
| 1 |  |  | 4.9 |  |  | 3.0 |  |  | 1.4 |  |  | 0. |
| 2 |  |  |  |  |  |  |  |  |  |  |  |  |
| 2 |  |  | 4.7 |  |  | 3.2 |  |  | 1.3 |  |  | 0. |
| 2 |  |  |  |  |  |  |  |  |  |  |  |  |
| 3 |  |  | 4.6 |  |  | 3.1 |  |  | 1.5 |  |  | 0. |
| 2 |  |  |  |  |  |  |  |  |  |  |  |  |
| 4 |  |  | 5.0 |  |  | 3.6 |  |  | 1.4 |  |  | 0. |
| 2 |  |  | | | | | | | | | | |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| sepal  \ | length | (cm) | sepal | width | (cm) | petal | length | (cm) | petal | width | (cm |
|  |  | 5.1 |  |  | 3.5 |  |  | 1.4 |  |  | 0. |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | 4.9 |  |  | 3.0 |  |  | 1.4 |  |  | 0. |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | 4.7 |  |  | 3.2 |  |  | 1.3 |  |  | 0. |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | 4.6 |  |  | 3.1 |  |  | 1.5 |  |  | 0. |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | 5.0 |  |  | 3.6 |  |  | 1.4 |  |  | 0. |
|  |  | | | | | | | | | | |
|  |
|  |
|  |
|  |
|  |

accuracy = accuracy\_score(y\_test, y\_pred) conf\_matrix = confusion\_matrix(y\_test, y\_pred) class\_report = classification\_report(y\_test, y\_pred)

print(f"Accuracy: {accuracy:.2f}") print("Confusion Matrix:") print(conf\_matrix) print("Classification Report:") print(class\_report)

Accuracy: 0.98

|  |  |  |
| --- | --- | --- |
| Confusion Matrix: | | |
| [[19 | 0 | 0] |
| [ 0 | 12 | 1] |
| [ 0 | 0 | 13]] |
| Classification Report: | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | precision | recall | f1-score | support |
| 0 | 1.00 | 1.00 | 1.00 | 19 |
| 1 | 1.00 | 0.92 | 0.96 | 13 |
| 2 | 0.93 | 1.00 | 0.96 | 13 |
| accuracy |  |  | 0.98 | 45 |
| macro avg | 0.98 | 0.97 | 0.97 | 45 |
| weighted avg | 0.98 | 0.98 | 0.98 | 45 |

## Implement Hidden Markov Models using hmmlearn

!pip install hmmlearn

**import** numpy **as** np

**import** matplotlib.pyplot **as** plt

**from** hmmlearn **import** hmm

Requirement already satisfied: hmmlearn in /usr/local/lib/python3.10/dist- packages (0.3.2)

Requirement already satisfied: numpy>=1.10 in /usr/local/lib/python3.10/di st-packages (from hmmlearn) (1.26.4)

Requirement already satisfied: scikit-learn!=0.22.0,>=0.16 in /usr/local/l ib/python3.10/dist-packages (from hmmlearn) (1.5.2)

Requirement already satisfied: scipy>=0.19 in /usr/local/lib/python3.10/di st-packages (from hmmlearn) (1.13.1)

Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/ dist-packages (from scikit-learn!=0.22.0,>=0.16->hmmlearn) (1.4.2) Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/pyth on3.10/dist-packages (from scikit-learn!=0.22.0,>=0.16->hmmlearn) (3.5.0)

np.random.seed(42)

n\_samples = 1000

n\_states = 2

trans\_probs = np.array([[0.7, 0.3],

[0.4, 0.6]])

means = np.array([[1.0], [0.5]])

covars = np.array([[0.1], [0.2]])

model = hmm.GaussianHMM(n\_components=n\_states, covariance\_type="diag", n\_i ter=100)

model.startprob\_ = np.array([0.6, 0.4]) model.transmat\_ = trans\_probs model.means\_ = means

