Institute of Distance and Open Learning

Vidya Nagari, Kalina, Santacruz East - 400098

A Practical Journal Submitted in fulfillment

of the degree of

MASTER OF SCIENCE

IN

COMPUTER SCIENCE

YEAR 2023-2024

Part 1 - Semester 2

PSCSP201

"Machine Intelligence"

By

Mr. YADAV YOGESH RAMASHREY MEENADEVI

Application ID:- 32579

Seat No:- 4500070

Under the Guidance of

Prof. Sujatha Iyer

Institute of Distance and Open Learning

Vidya Nagari, kalina, Santacruz East – 400098

CERTIFICATE

This is to certify that, this practical journal entitled **"Machine Intelligence"** is a record of work carried out by **Mr. YADAV YOGESH RAMASHREY MEENADEVI**, student of **Master of Science in Computer Science Part 2** class and is submitted to University of Mumbai, in partial fulfillment of the requirement for the award of the degree of **Master of Science in Computer Science.** The practical journal has been approved.

----------	----------	----------
Guide	External Examiner	Coordinator- M.Sc.CS

Index

Sr.No	Practical	Signature
1	Import the required python packages	
2	Implement multiple regression model on a standard data set	
3	Implement Logistic Regression	
4	Fit a classification model using K Nearest Neighbour (KNN) Algorithm on a given data set.	
5	Use bootstrap to give an estimate of a given statistic. Example of how bootstrap samples are created and used to estimate a statistic of interest.	
6	For a given data set, split the data into two training and testing and fit the following on the training set: (i) Linear model using least squares (ii) Ridge regression model (iii) Lasso model (iv) PCR model (v) PLS model	
7	For a given data set, perform the following: Perform the polynomial regression and make a plot of the resulting polynomial fit to the data.	
8	Decision Tree	
9	For a given data set, split the dataset into training and testing. Fit the following models on the training set and evaluate the performance on the test set: (i) Boosting and Bagging (ii) Random Forest	

Practical 1

Aim: Import the required python packages

Source Code:- Step 1: Import the required python packages

```
# Import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from pandas.core.common import random_state
from sklearn.linear_model import LinearRegression
# Get dataset
df_sal = pd.read_csv('/content/Salary_Data.csv')
df_sal.head()
# Describe data
df_sal.describe()
# Data distribution
plt.title('Salary Distribution Plot')
sns.distplot(df_sal['Salary'])
plt.show()
# Relationship between Salary and Experience
plt.scatter(df_sal['YearsExperience'], df_sal['Salary'], color = 'lightcoral')
plt.title('Salary vs Experience')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.box(False)
plt.show()
# Splitting variables
X = df_sal.iloc[:, :1]  # independent
y = df_sal.iloc[:, 1:]  # dependent
# Splitting dataset into test/train
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
# Regressor model
regressor = LinearRegression()
regressor.fit(X_train, y_train)
# Prediction result
y_pred_test = regressor.predict(X_test)     # predicted value of y_test
y_pred_train = regressor.predict(X_train)   # predicted value of y_train
# Prediction on training set
plt.scatter(X_train, y_train, color = 'lightcoral')
plt.plot(X_train, y_pred_train, color = 'firebrick')
plt.title('Salary vs Experience (Training Set)')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.legend(['X_train/Pred(y_test)', 'X_train/y_train'], title = 'Sal/Exp', loc='best', facecolor='white')
plt.box(False)
plt.show()
```

Output:


Salary vs Experience (Training Set)
Sal/Exp
X_train/Pred(y_test)
X_train/y_train
120000
100000
80000
60000
40000
Salary
2
4
6
8
10
Years of Experience

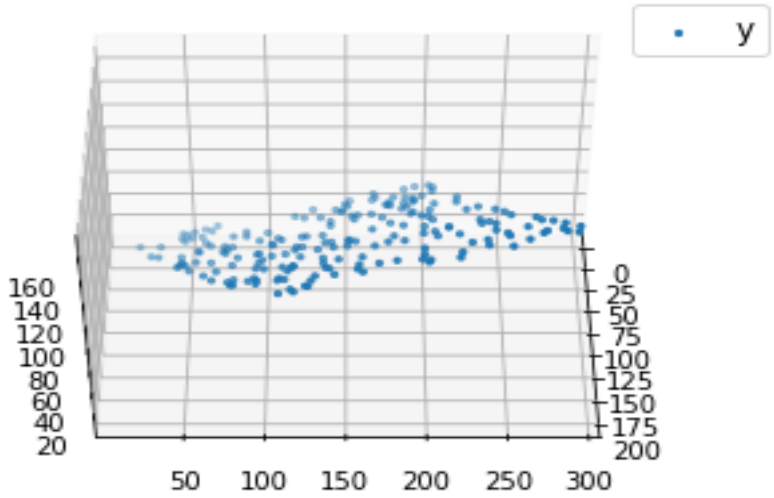
Practical 2

Aim: Implement multiple regression model on a standard data set

Source Code:-

```
import numpy as np
import matplotlib as mpl
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt

def generate_dataset(n):
        x = []
        y = []
        random_x1 = np.random.rand()
        random_x2 = np.random.rand()
        for i in range(n):
                x1 = i
                x2 = i/2 + np.random.rand()*n
                x.append([1, x1, x2])
                y.append(random_x1 * x1 + random_x2 * x2 + 1)
        return np.array(x), np.array(y)

x, y = generate_dataset(200)

mpl.rcParams['legend.fontsize'] = 12

fig = plt.figure()
ax = fig.add_subplot(projection ='3d')
ax.scatter(x[:, 1], x[:, 2], y, label ='y', s = 5)
ax.legend()
ax.view_init(45, 0)
plt.show()
```

