Assignment 1

Shivani Narwariya

211112247

ML lab

Question 1

Q1 Refer to any 5 regression problems publicly available at https://sci2s.ugr.es/keel/category.php?cat=reg Find RMSE, MAE and coefficient of determination using following approaches.

1. Linear regression

2. Regression order 2 and 3

3. Regression with Ridge Regularization

Solution :

DATASET 1 : Diabetes

1) linear regression

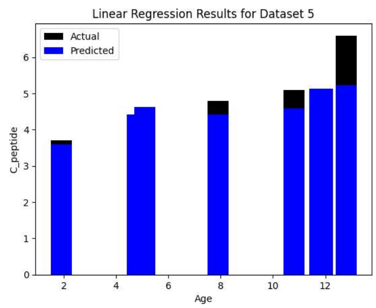
```
import numpy as np

import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

import matplotlib.pyplot as plt

sum_rmse=0

sum_mae=0

sum_r_squared=0

for i in range(1, 6):

 train_path=r"C:/Users/rinki/OneDrive/Desktop/test jupyter/diabetes-5-fold/diabetes-5-"+str(i)+"tra.dat"

 filename_train = train_path

 test_path=r"C:/Users/rinki/OneDrive/Desktop/test jupyter/diabetes-5-fold/diabetes-5-"+str(i)+"tst.dat"
```

```
 filename_test =test_path
 data_train = pd.read_csv(filename_train, delimiter=',', skiprows=7, names=['Age',
'Deficit','C_peptide'])
 X_train = data_train[['Age', 'Deficit']]
 y_train = data_train['C_peptide']
 data_test = pd.read_csv(filename_test, delimiter=',', skiprows=7, names=['Age',
'Deficit','C_peptide'])
 X_test = data_test[['Age', 'Deficit']]
 y_test = data_test['C_peptide']
 model = LinearRegression()
 model.fit(X_train, y_train)
 y_pred = model.predict(X_test)
 rmse = np.sqrt(mean_squared_error(y_test, y_pred))
 mae = mean_absolute_error(y_test, y_pred)
 r_squared = r2_score(y_test, y_pred)
 print(f'\nResults for dataset {i}:')
 print(f'RMSE: {rmse}')
 print(f'MAE: {mae}')
 print(f'R-squared: {r_squared}')
 sum_rmse=sum_rmse+rmse
 sum_mae=sum_mae+mae
 sum_r_squared=sum_r_squared+r_squared
plt.bar(X_test['Age'], y_test, color='black', label='Actual')
plt.bar(X_test['Age'], y_pred, color='blue', label='Predicted')
plt.xlabel('Age')
plt.ylabel('C_peptide')
plt.title(f'Linear Regression Results for Dataset {i}')
plt.legend()
plt.show()
avg_rmse=sum_rmse/5
avg_mae=sum_mae/5
avg_r_squared=r_squared/5
```

print(f'AVG_RMSE: {avg_rmse}')

print(f'AVG_MAE: {avg_mae}')

print(f'AVG_R-squared: {avg_r_squared}')



AVG_RMSE: 0.6275034933741999

AVG_MAE: 0.49407720733295485

AVG_R-squared: 0.0973983746540477

2) linear regression degree=2

from sklearn.pipeline import make_pipeline

from sklearn.preprocessing import PolynomialFeatures

sum_rmse=0

sum_mae=0

sum_r_squared=0

for i in range(1, 6):

 train_path=r"C:/Users/rinki/OneDrive/Desktop/test jupyter/diabetes-5-fold/diabetes-5-"+str(i)+"tra.dat"

 filename_train = train_path

 test_path=r"C:/Users/rinki/OneDrive/Desktop/test jupyter/diabetes-5-fold/diabetes-5-"+str(i)+"tst.dat"

 filename_test =test_path

 data_train = pd.read_csv(filename_train, delimiter=',', skiprows=7, names=['Age', 'Deficit', 'C_peptide'])

 X_train = data_train[['Age', 'Deficit']]

 y_train = data_train['C_peptide']

 data_test = pd.read_csv(filename_test, delimiter=',', skiprows=7, names=['Age', 'Deficit', 'C_peptide'])

 X_test = data_test[['Age', 'Deficit']]

 y_test = data_test['C_peptide']

```
model = make_pipeline(PolynomialFeatures(degree=2), LinearRegression())
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
mae = mean_absolute_error(y_test, y_pred)
r_squared = r2_score(y_test, y_pred)
print(f'\nResults for dataset {i}:')
print(f'RMSE: {rmse}')
print(f'MAE: {mae}')
print(f'R-squared: {r_squared}')
sum_rmse=sum_rmse+rmse
sum_mae=sum_mae+mae
sum_r_squared=sum_r_squared+r_squared
plt.scatter(X_test['Age'], y_test, color='black', label='Actual')
plt.scatter(X_test['Age'], y_pred, color='blue', label='Predicted')
plt.xlabel('Age')
plt.ylabel('C_peptide')
plt.title(f'Regression of order 2 Results for Dataset {i}')
plt.legend()
plt.show()
avg_rmse=sum_rmse/5
avg_mae=sum_mae/5
avg_r_squared=r_squared/5
print(f'AVG_RMSE: {avg_rmse}')
print(f'AVG_MAE: {avg_mae}')
print(f'AVG_R-squared: {avg_r_squared}')
```

output:

AVG_RMSE: 0.547295759722552

AVG_MAE: 0.45727247144417393

AVG_R-squared: 0.09516721255278697

3) linear regression degree=3

```
sum_rmse=0

sum_mae=0

sum_r_squared=0

for i in range(1, 6):

 train_path=r"C:/Users/rinki/OneDrive/Desktop/test jupyter/diabetes-5-fold/diabetes-5-
"+str(i)+"tra.dat"

 filename_train = train_path

 test_path=r"C:/Users/rinki/OneDrive/Desktop/test jupyter/diabetes-5-fold/diabetes-5-
"+str(i)+"tst.dat"

 filename_test =test_path

 data_train = pd.read_csv(filename_train, delimiter=',', skiprows=7, names=['Age', 'Deficit',
'C_peptide'])

 X_train = data_train[['Age', 'Deficit']]

 y_train = data_train['C_peptide']

 data_test = pd.read_csv(filename_test, delimiter=',', skiprows=7, names=['Age', 'Deficit',
'C_peptide'])

 X_test = data_test[['Age', 'Deficit']]

 y_test = data_test['C_peptide']

 model = make_pipeline(PolynomialFeatures(degree=3), LinearRegression())

 model.fit(X_train, y_train)

 y_pred = model.predict(X_test)

 rmse = np.sqrt(mean_squared_error(y_test, y_pred))

 mae = mean_absolute_error(y_test, y_pred)

 r_squared = r2_score(y_test, y_pred)

 print(f'\nResults for dataset {i}:')

 print(f'RMSE: {rmse}')

 print(f'MAE: {mae}')

 print(f'R-squared: {r_squared}')

 sum_rmse=sum_rmse+rmse

 sum_mae=sum_mae+mae

 sum_r_squared=sum_r_squared+r_squared

plt.scatter(X_test['Age'], y_test, color='black', label='Actual')
```

```
plt.scatter(X_test['Age'], y_pred, color='blue', label='Predicted')
plt.xlabel('Age')
plt.ylabel('C_peptide')
plt.title(f'Regression of order 2 Results for Dataset {i}')
plt.legend()
plt.show()
avg_rmse=sum_rmse/5
avg_mae=sum_mae/5
avg_r_squared=r_squared/5
print(f'AVG_RMSE: {avg_rmse}')
print(f'AVG_MAE: {avg_mae}')
print(f'AVG_R-squared: {avg_r_squared}')
```

OUTPUT:-

AVG_RMSE: 1.0181883870270616

AVG_MAE: 0.7174549374900028

AVG_R-squared: 0.057439353600545376

4) Ridge Regularization

```
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler
sum_rmse=0
sum_mae=0
sum_r_squared=0
sum_best_c_rmse=0
sum_best_c_mae=0
sum_best_c_r=0
sum_best_d_rmse=0
sum_best_d_mae=0
sum_best_d_r=0
sum_C=0
sum_D=0
```

```
for i in range(1, 6):
 train_path=r"C:/Users/rinki/OneDrive/Desktop/test jupyter/diabetes-5-fold/diabetes-5-"+str(i)+"tra.dat"
 filename_train = train_path
 test_path=r"C:/Users/rinki/OneDrive/Desktop/test jupyter/diabetes-5-fold/diabetes-5-"+str(i)+"tst.dat"
 filename_test =test_path
 data_train = pd.read_csv(filename_train, delimiter=',', skiprows=7, names=['Age', 'Deficit', 'C_peptide'])
 X_train = data_train[['Age', 'Deficit']]
 y_train = data_train['C_peptide']
 data_test = pd.read_csv(filename_test, delimiter=',', skiprows=7, names=['Age', 'Deficit', 'C_peptide'])
 X_test = data_test[['Age', 'Deficit']]
 y_test = data_test['C_peptide']
 alphas = 2.0**np.arange(-18, 51)
 degrees = [1, 2, 3]
 best_alpha = None
 best_degree = None
 min_rmse = float('inf')
 min_mae=float('inf')
 max_r_squared=float('-inf')
 for degree in degrees:
  for alpha in alphas:
   polynomial_features = PolynomialFeatures(degree)
   X_poly_train = polynomial_features.fit_transform(X_train)
   X_poly_test = polynomial_features.transform(X_test)
   scaler = StandardScaler()
   X_poly_train_scaled = scaler.fit_transform(X_poly_train)
   X_poly_test_scaled = scaler.transform(X_poly_test)
   ridge = Ridge(alpha=alpha)
   ridge.fit(X_poly_train_scaled, y_train)
```

```
   y_pred = ridge.predict(X_poly_test_scaled)
   rmse = np.sqrt(mean_squared_error(y_test, y_pred))
   mae = mean_absolute_error(y_test, y_pred)
   r_squared = r2_score(y_test, y_pred)
   best_alpha=alpha
   best_degree=degree
   if rmse < min_rmse:
    min_rmse = rmse
  best_alpha_rmse = alpha
  best_degree_rmse = degree
  if mae < min_mae:
    min_mae = mae
    best_alpha_mae = alpha
    best_degree_mae = degree
  if r_squared> max_r_squared:
   max_r_squared = r_squared
   best_alpha_r = alpha
   best_degree_r = degree
 sum_C=sum_C+best_alpha
 sum_D=sum_D+best_degree
 sum_rmse=sum_rmse+min_rmse
 sum_mae=sum_mae+min_mae
 sum_r_squared=sum_r_squared+max_r_squared
 sum_best_c_rmse=sum_best_c_rmse+ best_alpha_rmse
 sum_best_d_rmse=sum_best_d_rmse+ best_degree_rmse
 sum_best_c_mae=sum_best_c_mae+ best_alpha_mae
 sum_best_d_mae=sum_best_d_mae+ best_degree_mae
 sum_best_c_r=sum_best_c_r+ best_alpha_r
 sum_best_d_r=sum_best_d_r+ best_degree_r
avg_c=sum_C/5
avg_d=sum_D/5
```

```
avg_r_squared=r_squared/5
avg_c_rmse=sum_best_c_rmse/5
avg_d_rmse=sum_best_d_rmse/5
avg_c_mae=sum_best_c_mae/5
avg_d_mae=sum_best_d_mae/5
avg_c_r=sum_best_c_r/5
avg_d_r=sum_best_d_r/5
print(f'best c={avg_c}')
print(f'best degree={avg_d}')
print(f'AVG_RMSE: {avg_rmse}')
print(f'best_C_RMSE: {avg_c_rmse}')
print(f'AVG_D_RMSE: {avg_d_rmse}')
print(f'AVG_MAE: {avg_mae}')
print(f'best_C_RMSE: {avg_c_mae}')
print(f'AVG_D_RMSE: {avg_d_mae}')
print(f'AVG_R-squared: {avg_r_squared}')
print(f'best_C_RMSE: {avg_c_r}')
print(f'AVG_D_RMSE: {avg_d_r}')
```