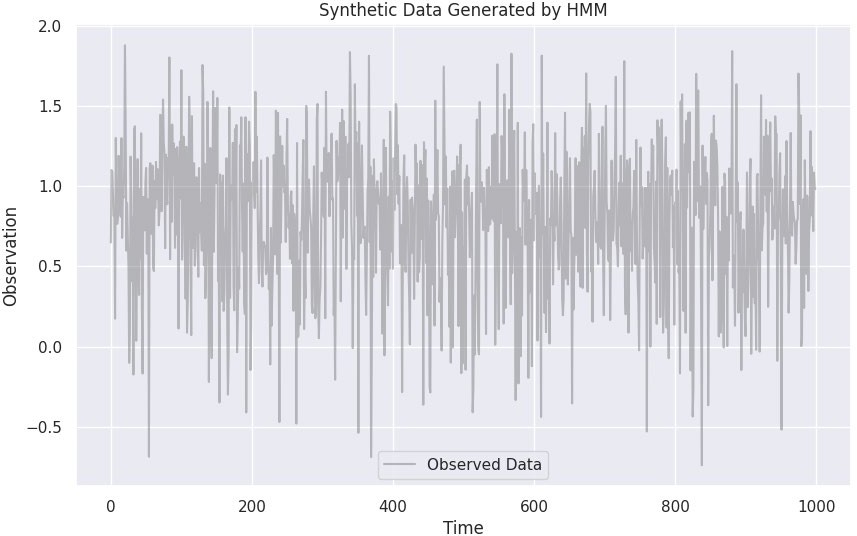
model.covars\_ = covars

X, Z = model.sample(n\_samples) plt.figure(figsize=(10, 6))

plt.plot(X, label='Observed Data', color='grey', alpha=0.5) plt.title('Synthetic Data Generated by HMM') plt.xlabel('Time')

plt.ylabel('Observation') plt.legend()

plt.show()



model.fit(X)

hidden\_states = model.predict(X)

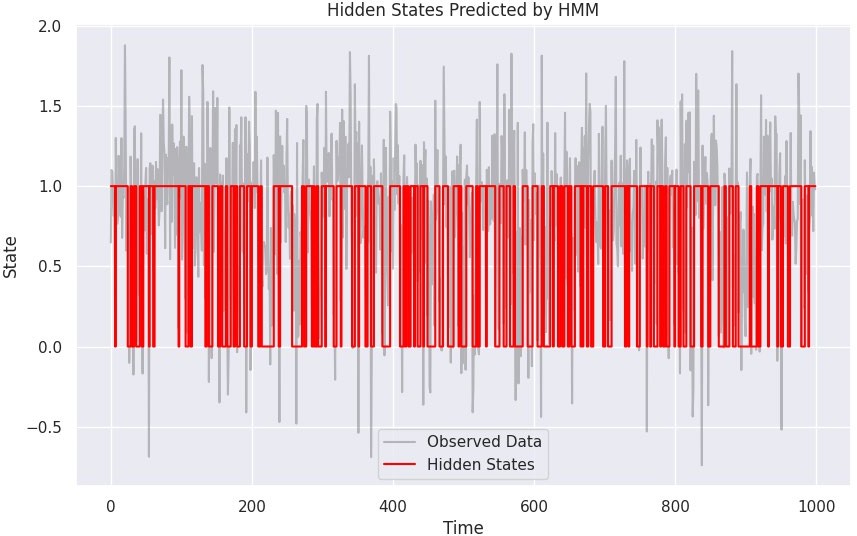
WARNING:hmmlearn.base:Even though the 'startprob\_' attribute is set, it wi ll be overwritten during initialization because 'init\_params' contains 's' WARNING:hmmlearn.base:Even though the 'transmat\_' attribute is set, it wil l be overwritten during initialization because 'init\_params' contains 't' WARNING:hmmlearn.base:Even though the 'means\_' attribute is set, it will b e overwritten during initialization because 'init\_params' contains 'm' WARNING:hmmlearn.base:Even though the 'covars\_' attribute is set, it will be overwritten during initialization because 'init\_params' contains 'c'

plt.figure(figsize=(10, 6))

plt.plot(X, label='Observed Data', color='grey', alpha=0.5) plt.step(range(n\_samples), hidden\_states, where="post", label='Hidden Stat es', color='red')

plt.title('Hidden States Predicted by HMM') plt.xlabel('Time')

plt.ylabel('State') plt.legend() plt.show()



print("Transition matrix:\n", model.transmat\_) print("Means:\n", model.means\_) print("Covariances:\n", model.covars\_)

Transition matrix: [[0.65865532 0.34134468]

[0.3121865 0.6878135 ]]

Means: [[0.54954006]

[1.00338912]]

Covariances: [[[0.22176075]]

[[0.09283459]]]