Output:



Practical 3

Aim: Implement Logistic Regression

Input:

```
# Import necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, roc_curve, auc

# Load the diabetes dataset
diabetes = load_diabetes()
X, y = diabetes.data, diabetes.target

# Convert the target variable to binary (1 for diabetes, 0 for no diabetes)
y_binary = (y > np.median(y)).astype(int)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
        X, y_binary, test_size=0.2, random_state=42)

# Standardize features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train the Logistic Regression model
model = LogisticRegression()
model.fit(X_train, y_train)

# Evaluate the model
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy: {:.2f}%".format(accuracy * 100))

# evaluate the model
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

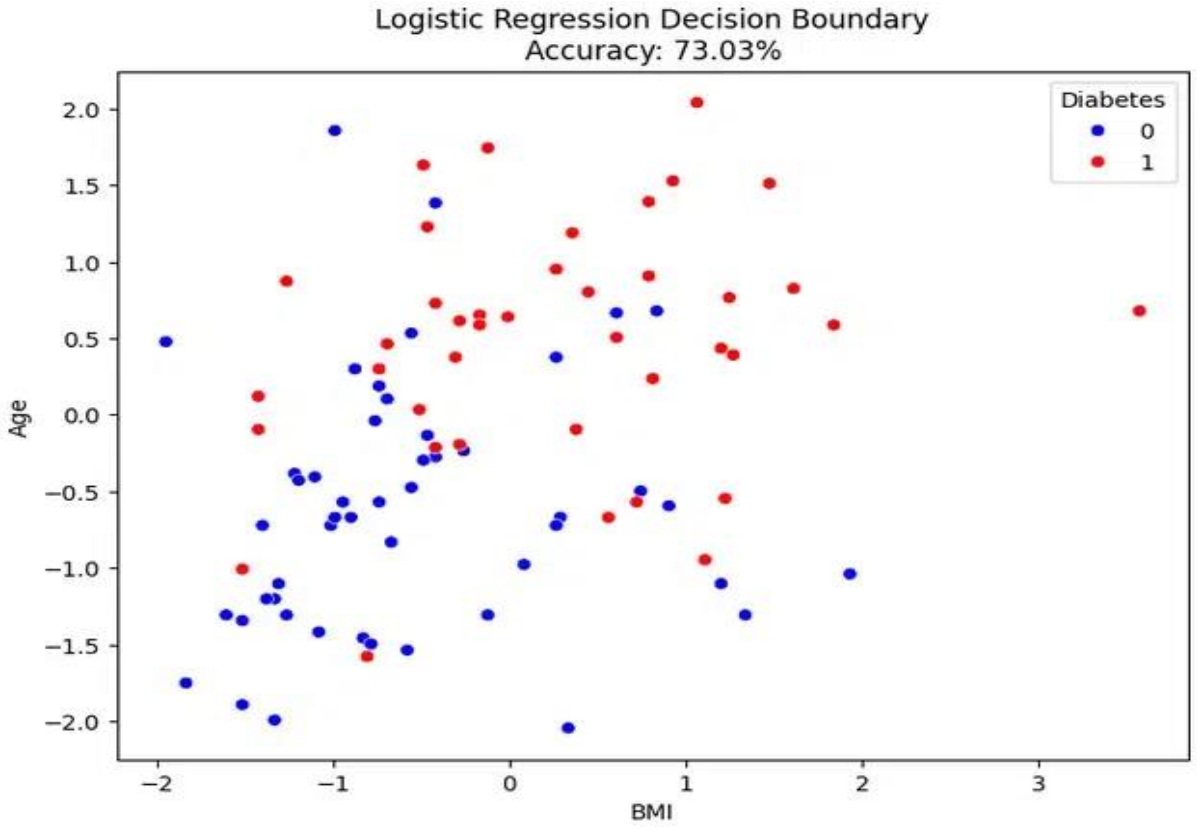
output:

Confusion Matrix:
[[36 13]
[11 29]]

Classification Report:

	precision	recall	f1-score	support
0	0.77	0.73	0.75	49
1	0.69	0.72	0.71	40
accuracy			0.73	89
macro avg	0.73	0.73	0.73	89
weighted avg	0.73	0.73	0.73	89