Output:

best c=1125899906842624.0

best degree=3.0

AVG_RMSE: 1.0181883870270616

best_C_RMSE: 225179981368525.06

AVG_D_RMSE: 2.2

AVG_MAE: 0.7174549374900028

best_C_RMSE: 1125899906842624.0

AVG_D_RMSE: 2.4

AVG_R-squared: -0.0012517006802597486

best_C_RMSE: 1125899906842624.0

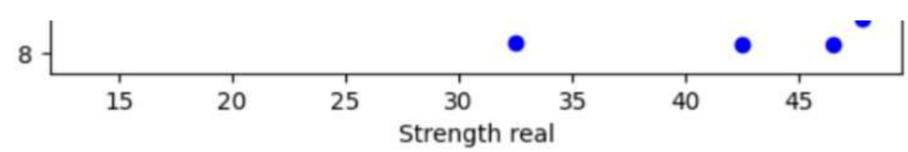
AVG_D_RMSE: 2.6

DATASET 2 PLASTIC

1) Linear regression

```
sum_rmse=0
sum_mae=0
sum_r_squared=0
for i in range(1, 6): # Assuming you have files --1.dat to --5.dat
 filename_train=f"C:/Users/rinki/OneDrive/Desktop/test jupyter/plastic-5-fold (1)/plastic-5-"+str(i)+"tra.dat"
 filename_test=f"C:/Users/rinki/OneDrive/Desktop/test jupyter/plastic-5-fold (1)/plastic-5-"+str(i)+"tst.dat"
 data_train = pd.read_csv(filename_train, delimiter=',', skiprows=7, names=['Strength real', 'Temperature real', 'Pressure real'])
 X_train = data_train[['Strength real', 'Temperature real']]
 y_train = data_train['Pressure real']
 # Load testing dataset
 data_test = pd.read_csv(filename_test, delimiter=',', skiprows=7, names=['Strength real', 'Temperature real', 'Pressure real'])
 X_test = data_test[['Strength real', 'Temperature real']]
 y_test = data_test['Pressure real']
 # Train a linear regression model
 model = LinearRegression()
 model.fit(X_train, y_train)
 # Make predictions on the test set
 y_pred = model.predict(X_test)
 # Calculate metrics - RMSE, MAE, R-squared
 rmse = np.sqrt(mean_squared_error(y_test, y_pred))
 mae = mean_absolute_error(y_test, y_pred)
 r_squared = r2_score(y_test, y_pred)
sum_rmse=sum_rmse+rmse
 sum_mae=sum_mae+mae
 sum_r_squared=sum_r_squared+r_squared
avg_rmse=sum_rmse/5
avg_mae=sum_mae/5
avg_r_squared=r_squared/5
print(f'AVG_RMSE: {avg_rmse}')
print(f'AVG_MAE: {avg_mae}')
print(f'AVG_R-squared: {avg_r_squared}')
```

output:



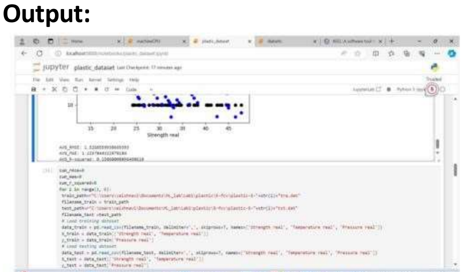

```
AVG_RMSE: 1.5304709053276642
AVG_MAE: 1.2324659378443346
AVG_R-squared: 0.1580948824838289
```

```
14]: from sklearn.pipeline import make_pipeline
```

2) Linear regression degree =2

```
sum_rmse=0
sum_mae=0
sum_r_squared=0
for i in range(1, 6): # Assuming you have files --1.dat to --5.dat
 filename_train=f"C:/Users/rinki/OneDrive/Desktop/test jupyter/plastic-5-fold (1)/plastic-5-"+str(i)+"tra.dat"
 filename_test=f"C:/Users/rinki/OneDrive/Desktop/test jupyter/plastic-5-fold (1)/plastic-5-"+str(i)+"tst.dat"
 data_train = pd.read_csv(filename_train, delimiter=',', skiprows=7, names=['Strength real', 'Temperature real', 'Pressure real'])
 X_train = data_train[['Strength real', 'Temperature real']]
 y_train = data_train['Pressure real']
 # Load testing dataset
 data_test = pd.read_csv(filename_test, delimiter=',', skiprows=7, names=['Strength real', 'Temperature real', 'Pressure real'])
 X_test = data_test[['Strength real', 'Temperature real']]
 y_test = data_test['Pressure real']
 # Train a linear regression model
 model = make_pipeline(PolynomialFeatures(degree=2), LinearRegression())
 model.fit(X_train, y_train)
 # Make predictions on the test set
 y_pred = model.predict(X_test)
 # Calculate metrics - RMSE, MAE, R-squared
 rmse = np.sqrt(mean_squared_error(y_test, y_pred))
 mae = mean_absolute_error(y_test, y_pred)
 r_squared = r2_score(y_test, y_pred)
sum_rmse=sum_rmse+rmse
 sum_mae=sum_mae+mae
 sum_r_squared=sum_r_squared+r_squared
avg_rmse=sum_rmse/5
avg_mae=sum_mae/5
avg_r_squared=r_squared/5
print(f'AVG_RMSE: {avg_rmse}')
print(f'AVG_MAE: {avg_mae}')
print(f'AVG_R-squared: {avg_r_squared}')
```

Output:

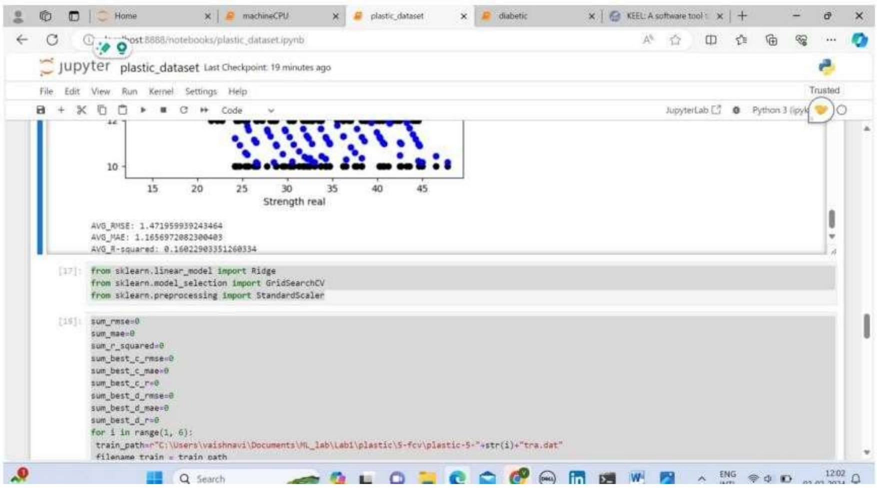


3) Linear regression degree=3

```
sum_rmse=0

sum_mae=0

sum_r_squared=0

for i in range(1, 6): # Assuming you have files --1.dat to --5.dat

 filename_train=f"C:/Users/rinki/OneDrive/Desktop/test jupyter/plastic-5-fold (1)/plastic-5-"+str(i)+"tra.dat"

 filename_test=f"C:/Users/rinki/OneDrive/Desktop/test jupyter/plastic-5-fold (1)/plastic-5-"+str(i)+"tst.dat"

 data_train = pd.read_csv(filename_train, delimiter=',', skiprows=7, names=['Strength real', 'Temperature real', 'Pressure real'])

 X_train = data_train[['Strength real', 'Temperature real']]

 y_train = data_train['Pressure real']

 # Load testing dataset

 data_test = pd.read_csv(filename_test, delimiter=',', skiprows=7, names=['Strength real', 'Temperature real', 'Pressure real'])

 X_test = data_test[['Strength real', 'Temperature real']]

 y_test = data_test['Pressure real']

 # Train a linear regression model

 model =  make_pipeline(PolynomialFeatures(degree=3), LinearRegression())

 model.fit(X_train, y_train)

 # Make predictions on the test set

 y_pred = model.predict(X_test)

 # Calculate metrics - RMSE, MAE, R-squared

 rmse = np.sqrt(mean_squared_error(y_test, y_pred))

 mae = mean_absolute_error(y_test, y_pred)

 r_squared = r2_score(y_test, y_pred)

sum_rmse=sum_rmse+rmse

 sum_mae=sum_mae+mae

 sum_r_squared=sum_r_squared+r_squared

avg_rmse=sum_rmse/5

avg_mae=sum_mae/5

avg_r_squared=r_squared/5
```

```
print(f'AVG_RMSE: {avg_rmse}')

print(f'AVG_MAE: {avg_mae}')

print(f'AVG_R-squared: {avg_r_squared}')
```

output:



4) <u>Ridge Regularization</u>

```
sum_mae=0

sum_rmse=0

sum_r_squared=0

sum_best_c_rmse=0

sum_best_c_mae=0

sum_best_c_r=0

sum_best_d_rmse=0

sum_best_d_mae=0

sum_best_d_r=0

sum_C=0

sum_D=0

for i in range(1, 6): # Assuming you have files --1.dat to --5.dat

 filename_train=f"C:/Users/rinki/OneDrive/Desktop/test jupyter/plastic-5-fold (1)/plastic-5-"+str(i)+"tra.dat"

 filename_test=f"C:/Users/rinki/OneDrive/Desktop/test jupyter/plastic-5-fold (1)/plastic-5-"+str(i)+"tst.dat"

 data_train = pd.read_csv(filename_train, delimiter=',', skiprows=7, names=['Strength real', 'Temperature real', 'Pressure real'])
```

```
X_train = data_train[['Strength real', 'Temperature real']]
y_train = data_train['Pressure real']
# Load testing dataset
data_test = pd.read_csv(filename_test, delimiter=',', skiprows=7, names=['Strength real',
'Temperature real', 'Pressure real'])
X_test = data_test[['Strength real', 'Temperature real']]
y_test = data_test['Pressure real']
alphas = 2.0**np.arange(-18, 51)
degrees = [1, 2, 3]
best_alpha = None
best_degree = None
min_rmse = float('inf')
min_mae=float('inf')
max_r_squared=float('-inf')
for degree in degrees:
 for alpha in alphas:
  polynomial_features = PolynomialFeatures(degree)
  X_poly_train = polynomial_features.fit_transform(X_train)
  X_poly_test = polynomial_features.transform(X_test)
  scaler = StandardScaler()
  X_poly_train_scaled = scaler.fit_transform(X_poly_train)
  X_poly_test_scaled = scaler.transform(X_poly_test)
  ridge = Ridge(alpha=alpha)
  ridge.fit(X_poly_train_scaled, y_train)
  y_pred = ridge.predict(X_poly_test_scaled)
  rmse = np.sqrt(mean_squared_error(y_test, y_pred))
  mae = mean_absolute_error(y_test, y_pred)
  r_squared = r2_score(y_test, y_pred)
  best_alpha=alpha
  best_degree=degree
  if rmse < min_rmse:
```

```
    min_rmse = rmse
  best_alpha_rmse = alpha
  best_degree_rmse = degree
  if mae < min_mae:
    min_mae = mae
    best_alpha_mae = alpha
    best_degree_mae = degree
  if r_squared> max_r_squared:
   max_r_squared = r_squared
   best_alpha_r = alpha
   best_degree_r = degree
 sum_C=sum_C+best_alpha
 sum_D=sum_D+best_degree
 sum_rmse=sum_rmse+min_rmse
 sum_mae=sum_mae+min_mae
 sum_r_squared=sum_r_squared+max_r_squared
 sum_best_c_rmse=sum_best_c_rmse+ best_alpha_rmse
 sum_best_d_rmse=sum_best_d_rmse+ best_degree_rmse
 sum_best_c_mae=sum_best_c_mae+ best_alpha_mae
 sum_best_d_mae=sum_best_d_mae+ best_degree_mae
 sum_best_c_r=sum_best_c_r+ best_alpha_r
 sum_best_d_r=sum_best_d_r+ best_degree_r
avg_c=sum_C/5
avg_d=sum_D/5
avg_r_squared=r_squared/5
avg_c_rmse=sum_best_c_rmse/5
avg_d_rmse=sum_best_d_rmse/5
avg_c_mae=sum_best_c_mae/5
avg_d_mae=sum_best_d_mae/5
avg_c_r=sum_best_c_r/5
avg_d_r=sum_best_d_r/5
```