# Practical 6

## Implement Bayesian Linear Regression to explore prior and posterior distribution.

**import** numpy **as** np

**import** matplotlib.pyplot **as** plt

**from** scipy.stats **import** multivariate\_normal

np.random.seed(42)

X = np.random.rand(100, 1) \* 10 true\_beta = np.array([2.0])

y = 2.0 \* X.flatten() + np.random.normal(0, 2, size=X.shape[0]) X\_b = np.c\_[np.ones((X.shape[0], 1)), X]

sigma\_0 = 10

sigma\_n = 4

sigma\_0\_inv = 1 / sigma\_0 sigma\_n\_inv = 1 / sigma\_n N = X\_b.shape[0]

beta\_prior\_mean = np.zeros(X\_b.shape[1]) beta\_prior\_cov = sigma\_0 \* np.eye(X\_b.shape[1])

posterior\_cov = np.linalg.inv(sigma\_n\_inv \* (X\_b.T @ X\_b) + sigma\_0\_inv \* np.eye(X\_b.shape[1]))

posterior\_mean = posterior\_cov @ (sigma\_n\_inv \* (X\_b.T @ y))

beta\_samples = np.random.multivariate\_normal(posterior\_mean, posterior\_cov

, size=1000) plt.figure(figsize=(10, 6))

beta\_prior\_samples = np.random.multivariate\_normal(beta\_prior\_mean, beta\_p rior\_cov, size=1000)

<Figure size 1000x600 with 0 Axes>

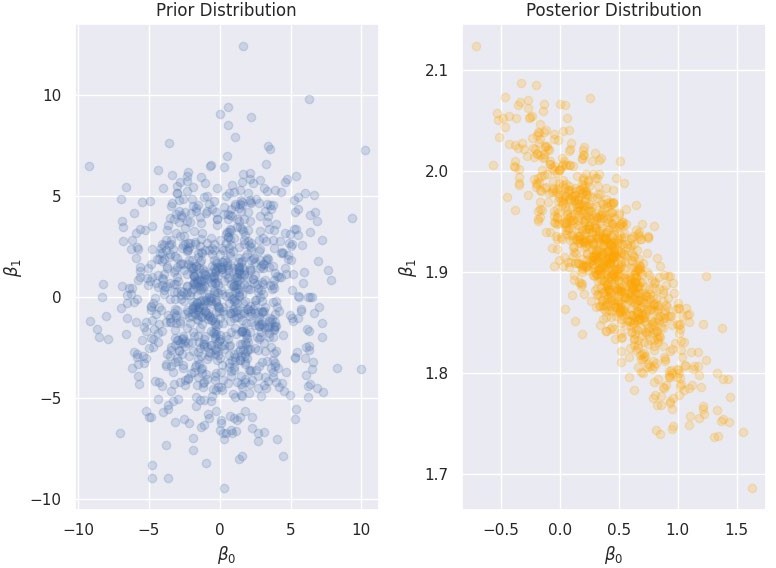
plt.subplot(1, 2, 1) plt.title("Prior Distribution")

plt.scatter(beta\_prior\_samples[:, 0], beta\_prior\_samples[:, 1], alpha=0.2) plt.xlabel("$\\beta\_0$")

plt.ylabel("$\\beta\_1$") plt.subplot(1, 2, 2) plt.title("Posterior Distribution")

plt.scatter(beta\_samples[:, 0], beta\_samples[:, 1], alpha=0.2, color='oran ge')

plt.xlabel("$\\beta\_0$") plt.ylabel("$\\beta\_1$") plt.tight\_layout() plt.show()



print("Posterior Mean:", posterior\_mean) print("Posterior Covariance:\n", posterior\_cov)

Posterior Mean: [0.42825291 1.90809351] Posterior Covariance:

[[ 0.1389247 -0.0211579 ]

[-0.0211579 0.00451795]]

## Implement Gaussian Mixture Models for density estimation and unsupervised clustering

**import** numpy **as** np

**import** matplotlib.pyplot **as** plt

**from** sklearn.mixture **import** GaussianMixture

np.random.seed(42)

means = [[2, 2], [8, 8], [5, 1]]

covariances = [[[1, 0], [0, 1]], [[1, 0], [0, 1]], [[1, 0], [0, 1]]]

n\_samples = 500 data = np.vstack([

np.random.multivariate\_normal(mean, cov, n\_samples // len(means))