```
# Visualize the decision boundary with accuracy information
plt.figure(figsize=(8, 6))
sns.scatterplot(x=X_test[:, 2], y=X_test[:, 8], hue=y_test, palette={
                                        0: 'blue', 1: 'red'}, marker='o')
plt.xlabel("BMI")
plt.ylabel("Age")
plt.title("Logistic Regression Decision Boundary\nAccuracy: {:.2f}%".format(
          accuracy * 100))
plt.legend(title="Diabetes", loc="upper right")
plt.show()
```



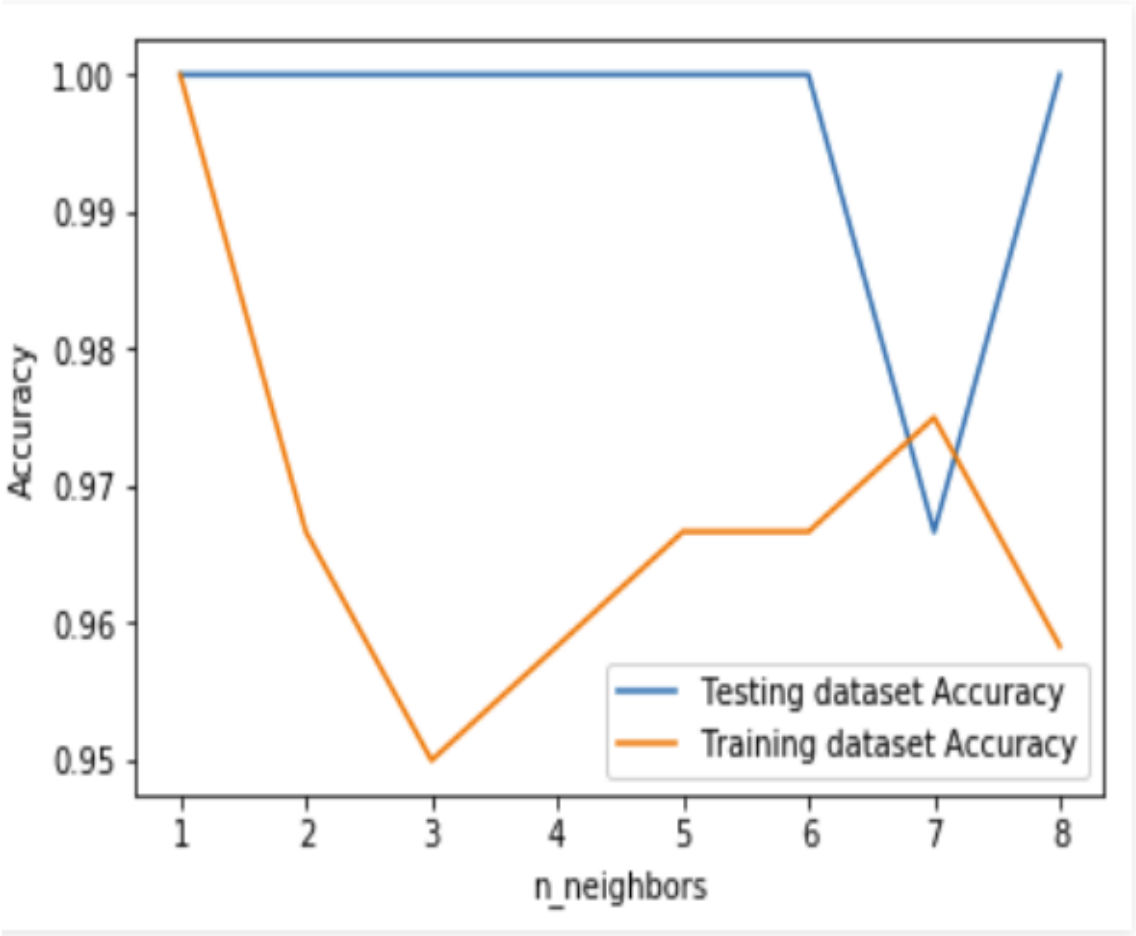
Practical 4

Aim: Fit a classification model using K Nearest Neighbour (KNN) Algorithm on a given data set.

Source Code:

```
# Import necessary modules
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
import numpy as np
import matplotlib.pyplot as plt

irisData = load_iris()

# Create feature and target arrays
X = irisData.data
y = irisData.target

# Split into training and test set
X_train, X_test, y_train, y_test = train_test_split(
                                X, y, test_size = 0.2, random_state=42)

neighbors = np.arange(1, 9)
train_accuracy = np.empty(len(neighbors))
test_accuracy = np.empty(len(neighbors))

# Loop over K values
for i, k in enumerate(neighbors):
        knn = KNeighborsClassifier(n_neighbors=k)
        knn.fit(X_train, y_train)

        # Compute training and test data accuracy
        train_accuracy[i] = knn.score(X_train, y_train)
        test_accuracy[i] = knn.score(X_test, y_test)

# Generate plot
plt.plot(neighbors, test_accuracy, label = 'Testing dataset Accuracy')
plt.plot(neighbors, train_accuracy, label = 'Training dataset Accuracy')