print(f'best c={avg_c}')

print(f'best degree={avg_d}')

print(f'AVG_RMSE: {avg_rmse}')

print(f'best_C_RMSE: {avg_c_rmse}')

print(f'AVG_D_RMSE: {avg_d_rmse}')

print(f'AVG_MAE: {avg_mae}')

print(f'best_C_RMSE: {avg_c_mae}')

print(f'AVG_D_RMSE: {avg_d_mae}')

print(f'AVG_R-squared: {avg_r_squared}')

print(f'best_C_RMSE: {avg_c_r}')

print(f'AVG_D_RMSE: {avg_d_r}')

output:

```
AVG_RMSE: 1.4715664867729503
best_C_RMSE: 1.7547607421875e-05
AVG_D_RMSE: 3.0
AVG_MAE: 1.1617130796758213
best_C_mae: 3.7384033203125e-05
AVG_D_mae: 3.0
AVG_R-squared: -0.0010138705077283206
best_C_r: 1.7547607421875e-05
AVG_D_r: 3.0
```

DATASET 3: LASER

1) linear regression

sum_rmse=0

sum_mae=0

sum_r_squared=0

for i in range(1, 6): # Assuming you have files --1.dat to --5.dat

filename_train=f"C:/Users/rinki/OneDrive/Desktop/test jupyter/laser-5-fold (1)/laser-5-"+str(i)+"tra.dat"

filename_test=f"C:/Users/rinki/OneDrive/Desktop/test jupyter/laser-5-fold (1)/laser-5-"+str(i)+"tst.dat"

```
data_train = pd.read_csv(filename_train, delimiter=',', skiprows=9, names=['Input1 real', 'Input2
real', 'Input3 real','Input4 real','Output real'])

X_train = data_train[['Input1 real', 'Input2 real', 'Input3 real','Input4 real']]

y_train = data_train['Output real']

# Load testing dataset

data_test = pd.read_csv(filename_test, delimiter=',', skiprows=9, names=['Input1 real', 'Input2 real',
'Input3 real','Input4 real','Output real'])

X_test = data_test[['Input1 real', 'Input2 real', 'Input3 real','Input4 real']]

y_test = data_test['Output real']

# Train a linear regression model

model = LinearRegression()

model.fit(X_train, y_train)

# Make predictions on the test set

y_pred = model.predict(X_test)

# Calculate metrics - RMSE, MAE, R-squared

rmse = np.sqrt(mean_squared_error(y_test, y_pred))

mae = mean_absolute_error(y_test, y_pred)


r_squared = r2_score(y_test, y_pred)

# print(f'\nResults for dataset {i}:')

# print(f'RMSE: {rmse}')

# print(f'MAE: {mae}')

# print(f'R-squared: {r_squared}')

sum_rmse=sum_rmse+rmse

sum_mae=sum_mae+mae

sum_r_squared=sum_r_squared+r_squared

avg_rmse=sum_rmse/5

avg_mae=sum_mae/5

avg_r_squared=r_squared/5

print(f'AVG_RMSE: {avg_rmse}')

print(f'AVG_MAE: {avg_mae}')

print(f'AVG_R-squared: {avg_r_squared}')
```

output:

AVG_RMSE: 23.267078883585608

AVG_MAE: 15.56788683873371

AVG_R-squared: 0.14835722722414296

2) linear regression degree=2

sum_rmse=0

sum_mae=0

sum_r_squared=0

for i in range(1, 6): # Assuming you have files --1.dat to --5.dat

filename_train=f"C:/Users/rinki/OneDrive/Desktop/test jupyter/laser-5-fold (1)/laser-5-"+str(i)+"tra.dat"

filename_test=f"C:/Users/rinki/OneDrive/Desktop/test jupyter/laser-5-fold (1)/laser-5-"+str(i)+"tst.dat"

data_train = pd.read_csv(filename_train, delimiter=',', skiprows=9, names=['Input1 real', 'Input2 real', 'Input3 real','Input4 real','Output real'])

X_train = data_train[['Input1 real', 'Input2 real', 'Input3 real','Input4 real']]

y_train = data_train['Output real']

Load testing dataset

data_test = pd.read_csv(filename_test, delimiter=',', skiprows=9, names=['Input1 real', 'Input2 real', 'Input3 real','Input4 real','Output real'])

X_test = data_test[['Input1 real', 'Input2 real', 'Input3 real','Input4 real']]

y_test = data_test['Output real']

Train a linear regression model

model = make_pipeline(PolynomialFeatures(degree=2), LinearRegression())

model.fit(X_train, y_train)

Make predictions on the test set

y_pred = model.predict(X_test)

Calculate metrics - RMSE, MAE, R-squared

rmse = np.sqrt(mean_squared_error(y_test, y_pred))

mae = mean_absolute_error(y_test, y_pred)

r_squared = r2_score(y_test, y_pred)

sum_rmse=sum_rmse+rmse

sum_mae=sum_mae+mae

sum_r_squared=sum_r_squared+r_squared

avg_rmse=sum_rmse/5

avg_mae=sum_mae/5

avg_r_squared=r_squared/5

print(f'AVG_RMSE: {avg_rmse}')

print(f'AVG_MAE: {avg_mae}')

print(f'AVG_R-squared: {avg_r_squared}')

output:

AVG_RMSE: 10.86992993711284

AVG_MAE: 6.669598885391929

AVG_R-squared: 0.18621967169509068

3) linear regression degree=3

sum_rmse=0

sum_mae=0

sum_r_squared=0

for i in range(1, 6): # Assuming you have files --1.dat to --5.dat

filename_train=f"C:/Users/rinki/OneDrive/Desktop/test jupyter/laser-5-fold (1)/laser-5-"+str(i)+"tra.dat"

filename_test=f"C:/Users/rinki/OneDrive/Desktop/test jupyter/laser-5-fold (1)/laser-5-"+str(i)+"tst.dat"

data_train = pd.read_csv(filename_train, delimiter=',', skiprows=9, names=['Input1 real', 'Input2 real', 'Input3 real','Input4 real','Output real'])