**for** mean, cov **in** zip(means, covariances)

])

n\_components = len(means) *# Number of clusters*

gmm = GaussianMixture(n\_components=n\_components, covariance\_type='full') gmm.fit(data)

labels = gmm.predict(data)

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)

plt.scatter(data[:, 0], data[:, 1], c=labels, s=30, cmap='viridis', alpha= 0.5)

plt.title('GMM Clustering') plt.xlabel('X1')

plt.ylabel('X2')

x = np.linspace(-1, 10, 100)

y = np.linspace(-1, 10, 100) X, Y = np.meshgrid(x, y)

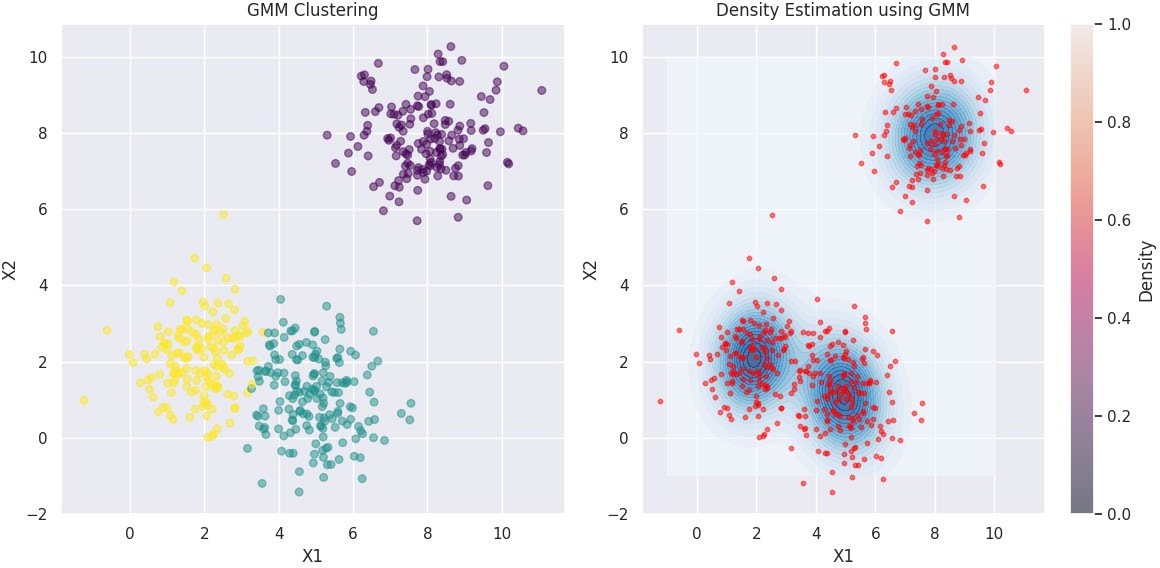
XX = np.column\_stack([X.ravel(), Y.ravel()]) logprob = gmm.score\_samples(XX)

pdf = np.exp(logprob).reshape(X.shape) plt.subplot(1, 2, 2)

plt.contourf(X, Y, pdf, levels=20, cmap='Blues', alpha=0.7) plt.scatter(data[:, 0], data[:, 1], c='red', s=10, alpha=0.5) plt.title('Density Estimation using GMM')

plt.xlabel('X1')

plt.ylabel('X2') plt.colorbar(label='Density') plt.tight\_layout() plt.show()



# Practical 7

## Implement cross-validation techniques (k-fold, stratified, etc.) for robust model evaluation

**import** numpy **as** np

**from** sklearn.model\_selection **import** KFold, StratifiedKFold

**from** sklearn.metrics **import** accuracy\_score

**from** sklearn.datasets **import** load\_iris

**from** sklearn.ensemble **import** RandomForestClassifier

data = load\_iris()

X, y = data.data, data.target model = RandomForestClassifier()

kf = KFold(n\_splits=5, shuffle=True, random\_state=42) kf\_scores = []

**for** train\_index, test\_index **in** kf.split(X): X\_train, X\_test = X[train\_index], X[test\_index] y\_train, y\_test = y[train\_index], y[test\_index] model.fit(X\_train, y\_train)

predictions = model.predict(X\_test)

score = accuracy\_score(y\_test, predictions) kf\_scores.append(score)

print(f'K-Fold Accuracy: {np.mean(kf\_scores):.2f} ± {np.std(kf\_scores):.2f

}')