plt.legend()
plt.xlabel('n_neighbors')
plt.ylabel('Accuracy')
plt.show()
```

Output:


1.00
0.99
0.98
0.97
0.96
0.95
Accuracy
1
2
3
4
5
6
7
8
n_neighbors
Testing dataset Accuracy
Training dataset Accuracy

Practical 5

Aim: Use bootstrap to give an estimate of a given statistic. Example of how bootstrap samples are created and used to estimate a statistic of interest.

Let's say we have a small dataset of 5 observations:
Original Data: [3, 4, 5, 6, 7]
Create bootstrap samples by resampling with replacement:
We'll create 3 bootstrap samples of size 5 by randomly drawing observations from the original data with replacement.
Each bootstrap sample will have the same size as the original dataset.
Bootstrap Sample 1: [5, 6, 3, 4, 7]
Bootstrap Sample 2: [4, 3, 6, 4, 6]
Bootstrap Sample 3: [7, 5, 7, 3, 4]
Calculate the statistic of interest (median) for each bootstrap sample:
Bootstrap Sample 1 median: 5
Bootstrap Sample 2 median: 4
Bootstrap Sample 3 median: 5
Repeat steps 1 and 2 many times (e.g., 10,000 times):
By repeating the process of creating bootstrap samples and calculating the median, we can build an empirical sampling distribution of the median.
Use the empirical sampling distribution to calculate confidence intervals or perform hypothesis tests:
For example, if we want to construct a 95% confidence interval for the median, we can find the 2.5th and 97.5th percentiles of the empirical sampling distribution of the median.
Let's say the 2.5th percentile is 4, and the 97.5th percentile is 6.
Then, the 95% confidence interval for the median would be [4, 6].

Example of Using Bootstrapping to Create Confidence Intervals

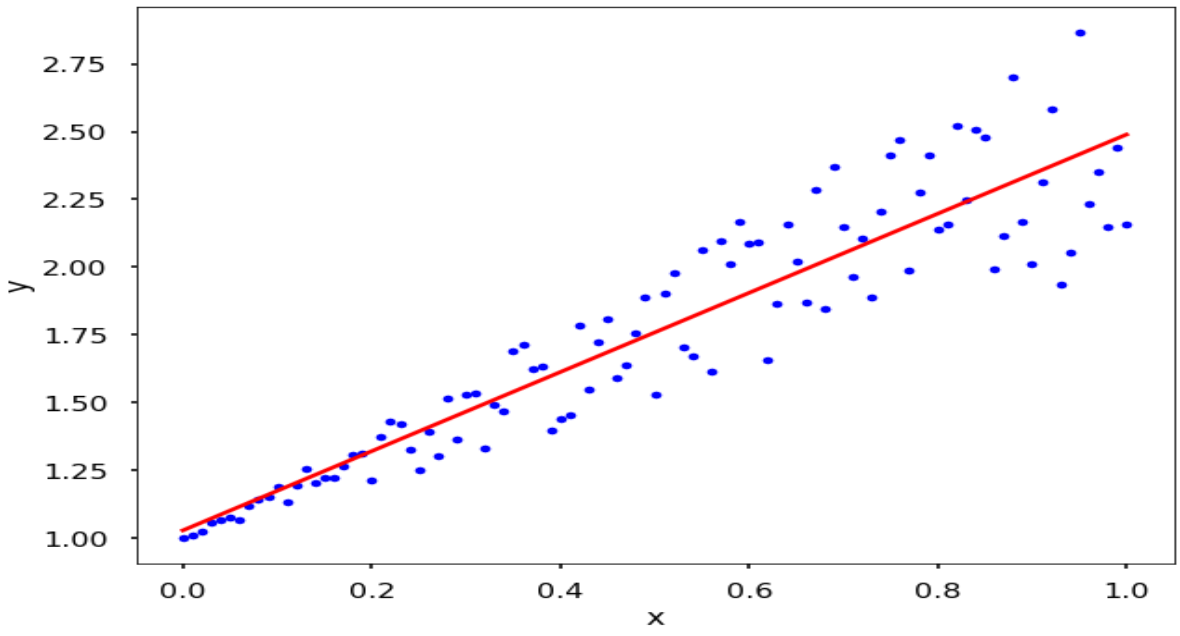
Let's say we have a small sample of data representing the heights (in inches) of 10 individuals:
Heights = [65.2, 67.1, 68.5, 69.3, 70.0, 71.2, 72.4, 73.1, 74.5, 75.8]
We want to estimate the 95% confidence interval for the mean height in the population using bootstrapping.
Here are the steps we would follow:
Calculate the sample mean from the original data:
Sample mean = (65.2 + 67.1 + 68.5 + 69.3 + 70.0 + 71.2 + 72.4 + 73.1 + 74.5 + 75.8) / 10 = 70.71 inches
Create a large number of bootstrap samples from the original data by resampling with replacement. For example, let's create 10,000 bootstrap samples, each of size 10.
For each bootstrap sample, calculate the mean height.
After computing the means for all 10,000 bootstrap samples, we now have an empirical bootstrap sampling distribution of the mean.
From this empirical bootstrap sampling distribution, we can determine the 95% confidence interval by finding the 2.5th and 97.5th percentiles of the distribution.
Let's say the 2.5th percentile is 69.8 inches, and the 97.5th percentile is 71.6 inches.
Then, the 95% confidence interval for the mean height is [69.8, 71.6] inches.
This confidence interval means that if we were to repeat the process of taking a sample of size 10 and constructing a bootstrap confidence interval many times, 95% of those intervals would contain the true population mean height.
The key advantage of bootstrapping in this example is that it does not require any assumptions about the underlying distribution of heights in the population. It relies solely on the information contained in the original sample data.

Practical 6

Aim: For a given data set, split the data into two training and testing and fit the following on the training set: (i) Linear model using least squares (ii) Ridge regression model (iii) Lasso model (iv) PCR model (v) PLS model

(i) Linear model using least squares

```
import numpy as np
from scipy import optimize
import matplotlib.pyplot as plt
plt.style.use('seaborn-poster')
# generate x and y
x = np.linspace(0, 1, 101)
y = 1 + x + x * np.random.random(len(x))
# assemble matrix A
A = np.vstack([x, np.ones(len(x))]).T