X_train = data_train[['Input1 real', 'Input2 real', 'Input3 real','Input4 real']]

y_train = data_train['Output real']

Load testing dataset

data_test = pd.read_csv(filename_test, delimiter=',', skiprows=9, names=['Input1 real', 'Input2 real', 'Input3 real','Input4 real','Output real'])

X_test = data_test[['Input1 real', 'Input2 real', 'Input3 real','Input4 real']]

y_test = data_test['Output real']

Train a linear regression model

```
model = make_pipeline(PolynomialFeatures(degree=3), LinearRegression())
model.fit(X_train, y_train)
# Make predictions on the test set
y_pred = model.predict(X_test)
# Calculate metrics - RMSE, MAE, R-squared
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
mae = mean_absolute_error(y_test, y_pred)
r_squared = r2_score(y_test, y_pred)
sum_rmse=sum_rmse+rmse
sum_mae=sum_mae+mae
sum_r_squared=sum_r_squared+r_squared
avg_rmse=sum_rmse/5
avg_mae=sum_mae/5
avg_r_squared=r_squared/5
print(f'AVG_RMSE: {avg_rmse}')
print(f'AVG_MAE: {avg_mae}')
print(f'AVG_R-squared: {avg_r_squared}')
```

output:-

AVG_RMSE: 7.116084630060236

AVG_MAE: 3.2607303552987545

AVG_R-squared: 0.19129031192526968

4) Ridge Regularization

```
sum_rmse=0
sum_mae=0
sum_r_squared=0
sum_best_c_rmse=0
sum_best_c_mae=0
sum_best_c_r=0
sum_best_d_rmse=0
sum_best_d_mae=0
sum_best_d_r=0
```

```
sum_C=0

sum_D=0

for i in range(1, 6): # Assuming you have files --1.dat to --5.dat

 filename_train=f"C:/Users/rinki/OneDrive/Desktop/test jupyter/laser-5-fold (1)/laser-5-"+str(i)+"tra.dat"

 filename_test=f"C:/Users/rinki/OneDrive/Desktop/test jupyter/laser-5-fold (1)/laser-5-"+str(i)+"tst.dat"

 data_train = pd.read_csv(filename_train, delimiter=',', skiprows=9, names=['Input1 real', 'Input2 real', 'Input3 real','Input4 real','Output real'])

 X_train = data_train[['Input1 real', 'Input2 real', 'Input3 real','Input4 real']]

 y_train = data_train['Output real']

 # Load testing dataset

 data_test = pd.read_csv(filename_test, delimiter=',', skiprows=9, names=['Input1 real', 'Input2 real', 'Input3 real','Input4 real','Output real'])

 X_test = data_test[['Input1 real', 'Input2 real', 'Input3 real','Input4 real']]

 y_test = data_test['Output real']

 # Train a linear regression model

 alphas = 2.0**np.arange(-18, 51)

 degrees = [1, 2, 3]

 best_alpha = None

 best_degree = None

 min_rmse = float('inf')

 min_mae=float('inf')

 max_r_squared=float('-inf')

 for degree in degrees:

  for alpha in alphas:

   polynomial_features = PolynomialFeatures(degree)

   X_poly_train = polynomial_features.fit_transform(X_train)

   X_poly_test = polynomial_features.transform(X_test)

   scaler = StandardScaler()

   X_poly_train_scaled = scaler.fit_transform(X_poly_train)

   X_poly_test_scaled = scaler.transform(X_poly_test)
```

```
 ridge = Ridge(alpha=alpha)
 ridge.fit(X_poly_train_scaled, y_train)
 y_pred = ridge.predict(X_poly_test_scaled)
 rmse = np.sqrt(mean_squared_error(y_test, y_pred))
 mae = mean_absolute_error(y_test, y_pred)
 r_squared = r2_score(y_test, y_pred)
 best_alpha=alpha
 best_degree=degree
 if rmse < min_rmse:
  min_rmse = rmse
best_alpha_rmse = alpha
best_degree_rmse = degree
if mae < min_mae:
  min_mae = mae
  best_alpha_mae = alpha
  best_degree_mae = degree
if r_squared> max_r_squared:
 max_r_squared = r_squared
 best_alpha_r = alpha
 best_degree_r = degree
sum_C=sum_C+best_alpha
sum_D=sum_D+best_degree
sum_rmse=sum_rmse+min_rmse
sum_mae=sum_mae+min_mae
sum_r_squared=sum_r_squared+max_r_squared
sum_best_c_rmse=sum_best_c_rmse+ best_alpha_rmse
sum_best_d_rmse=sum_best_d_rmse+ best_degree_rmse
sum_best_c_mae=sum_best_c_mae+ best_alpha_mae
sum_best_d_mae=sum_best_d_mae+ best_degree_mae
sum_best_c_r=sum_best_c_r+ best_alpha_r
sum_best_d_r=sum_best_d_r+ best_degree_r
```

avg_c=sum_C/5

avg_d=sum_D/5

avg_r_squared=r_squared/5

avg_c_rmse=sum_best_c_rmse/5

avg_d_rmse=sum_best_d_rmse/5

avg_c_mae=sum_best_c_mae/5

avg_d_mae=sum_best_d_mae/5

avg_c_r=sum_best_c_r/5

avg_d_r=sum_best_d_r/5

print(f'best c={avg_c}')

print(f'best degree={avg_d}')

print(f'AVG_RMSE: {avg_rmse}')

print(f'best_C_RMSE: {avg_c_rmse}')

print(f'AVG_D_RMSE: {avg_d_rmse}')

print(f'AVG_MAE: {avg_mae}')

print(f'best_C_RMSE: {avg_c_mae}')

print(f'AVG_D_RMSE: {avg_d_mae}')

print(f'AVG_R-squared: {avg_r_squared}')

print(f'best_C_RMSE: {avg_c_r}')

print(f'AVG_D_RMSE: {avg_d_r}')