K-Fold Accuracy: 0.96 ± 0.02

skf = StratifiedKFold(n\_splits=5, shuffle=True, random\_state=42) skf\_scores = []

**for** train\_index, test\_index **in** skf.split(X, y): X\_train, X\_test = X[train\_index], X[test\_index] y\_train, y\_test = y[train\_index], y[test\_index] model.fit(X\_train, y\_train)

predictions = model.predict(X\_test)

score = accuracy\_score(y\_test, predictions) skf\_scores.append(score)

print(f'Stratified K-Fold Accuracy: {np.mean(skf\_scores):.2f} ± {np.std(sk f\_scores):.2f}')

Stratified K-Fold Accuracy: 0.95 ± 0.03

## Systematically explore combinations of hyperparameters to optimize model performance.(use grid and randomized search)

**import** numpy **as** np

**import** pandas **as** pd

**from** sklearn.datasets **import** load\_iris

**from** sklearn.model\_selection **import** train\_test\_split, GridSearchCV, Random izedSearchCV

**from** sklearn.ensemble **import** RandomForestClassifier

**from** sklearn.metrics **import** accuracy\_score

data = load\_iris()

X, y = data.data, data.target

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25, random\_state=42)

model = RandomForestClassifier(random\_state=42) param\_grid = {

'n\_estimators': [10, 50, 100],

'max\_depth': [None, 5, 10, 20],

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

}

grid\_search = GridSearchCV(estimator=model, param\_grid=param\_grid, scoring

='accuracy', cv=5) grid\_search.fit(X\_train, y\_train)

print("Grid Search Best Parameters:", grid\_search.best\_params\_) print("Grid Search Best Score:", grid\_search.best\_score\_)

Grid Search Best Parameters: {'max\_depth': 5, 'min\_samples\_split': 5, 'n\_e stimators': 10}

Grid Search Best Score: 0.9636363636363636

param\_dist = {

'n\_estimators': np.arange(10, 200, 10),

'max\_depth': [None] + list(np.arange(1, 20, 1)),

'min\_samples\_split': np.arange(2, 20, 2),

}

random\_search = RandomizedSearchCV(estimator=model, param\_distributions=pa ram\_dist, n\_iter=50, scoring='accuracy', cv=5, random\_state=42) random\_search.fit(X\_train, y\_train)

print("Randomized Search Best Parameters:", random\_search.best\_params\_) print("Randomized Search Best Score:", random\_search.best\_score\_)

Randomized Search Best Parameters: {'n\_estimators': 120, 'min\_samples\_spli t': 16, 'max\_depth': None}

Randomized Search Best Score: 0.9636363636363636

best\_model = grid\_search.best\_estimator\_ y\_pred = best\_model.predict(X\_test)

# Practical 8

## Implement Bayesian Learning using inferences

**import** numpy **as** np

P\_A = 0.5

P\_B = 0.5

**def** likelihood\_heads(coin, flips):

**if** coin == 'A':

**return** (0.5 \* flips) \* (0.5 \* (10 - flips))

**elif** coin == 'B':

**return** (0.9 \* flips) \* (0.1 \* (10 - flips))

observed\_heads = 8

total\_flips = 10

likelihood\_A = likelihood\_heads('A', observed\_heads) likelihood\_B = likelihood\_heads('B', observed\_heads)

marginal\_likelihood = (likelihood\_A \* P\_A) + (likelihood\_B \* P\_B)

posterior\_A = (likelihood\_A \* P\_A) / marginal\_likelihood posterior\_B = (likelihood\_B \* P\_B) / marginal\_likelihood

print(f"Posterior Probability of Coin A: {posterior\_A:.4f}") print(f"Posterior Probability of Coin B: {posterior\_B:.4f}")

Posterior Probability of Coin A: 0.7353 Posterior Probability of Coin B: 0.2647

# Practical 9

## Set up a generator network to produce samples and a discriminator network to distinguish between real and generated data. (Use a simple small dataset)