# turn y into a column vector
y = y[:, np.newaxis]
# Direct least square regression
alpha = np.dot((np.dot(np.linalg.inv(np.dot(A.T,A)),A.T)),y)
print(alpha)
# plot the results
plt.figure(figsize = (10,8))
plt.plot(x, y, 'b.')
plt.plot(x, alpha[0]*x + alpha[1], 'r')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```



ii) Ridge regression model

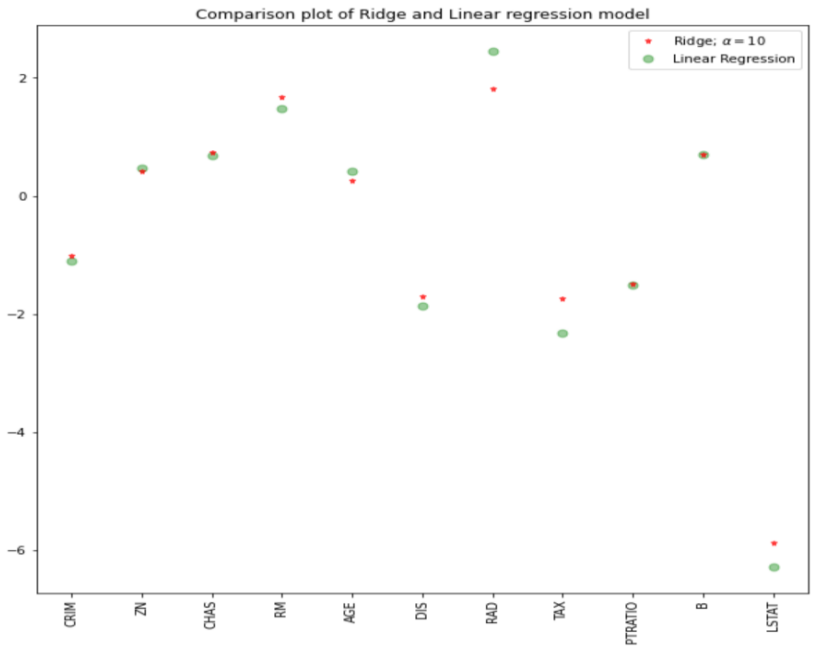
```
#Model
lr = LinearRegression()

#Fit model
lr.fit(X_train, y_train)

#predict
#prediction = lr.predict(X_test)

#actual
actual = y_test

train_score_lr = lr.score(X_train, y_train)
test_score_lr = lr.score(X_test, y_test)

print("The train score for lr model is {}".format(train_score_lr))
print("The test score for lr model is {}".format(test_score_lr))


#Ridge Regression Model
ridgeReg = Ridge(alpha=10)

ridgeReg.fit(X_train,y_train)

#train and test scorefor ridge regression
train_score_ridge = ridgeReg.score(X_train, y_train)
test_score_ridge = ridgeReg.score(X_test, y_test)

print("\nRidge Model.....................................\n")
print("The train score for ridge model is {}".format(train_score_ridge))
print("The test score for ridge model is {}".format(test_score_ridge))
```

Output:

```
The train score for lr model is 0.7859187129718976
The test score for lr model is 0.7672379770848983

Ridge Model.....................................

The train score for ridge model is 0.7844233397895741
The test score for ridge model is 0.7696722158755336
```

```
plt.figure(figsize = (10, 10))
plt.plot(features,ridgeReg.coef_,alpha=0.7,linestyle='none',marker='*',markersize=5,color='red',label=r'Ridge; $\alpha = 10$',zorder=7)
#plt.plot(rr100.coef_,alpha=0.5,linestyle='none',marker='d',markersize=6,color='blue',label=r'Ridge; $\alpha = 100$')
```

plt.plot(features,lr.coef_,alpha=0.4,linestyle='none',marker='o',markersize=7,color='green',label='Linear Regression')
plt.xticks(rotation = 90)
plt.legend()
plt.show()



iii) Lasso model

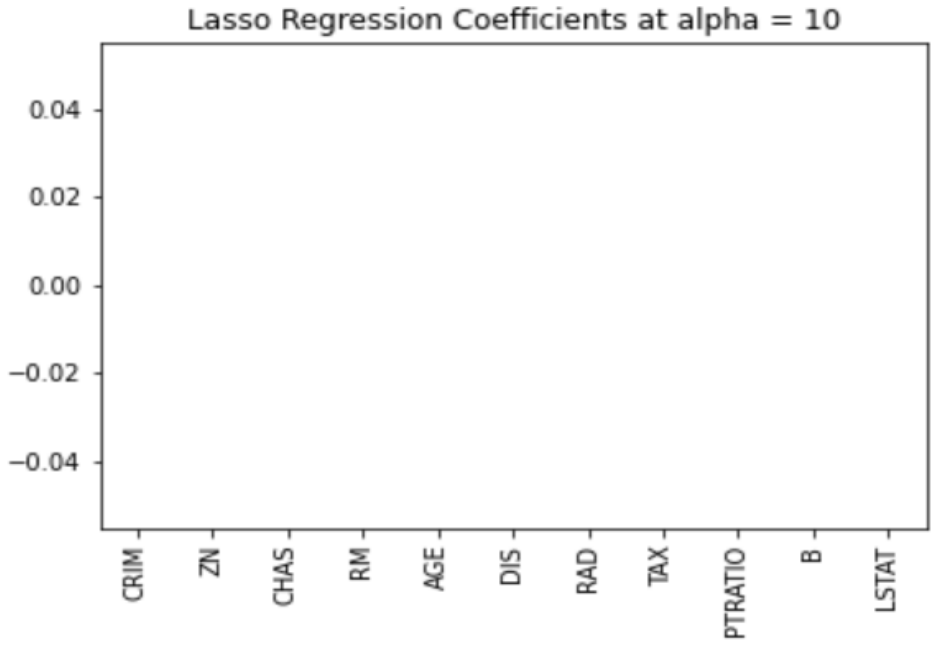
#Lasso regression model
print("\nLasso Model.....\n")
lasso = Lasso(alpha = 10)
lasso.fit(X_train,y_train)
train_score_ls =lasso.score(X_train,y_train)
test_score_ls =lasso.score(X_test,y_test)