output: **best c=1125899906842624.0**

best degree=3.0

AVG_RMSE: 4.890015724666586

best_C_RMSE: 0.09375

AVG_D_RMSE: 3.0

AVG_MAE: 6.007107438010093

best_C_RMSE: 1125899906842624.0

AVG_D_RMSE: 3.0

AVG_R-squared: -0.0005716908651410169

best_C_RMSE: 1125899906842624.0

AVG_D_RMSE: 3.0

Shivani

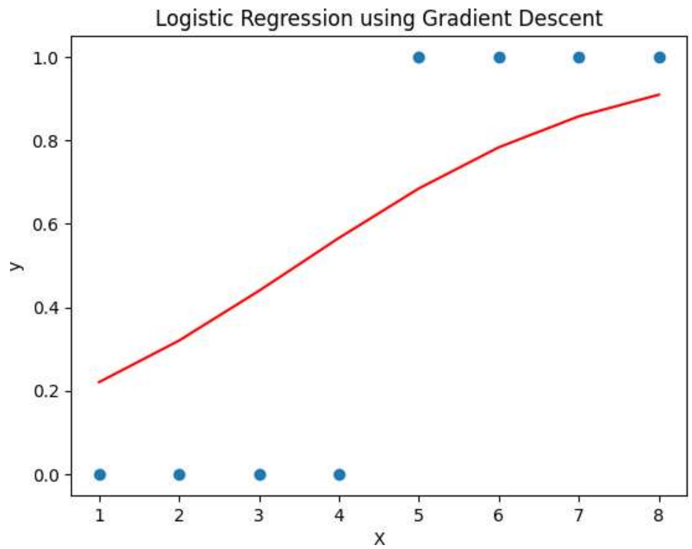
Narwariya

211112247

Question1:

```
import numpy as np
import matplotlib.pyplot as plt
X = np.array([1, 2, 3, 4, 5, 6, 7, 8])
y = np.array([0, 0, 0, 0, 1, 1, 1, 1])
X = X.reshape(-1, 1)
learning_rate = 0.01
n_iterations = 1000
X_b = np.c_[np.ones((8, 1)), X]
theta = np.random.randn(2, 1)

# Sigmoid function for logistic regression
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

# Implement gradient descent for logistic regression
for iteration in range(n_iterations):
    # Calculate predictions
    predictions = sigmoid(X_b.dot(theta))
    error = predictions - y.reshape(-1, 1)
    gradients = X_b.T.dot(error) / len(y)
    theta = theta - learning_rate * gradients

# Print the final parameters
print("Final Parameters (theta):", theta)
plt.scatter(X, y)
plt.plot(X, sigmoid(X_b.dot(theta)), color='red')
plt.xlabel('X')
plt.ylabel('y')
plt.title('Logistic Regression using Gradient Descent')
plt.show()
```

```
Final Parameters (theta): [[-1.77306504]
 [ 0.50973991]]
```



Question 2:

```
class PerformanceMetrics:
    @staticmethod
    def accuracy(y_true, y_pred):
        correct = sum(1 for true, pred in zip(y_true, y_pred) if true == pred)
        return correct / len(y_true)

    @staticmethod
    def precision(y_true, y_pred):
        true_positives = sum(1 for true, pred in zip(y_true, y_pred) if true
== 1 and pred == 1)
        predicted_positives = sum(1 for pred in y_pred if pred == 1)
        return true_positives / predicted_positives if predicted_positives !=
0 else 0

    @staticmethod
    def recall(y_true, y_pred):
        true_positives = sum(1 for true, pred in zip(y_true, y_pred) if true
== 1 and pred == 1)
        actual_positives = sum(1 for true in y_true if true == 1)
        return true_positives / actual_positives if actual_positives != 0 else
0

    @staticmethod
    def f1_score(y_true, y_pred):
        precision = PerformanceMetrics.precision(y_true, y_pred)
        recall = PerformanceMetrics.recall(y_true, y_pred)
        return 2 * (precision * recall) / (precision + recall) if (precision +
recall) != 0 else 0

    @staticmethod
```

```
    def true_positive_rate(conf_matrix):
        true_positive, false_positive, false_negative, true_negative =
conf_matrix
        return true_positive / (true_positive + false_negative) if
(true_positive + false_negative) != 0 else 0

    @staticmethod
    def false_positive_rate(conf_matrix):
        true_positive, false_positive, false_negative, true_negative =
conf_matrix
        return false_positive / (false_positive + true_negative) if
(false_positive + true_negative) != 0 else 0
    def gini(y_true, y_pred):
        n = len(y_true)
        sorted_indices = np.argsort(y_pred)
        y_true_sorted = y_true[sorted_indices]
        sorted_gini = y_true_sorted.cumsum().sum() / y_true_sorted.sum()
        gini_index = 1 - 2 * sorted_gini / n
        return gini_index

    @staticmethod
    def auc(y_true, y_pred):
        sorted_indices = np.argsort(y_pred)
        y_true_sorted = y_true[sorted_indices]
        n = len(y_true)
        tpr = np.zeros(n+1)
        fpr = np.zeros(n+1)
        tp = 0
        fp = 0
        for i in range(n):
            if y_true_sorted[i] == 1:
                tp += 1
            else:
                fp += 1
            tpr[i+1] = tp / sum(y_true)
            fpr[i+1] = fp / (n - sum(y_true))
        auc = np.trapz(tpr, fpr)
        return auc
    @staticmethod
    def confusion_matrix(y_true, y_pred):
        true_positive = sum(1 for true, pred in zip(y_true, y_pred) if true ==
1 and pred == 1)
        true_negative = sum(1 for true, pred in zip(y_true, y_pred) if true ==
0 and pred == 0)
        false_positive = sum(1 for true, pred in zip(y_true, y_pred) if true
== 0 and pred == 1)
        false_negative = sum(1 for true, pred in zip(y_true, y_pred) if true
== 1 and pred == 0)
```

```python
        return true_positive, false_positive, false_negative, true_negative

# Assuming y_test contains the true labels and y_pred contains the predicted labels
y_true = np.array([1, 1, 1, 1])  # True labels for test data
y_pred = np.array([1, 0, 0, 1])  # Predicted labels for test data

# Calculate performance metrics
accuracy = PerformanceMetrics.accuracy(y_true, y_pred)
precision = PerformanceMetrics.precision(y_true, y_pred)
recall = PerformanceMetrics.recall(y_true, y_pred)
f1_score = PerformanceMetrics.f1_score(y_true, y_pred)
conf_matrix = PerformanceMetrics.confusion_matrix(y_true, y_pred)
tpr = PerformanceMetrics.true_positive_rate(conf_matrix)
fpr = PerformanceMetrics.false_positive_rate(conf_matrix)
gini_index = PerformanceMetrics.gini(y_true, y_pred)
# tpr, far = PerformanceMetrics.roc_curve(y_true, y_pred)
auc_score = PerformanceMetrics.auc(y_true, y_pred)
print("Area Under the Curve (AUC):", auc_score)

print("Gini Index:", gini_index)
# print("True Positive Rate (TPR):", tpr)
# print("False Acceptance Rate (FAR):", far)
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1_score)
print("Confusion Matrix:", conf_matrix)
print("True Positive Rate (TPR):", tpr)
print("False Positive Rate (FPR):", fpr)
```

Area Under the Curve (AUC): nan

Gini Index: -0.25

Accuracy: 0.5

Precision: 1.0

Recall: 0.5

F1 Score: 0.6666666666666666

Confusion Matrix: (2, 0, 2, 0)

True Positive Rate (TPR): 0.5

False Positive Rate (FPR): 0

C:\Users\rinki\AppData\Local\Temp\ipykernel_15060\4219842578.py:57: RuntimeWarning: invalid value encountered in scalar divide

fpr[i+1] = fp / (n - sum(y_true)

Question 3:

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
import pandas as pd

# Load dataset

file_path=r"C:/Users/rinki/OneDrive/Desktop/test
jupyter/diabetes/diabetes.dat"
data = pd.read_csv(file_path, delimiter=',', skiprows=7, names=['Age',
'Deficit','C_peptide'])
# Split dataset 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)

# Create SVM classifier
svm_classifier = SVC(kernel='linear')

# Train SVM classifier
svm_classifier.fit(X_train, y_train)

# Make predictions
y_pred = svm_classifier.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 1.0
```

Assignment 3:

1. Write a python program of naive bayes classifier for iris dataset classification .

```
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

# Loading Iris dataset

iris_df = pd.read_csv("C:/Users/rinki/Downloads/irisd/irisf/iris.csv")

X = iris_df.drop('species', axis=1)

y = iris_df['species']

# Splitting dataset into training set and test set

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Creating Naive Bayes classifier

clf = GaussianNB()

# Training Naive Bayes classifier

clf.fit(X_train, y_train)

# Predicting the response for test dataset

y_pred = clf.predict(X_test)

# Model Accuracy

print("Accuracy:", accuracy_score(y_test, y_pred))
```

Output:

```
Accuracy: 0.9777777777777777
```

2. Write a python program of naive bayes classifier for Play Tennis dataset classification

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.naive_bayes import GaussianNB

from sklearn.metrics import accuracy_score
```

```
from sklearn import preprocessing
# Loading dataset
data = pd.read_csv("C:/Users/rinki/Downloads/playt/play/PlayTennis.csv")
# Preprocessing: Convert categorical variables into numerical values
le = preprocessing.LabelEncoder()
for column in data.columns:
    data[column] = le.fit_transform(data[column])
# Splitting dataset into features and target variable
X = data.iloc[:, :-1]
y = data.iloc[:, -1]
# Splitting dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
# Creating Naive Bayes classifier
clf = GaussianNB()
# Training Naive Bayes classifier
clf.fit(X_train, y_train)
# Predicting the response for test dataset
y_pred = clf.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
```

Output:

```
Accuracy: 0.6
```

3. Write a python program of naive bayes classifier for Large Movie Review Dataset dataset classification

```
import nltk
from nltk.corpus import movie_reviews
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report, accuracy_score
from sklearn.model_selection import train_test_split
# Download the movie reviews dataset
```

```
nltk.download('movie_reviews')
# Load movie reviews data
documents = [(list(movie_reviews.words(fileid)), category)
        for category in movie_reviews.categories()
        for fileid in movie_reviews.fileids(category)]
# Shufle the documents
import random
random.shufle(documents)
# Split the data into features and labels
X = [" ".join(words) for words, label in documents]
y = [label for words, label in documents]
# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Vectorize the text data
vectorizer = TfidfVectorizer(max_features=5000)  # Adjust max_features as needed
X_train_counts = vectorizer.fit_transform(X_train)
X_test_counts = vectorizer.transform(X_test)
# Train the Naive Bayes classifier
naive_bayes = MultinomialNB()
naive_bayes.fit(X_train_counts, y_train)
# Make predictions on the test set
y_pred = naive_bayes.predict(X_test_counts)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
# Generate classification report
report = classification_report(y_test, y_pred)
print("\nClassification Report:\n", report)
```

Output:

```
Accuracy: 0.81

Classification Report:
              precision    recall  f1-score   support

         neg       0.76      0.88      0.82       192
         pos       0.87      0.75      0.80       208

    accuracy                           0.81       400
   macro avg       0.82      0.81      0.81       400
weighted avg       0.82      0.81      0.81       400
```

Assignment 4

Question 1: Decision Tree classification of Play Tennis dataset

```
import pandas as pd

from sklearn.tree import DecisionTreeClassifier

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score

from sklearn import preprocessing

# Loading dataset

data = pd.read_csv("C:/Users/rinki/Downloads/playt/play/PlayTennis.csv")

# Preprocessing: Convert categorical variables into numerical values

le = preprocessing.LabelEncoder()

for column in data.columns:

    data[column] = le.fit_transform(data[column])

# Splitting dataset into features and target variable

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

y = data.iloc[:, -1]

# Splitting dataset into training set and test set

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Creating Decision Tree classifier

clf = DecisionTreeClassifier()

# Training Decision Tree classifier

clf = clf.fit(X_train,y_train)

# Predicting the response for test dataset

y_pred = clf.predict(X_test)

# Model Accuracy

print("Accuracy:", accuracy_score(y_test, y_pred))
```

Output: `Accuracy: 0.6`