**import** numpy **as** np

**import** matplotlib.pyplot **as** plt

**import** tensorflow **as** tf

**from** tensorflow.keras **import** layers, models

(X\_train, \_), (\_, \_) = tf.keras.datasets.mnist.load\_data() X\_train = (X\_train.astype(np.float32) - 127.5) / 127.5 X\_train = np.expand\_dims(X\_train, axis=-1)

latent\_dim = 100

num\_examples\_to\_generate = 16

**def** build\_generator():

model = models.Sequential() model.add(layers.Dense(256, input\_dim=latent\_dim)) model.add(layers.LeakyReLU(alpha=0.2)) model.add(layers.BatchNormalization()) model.add(layers.Dense(512)) model.add(layers.LeakyReLU(alpha=0.2)) model.add(layers.BatchNormalization()) model.add(layers.Dense(1024)) model.add(layers.LeakyReLU(alpha=0.2)) model.add(layers.BatchNormalization())

model.add(layers.Dense(28 \* 28 \* 1, activation='tanh'))

model.add(layers.Reshape((28, 28, 1)))

**return** model

**def** build\_discriminator(): model = models.Sequential()

model.add(layers.Flatten(input\_shape=(28, 28, 1))) model.add(layers.Dense(512)) model.add(layers.LeakyReLU(alpha=0.2)) model.add(layers.Dense(256)) model.add(layers.LeakyReLU(alpha=0.2)) model.add(layers.Dense(1, activation='sigmoid')) **return** model

generator = build\_generator() discriminator = build\_discriminator()

discriminator.compile(optimizer='adam', loss='binary\_crossentropy', metric s=['accuracy'])

discriminator.trainable = False

gan\_input = layers.Input(shape=(latent\_dim,)) generated\_image = generator(gan\_input) gan\_output = discriminator(generated\_image) gan = models.Model(gan\_input, gan\_output)

gan.compile(optimizer='adam', loss='binary\_crossentropy')

**def** generate\_and\_save\_images(model, epoch, test\_input): predictions = model(test\_input)

predictions = (predictions.numpy() + 1) / 2 *# Rescale to [0, 1]*

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

**for** i **in** range(predictions.shape[0]): plt.subplot(4, 4, i + 1) plt.imshow(predictions[i, :, :, 0], cmap='gray') plt.axis('off')

plt.savefig(f'gan\_epoch\_{epoch}.png') plt.show()

**def** train\_gan(epochs, batch\_size):

random\_latent\_vectors = tf.random.normal(shape=(num\_examples\_to\_genera te, latent\_dim))

**for** epoch **in** range(epochs):

idx = np.random.randint(0, X\_train.shape[0], batch\_size) real\_images = X\_train[idx]

noise = tf.random.normal(shape=(batch\_size, latent\_dim)) fake\_images = generator(noise)

combined\_images = tf.concat([real\_images, fake\_images], axis=0) labels = tf.constant([[1.0]] \* batch\_size + [[0.0]] \* batch\_size) d\_loss = discriminator.train\_on\_batch(combined\_images, labels)

noise = tf.random.normal(shape=(batch\_size, latent\_dim)) misleading\_labels = tf.constant([[1.0]] \* batch\_size)

g\_loss = gan.train\_on\_batch(noise, misleading\_labels)

**if** epoch % 100 == 0: print(f"Epoch: {epoch}")

print(f"Discriminator Loss: {d\_loss[0]}") print(f"Generator Loss: {g\_loss}")

generate\_and\_save\_images(generator, epoch, random\_latent\_vecto

rs)

epochs = 300

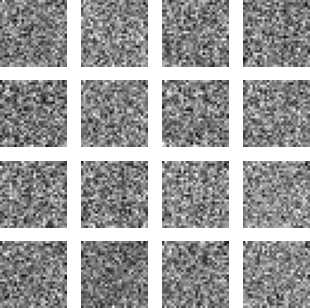
batch\_size = 64

train\_gan(epochs, batch\_size) Epoch: 0

Discriminator Loss: 0.7258248329162598

Generator Loss: [array(0.72582483, dtype=float32), array(0.72582483, dtype

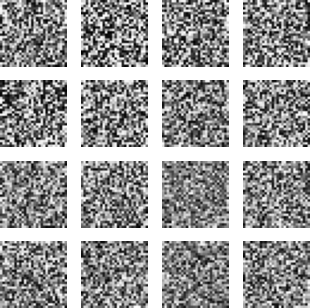
=float32), array(0.390625, dtype=float32)]