print("The train score for ls model is {}".format(train_score_ls))
print("The test score for ls model is {}".format(test_score_ls**))**

Output:

```
Lasso Model.....................................

The train score for ls model is 0.0
The test score for ls model is -0.0030704836212473996
```

pd.Series(lasso.coef_, features).sort_values(ascending = True).plot(kind = "bar")



iv) PCR model

```
%matplotlib inline
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import scale
from sklearn import model_selection
from sklearn.decomposition import PCA
from sklearn.linear_model import LinearRegression
from sklearn.cross_decomposition import PLSRegression, PLSSVD
from sklearn.metrics import mean_squared_error
df = pd.read_csv('Hitters.csv').dropna().drop('Player', axis=1)
df.info()
dummies = pd.get_dummies(df[['League', 'Division', 'NewLeague']])
y = df.Salary

# Drop the column with the independent variable (Salary), and columns for which we created dummy variables
X_ = df.drop(['Salary', 'League', 'Division', 'NewLeague'], axis=1).astype('float64')

# Define the feature set X.
X = pd.concat([X_, dummies[['League_N', 'Division_W', 'NewLeague_N']]], axis=1)
pca = PCA()
X_reduced = pca.fit_transform(scale(X))
pd.DataFrame(pca.components_.T).loc[:4,:5]
# 10-fold CV, with shuffle
n = len(X_reduced)
kf_10 = model_selection.KFold( n_splits=10, shuffle=True, random_state=1)
regr = LinearRegression()
mse = []
```

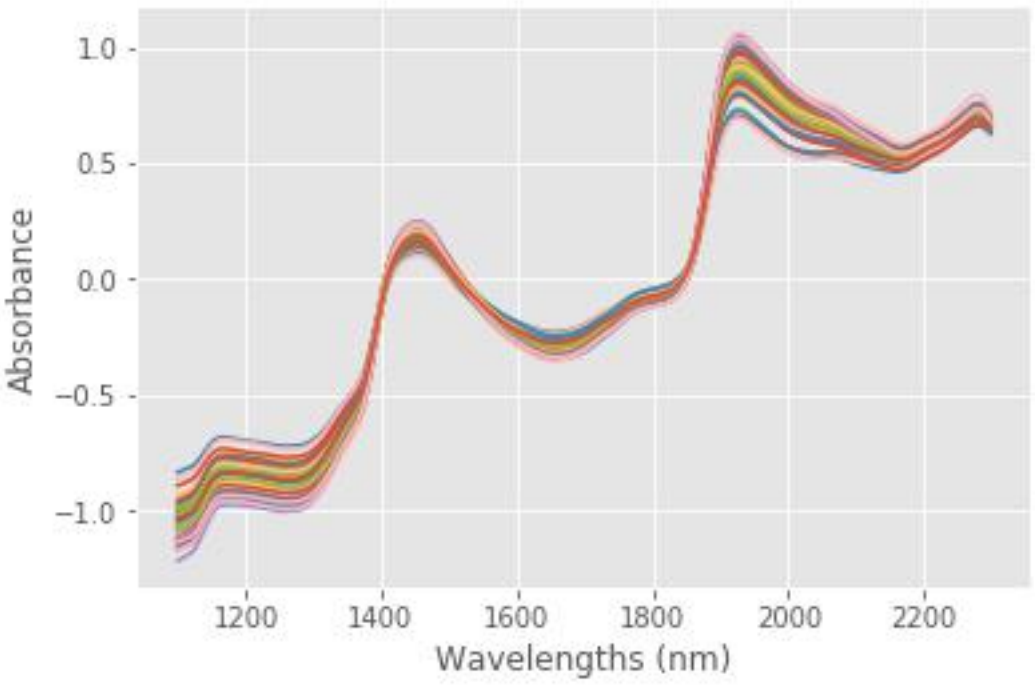
```
# Calculate MSE with only the intercept (no principal components in regression)
score = -1*model_selection.cross_val_score(regr, np.ones((n,1)), y.ravel(), cv=kf_10,
scoring='neg_mean_squared_error').mean()
mse.append(score)

# Calculate MSE using CV for the 19 principle components, adding one component at the time.
for i in np.arange(1, 20):
    score = -1*model_selection.cross_val_score(regr, X_reduced[:,:i], y.ravel(), cv=kf_10,
scoring='neg_mean_squared_error').mean()
    mse.append(score)

# Plot results
plt.plot(mse, '-v')
plt.xlabel('Number of principal components in regression')
plt.ylabel('MSE')
plt.title('Salary')
plt.xlim(xmin=-1);
pca2 = PCA()

# Split into training and test sets
X_train, X_test , y_train, y_test = model_selection.train_test_split(X, y, test_size=0.5, random_state=1)

# Scale the data
X_reduced_train = pca2.fit_transform(scale(X_train))
n = len(X_reduced_train)

# 10-fold CV, with shuffle
kf_10 = model_selection.KFold( n_splits=10, shuffle=True, random_state=1)

mse = []

# Calculate MSE with only the intercept (no principal components in regression)
score = -1*model_selection.cross_val_score(regr, np.ones((n,1)), y_train.ravel(), cv=kf_10,
scoring='neg_mean_squared_error').mean()
mse.append(score)

# Calculate MSE using CV for the 19 principle components, adding one component at the time.
for i in np.arange(1, 20):
    score = -1*model_selection.cross_val_score(regr, X_reduced_train[:,:i], y_train.ravel(), cv=kf_10,
scoring='neg_mean_squared_error').mean()
    mse.append(score)

plt.plot(np.array(mse), '-v')
plt.xlabel('Number of principal components in regression')
plt.ylabel('MSE')
plt.title('Salary')
plt.xlim(xmin=-1);
```