Epoch: 100

Discriminator Loss: 2.1150412559509277

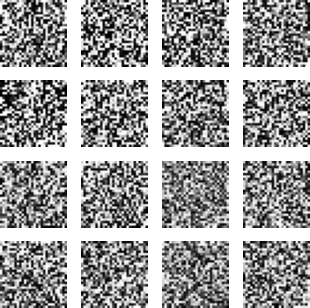
Generator Loss: [array(2.1150413, dtype=float32), array(2.1150413, dtype=f loat32), array(0.20482673, dtype=float32)]



Epoch: 200

Discriminator Loss: 2.8626105785369873

Generator Loss: [array(2.8626106, dtype=float32), array(2.8626106, dtype=f loat32), array(0.20747824, dtype=float32)]



# Practical 10

## Develop an API to deploy your model and perform predictions

*# Required Libraries*

!pip install pyngrok flask scikit-learn

*# Importing Libraries*

**from** sklearn.datasets **import** load\_iris

**from** sklearn.model\_selection **import** train\_test\_split **from** sklearn.ensemble **import** RandomForestClassifier **import** pickle

**from** flask **import** Flask, request, jsonify

**from** pyngrok **import** ngrok

*# Load dataset*

iris = load\_iris()

X, y = iris.data, iris.target

*# Split data*

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

*# Train a model*

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

*# Save the model*

**with** open('model.pkl', 'wb') **as** model\_file: pickle.dump(model, model\_file)

*# Load the model*

**with** open('model.pkl', 'rb') **as** model\_file: model = pickle.load(model\_file)

*# Create Flask app* app = Flask( name ) port = "5000"

@app.route('/')

**def** home():

**return** "Welcome to the Iris Prediction API! Use the /predict endpoint to make predictions."

@app.route('/predict', methods=['POST'])

**def** predict():

data = request.json

features = data.get('features')

*# Ensure the features are in the correct format*

**if not** features **or** len(features) != 4: *# Assuming 4 features for iris dataset*

**return** jsonify({'error': 'Invalid input format. Please provide 4 f eatures.'}), 400

**try**:

prediction = model.predict([features]) *# Wrap features in a list*

*to create 2D array*

**return** jsonify({'prediction': int(prediction[0])}) *# Convert pred iction to int*

**except** Exception **as** e:

**return** jsonify({'error': str(e)}), 500

*# Start ngrok and print the public URL* ngrok.set\_auth\_token("api\_auth\_token") public\_url = ngrok.connect(port).public\_url print("Public URL:", public\_url)

*# Run the Flask app*

**if**  name == ' main ': app.run(port=port)

Requirement already satisfied: pyngrok in /usr/local/lib/python3.10/dist-p ackages (7.2.0)

Requirement already satisfied: flask in /usr/local/lib/python3.10/dist-pac kages (2.2.5)

Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/d ist-packages (1.5.2)

Requirement already satisfied: PyYAML>=5.1 in /usr/local/lib/python3.10/di st-packages (from pyngrok) (6.0.2)

Requirement already satisfied: Werkzeug>=2.2.2 in /usr/local/lib/python3.1 0/dist-packages (from flask) (3.0.4)

Requirement already satisfied: Jinja2>=3.0 in /usr/local/lib/python3.10/di st-packages (from flask) (3.1.4)

Requirement already satisfied: itsdangerous>=2.0 in /usr/local/lib/python3

.10/dist-packages (from flask) (2.2.0)

Requirement already satisfied: click>=8.0 in /usr/local/lib/python3.10/dis t-packages (from flask) (8.1.7)

Requirement already satisfied: numpy>=1.19.5 in /usr/local/lib/python3.10/ dist-packages (from scikit-learn) (1.26.4)

Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.10/d ist-packages (from scikit-learn) (1.13.1)

Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/ dist-packages (from scikit-learn) (1.4.2)

Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/pyth on3.10/dist-packages (from scikit-learn) (3.5.0)

Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.1 0/dist-packages (from Jinja2>=3.0->flask) (2.1.5)

Public URL: https://2f62-49-43-24-101.ngrok-free.app

* Serving Flask app ' main '
* Debug mode: off

INFO:werkzeug:WARNING: This is a development server. Do not use it in a pr oduction deployment. Use a production WSGI server instead.

* Running on http://127.0.0.1:5000 INFO:werkzeug:Press CTRL+C to quit

INFO:werkzeug:127.0.0.1 - - [24/Oct/2024 07:50:42] "POST /predict HTTP/1.1 " 200 –