(v) PLS model

```
from sys import stdout
import numpy as np
```

```
import pandas as pd
import matplotlib.pyplot as plt

from scipy.signal import savgol_filter

from sklearn.cross_decomposition import PLSRegression
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import mean_squared_error, r2_score
data = pd.read_csv("../input/peach-nir-spectra-brix-values/peach_spectrabrixvalues.csv")
data.head()
y = data['Brix'].values
X = data.values[:, 1:]
y.shape
X.shape
# Plot the data
wl = np.arange(1100, 2300, 2)
print(len(wl)
with plt.style.context('ggplot'):
    plt.plot(wl, X.T)
    plt.xlabel("Wavelengths (nm)")
    plt.ylabel("Absorbance")
```

Output:



Practical 7

Aim: For a given data set, perform the following: Perform the polynomial regression and make a plot of the resulting polynomial fit to the data.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
# Importing the dataset
datas = pd.read_csv('data.csv')
datas
```

	sno	Temperature	Pressure
0	1	0	0.0002
1	2	20	0.0012
2	3	40	0.0060
3	4	60	0.0300
4	5	80	0.0900
5	6	100	0.2700

```
X = datas.iloc[:, 1:2].values
y = datas.iloc[:, 2].values
# Features and the target variables
X = datas.iloc[:, 1:2].values
y = datas.iloc[:, 2].values
# Fitting Linear Regression to the dataset
from sklearn.linear_model import LinearRegression
lin = LinearRegression()
lin.fit(X, y)
# Fitting Polynomial Regression to the dataset
from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(degree=4)
X_poly = poly.fit_transform(X)
poly.fit(X_poly, y)
lin2 = LinearRegression()
lin2.fit(X_poly, y)
# Visualising the Linear Regression results
plt.scatter(X, y, color='blue')
plt.plot(X, lin.predict(X), color='red')
plt.title('Linear Regression')
plt.xlabel('Temperature')
plt.ylabel('Pressure')
plt.show()
```

Practical 8

Aim: Decision Tree

```
# Load libraries
import pandas as pd
from sklearn.tree import DecisionTreeClassifier # Import Decision Tree Classifier
from sklearn.model_selection import train_test_split # Import train_test_split function
from sklearn import metrics #Import scikit-learn metrics module for accuracy calculation
col_names = ['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi', 'pedigree', 'age', 'label']
# load dataset
pima = pd.read_csv("diabetes.csv", header=None, names=col_names)
pima.head()
```

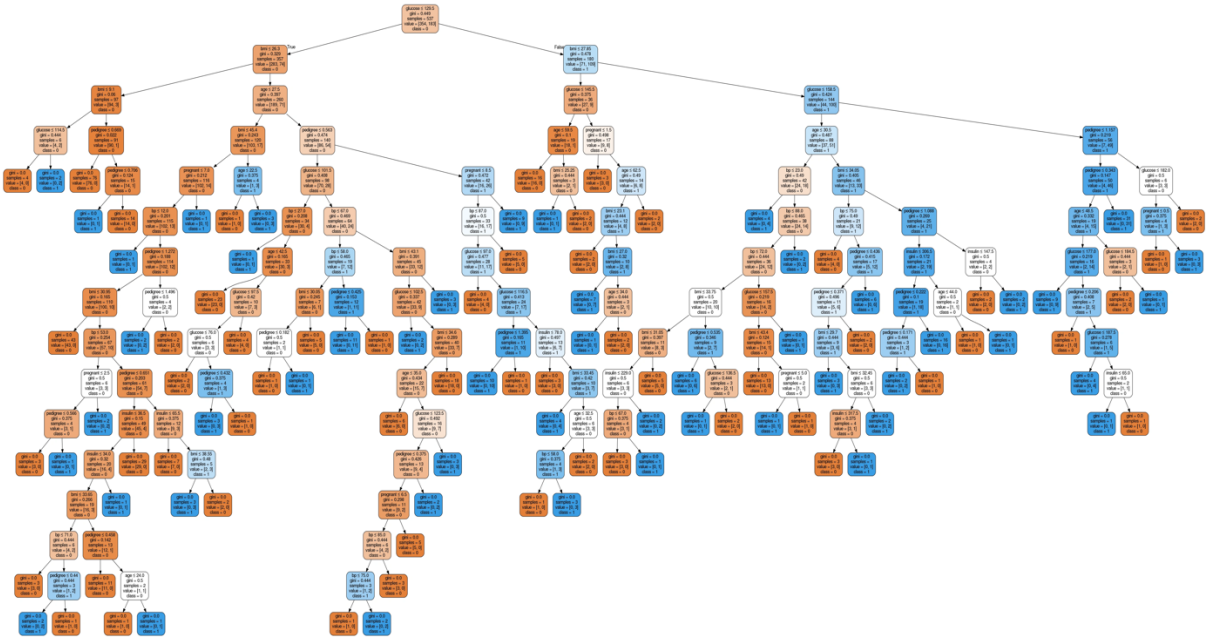
Output:

	pregnant	glucose	bp	skin	insulin	bmi	pedigree	age	label
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

```
#split dataset in features and target variable
feature_cols = ['pregnant', 'insulin', 'bmi', 'age','glucose','bp','pedigree']
X = pima[feature_cols] # Features
y = pima.label # Target variable

# Split 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=1) # 70% training and
30% test
# Create Decision Tree classifer object
clf = DecisionTreeClassifier()

# Train Decision Tree Classifer
clf = 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))
Accuracy: 0.6753246753246753
from sklearn.tree import export_graphviz
from sklearn.externals.six import StringIO
from IPython.display import Image
import pydotplus
dot_data = StringIO()
export_graphviz(clf, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True,feature_names = feature_cols,class_names=['0','1'])
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('diabetes.png')
Image(graph.create_png())
```

Output:



Practical 9

Aim: For a given data set, split the dataset into training and testing. Fit the following models on the training set and evaluate the performance on the test set: (i) Boosting and Bagging (ii) Random Forest

i) Boosting and Bagging

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline
sns.set_style("whitegrid")
plt.style.use("fivethirtyeight")
df = pd.read_csv("../input/pima-indians-diabetes-database/diabetes.csv")
df.head()
df.info()
df.isnull().sum()
pd.set_option('display.float_format', '{:.2f}'.format)
df.describe()

categorical_val = []
continous_val = []
for column in df.columns:
#     print('==============================')
#     print(f"{column} : {df[column].unique()}")
    if len(df[column].unique()) <= 10:
        categorical_val.append(column)
    else:
        continous_val.append(column)
df.columns

# How many missing zeros are mising in each feature
feature_columns = [
    'Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness',
    'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age'
]

for column in feature_columns:
    print("==========================================")
    print(f"{column} ==> Missing zeros : {len(df.loc[df[column] == 0])}")
from sklearn.impute import SimpleImputer


fill_values = SimpleImputer(missing_values=0, strategy="mean", copy=False)
df[feature_columns] = fill_values.fit_transform(df[feature_columns])

for column in feature_columns:
    print("==========================================")
    print(f"{column} ==> Missing zeros : {len(df.loc[df[column] == 0])}")

from sklearn.model_selection import train_test_split
```

```
X = df[feature_columns]
y = df.Outcome

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


from sklearn.metrics import confusion_matrix, accuracy_score, classification_report


def evaluate(model, X_train, X_test, y_train, y_test):
    y_test_pred = model.predict(X_test)
    y_train_pred = model.predict(X_train)

    print("TRAINIG RESULTS: \n==============================")
    clf_report = pd.DataFrame(classification_report(y_train, y_train_pred, output_dict=True))
    print(f"CONFUSION MATRIX:\n{confusion_matrix(y_train, y_train_pred)}")
    print(f"ACCURACY SCORE:\n{accuracy_score(y_train, y_train_pred):.4f}")
    print(f"CLASSIFICATION REPORT:\n{clf_report}")

    print("TESTING RESULTS: \n==============================")
    clf_report = pd.DataFrame(classification_report(y_test, y_test_pred, output_dict=True))
    print(f"CONFUSION MATRIX:\n{confusion_matrix(y_test, y_test_pred)}")
    print(f"ACCURACY SCORE:\n{accuracy_score(y_test, y_test_pred):.4f}")
    print(f"CLASSIFICATION REPORT:\n{clf_report}")
```

ii)Random Forest

```
# Data Processing
import pandas as pd
import numpy as np

# Modelling
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score, recall_score,
ConfusionMatrixDisplay
from sklearn.model_selection import RandomizedSearchCV, train_test_split
from scipy.stats import randint

# Tree Visualisation
from sklearn.tree import export_graphviz
from IPython.display import Image
import graphviz
bank_data['default'] = bank_data['default'].map({'no':0,'yes':1,'unknown':0})
bank_data['y'] = bank_data['y'].map({'no':0,'yes':1})
# Split the data into features (X) and target (y)
X = bank_data.drop('y', axis=1)
y = bank_data['y']

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
rf = RandomForestClassifier()
```

```
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
# Export the first three decision trees from the forest

for i in range(3):
    tree = rf.estimators_[i]
    dot_data = export_graphviz(tree,
                               feature_names=X_train.columns,
                               filled=True,
                               max_depth=2,
                               impurity=False,
                               proportion=True)
    graph = graphviz.Source(dot_data)
    display(graph)
```

output: