

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.



LAB REPORT
on

Machine Learning (23CS6PCMAL)

Submitted by

Anjali(1BM22CS043)

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING



B.M.S. COLLEGE OF ENGINEERING

(Autonomous Institution under VTU)

BENGALURU-560019

Sep-2024 to Jan-2025

B.M.S. College of Engineering,
Bull Temple Road, Bangalore 560019
(Affiliated To Visvesvaraya Technological University, Belgaum)

Department of Computer Science and Engineering



CERTIFICATE

This is to certify that the Lab work entitled “Machine Learning (23CS6PCMAL)” carried out by Anjali(1BM22CS043) , who is Bonafide student of **B.M.S. College of Engineering**. It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum. The Lab report has been approved as it satisfies the academic requirements in respect of a Machine Learning (23CS6PCMAL) work prescribed for the said degree.

Lab Faculty Incharge Name: Ms. Saritha A N Assistant Professor Department of CSE, BMSCE	Dr. Kavitha Sooda Professor & HOD Department of CSE, BMSCE
--	---

Index

Sl. No.	Date	Experiment Title	Page No.
1	21-2-2025	Write a python program to import and export data using Pandas library functions	4
2	3-3-2025	Demonstrate various data pre-processing techniques for a given dataset	6
3	10-3-2025	Implement Linear and Multi-Linear Regression algorithm using appropriate dataset	10
4	17-3-2025	Build Logistic Regression Model for a given dataset	13
5	24-3-2025	Use an appropriate data set for building the decision tree (ID3) and apply this knowledge to classify a new sample	15
6	7-4-2025	Build KNN Classification model for a given dataset	20
7	21-4-2025	Build Support vector machine model for a given dataset	24
8	5-5-2025	Implement Random forest ensemble method on a given dataset	28
9	5-5-2025	Implement Boosting ensemble method on a given dataset	30
10	12-5-2025	Build k-Means algorithm to cluster a set of data stored in a .CSV file	33
11	12-5-2025	Implement Dimensionality reduction using Principal Component Analysis (PCA) method	34

Program 1

Write a python program to import and export data using Panda's library functions

Screenshot

The image shows a handwritten Python program on lined paper. The code imports pandas as 'pd', reads a CSV file 'housing.csv' into a DataFrame 'df1', and sets specific columns to 'object' dtype. It then prints the columns, checks for null values, and prints the index. A second DataFrame 'df2' is also read from a CSV file 'diabetes.csv' and its columns are printed. The output shows the first row of 'df1' with values for longitude, latitude, housing median age, total rooms, and total bedrooms.

```
LAB 1

import pandas as pd

df1 = pd.read_csv('housing.csv')

df1.columns = ['longitude', 'latitude', 'housing - median age',
               'total - rooms', 'total - bedrooms'], dtype=
               'object')

df1.describe()

count      longitude      latitude      housing - median      total rooms      total - bedrooms
20640      20640      20640      20640      20640      20433

population      households      median      income      median-hou
20640      20640      20640      20640      20640

cnt = len(df1.ocean-proximity.unique())
print(cnt)
5

print(df1.columns[df1.null().any()])
Index(['total - bedrooms'], dtype='object')

df2 = pd.read_csv('diabetes.csv')

print(df2.columns[df2.isnull().any()])
Index([], dtype='object')
```

Code:

```
import pandas as pd

try:

    df = pd.read_csv('input.csv')

    print("Data imported successfully!\n")

    print(df)

except FileNotFoundError:

    print("The file 'input.csv' was not found.")

df["Processed"] = True

df.to_csv('output.csv', index=False)

print("\nData exported successfully to 'output.csv'.")
```

Program 2

Demonstrate various data pre-processing techniques for a given dataset

Screenshots:

```
LAB-0
Date: __/__/__
Page: __

tickers = ['HDFC BANK.NS', 'ICICBANK.NS', 'KOTAK.NS']
data = yf.download(tickers, start="2024-01-01", end="2024-12-30", group_by='tickers')
import sys, yf, pandas as pd, matplotlib.pyplot as plt
hdfc = data['Daily Return'] = hdfc - with ['close']
plt.figure(figsize=(12,8))
plt.subplot(2,1,1)
hdfc = data['close'].plot(title="HDFC Industries")
plt.subplot(2,1,2)
hdfc = data['Daily Return'].plot(title="HDFC Industries")
plt.tight_layout()
plt.show()

Daily Return for HDFC Industries
Date:
2024-01-01
2024-01-02
2024-01-03
2024-01-04
2024-01-05
```

Code

```
import pandas as pd
```

```
import numpy as np
```

```
from sklearn.preprocessing import LabelEncoder, StandardScaler, MinMaxScaler
```

```
data = {
```

```
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve', None],
```

```
    'Age': [25, 30, np.nan, 35, 29, 40],
```

```

    'Department': ['HR', 'IT', 'Finance', 'IT', 'HR', 'Finance'],
    'Salary': [50000, 60000, 58000, 62000, np.nan, 52000]
}

df = pd.DataFrame(data)

print("Original DataFrame:\n", df)

df['Age'].fillna(df['Age'].mean(), inplace=True)
df['Salary'].fillna(df['Salary'].median(), inplace=True)
df['Name'].fillna('Unknown', inplace=True)

le = LabelEncoder()
df['Department_Encoded'] = le.fit_transform(df['Department'])

df.drop_duplicates(inplace=True)

df.rename(columns={'Salary': 'Monthly_Salary'}, inplace=True)

df['Age'] = df['Age'].astype(int)

scaler = MinMaxScaler()
df['Salary_Normalized'] = scaler.fit_transform(df[['Monthly_Salary']])

standard_scaler = StandardScaler()

```



```
df['Age_Standardized'] = standard_scaler.fit_transform(df[['Age']])
```

```
print("\nPreprocessed DataFrame:\n", df)
```

Program 3

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

Screenshots

```
Lab - 2
Date: __/__/__
Page: __

Linear Regression
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression

dt = pd.read_csv('1000-companies.csv')

dt_encoded = pd.get_dummies(dt, columns=['State'],
                             drop_first=True)

X = dt_encoded.drop('Profit', axis='columns')
y = dt_encoded['Profit']

def predict_profit (rnd_spend, admin_spend,
                    marketng_spend, State):
    input_data = [col:0 for col in X.columns]

    input_data[0] = rnd_spend

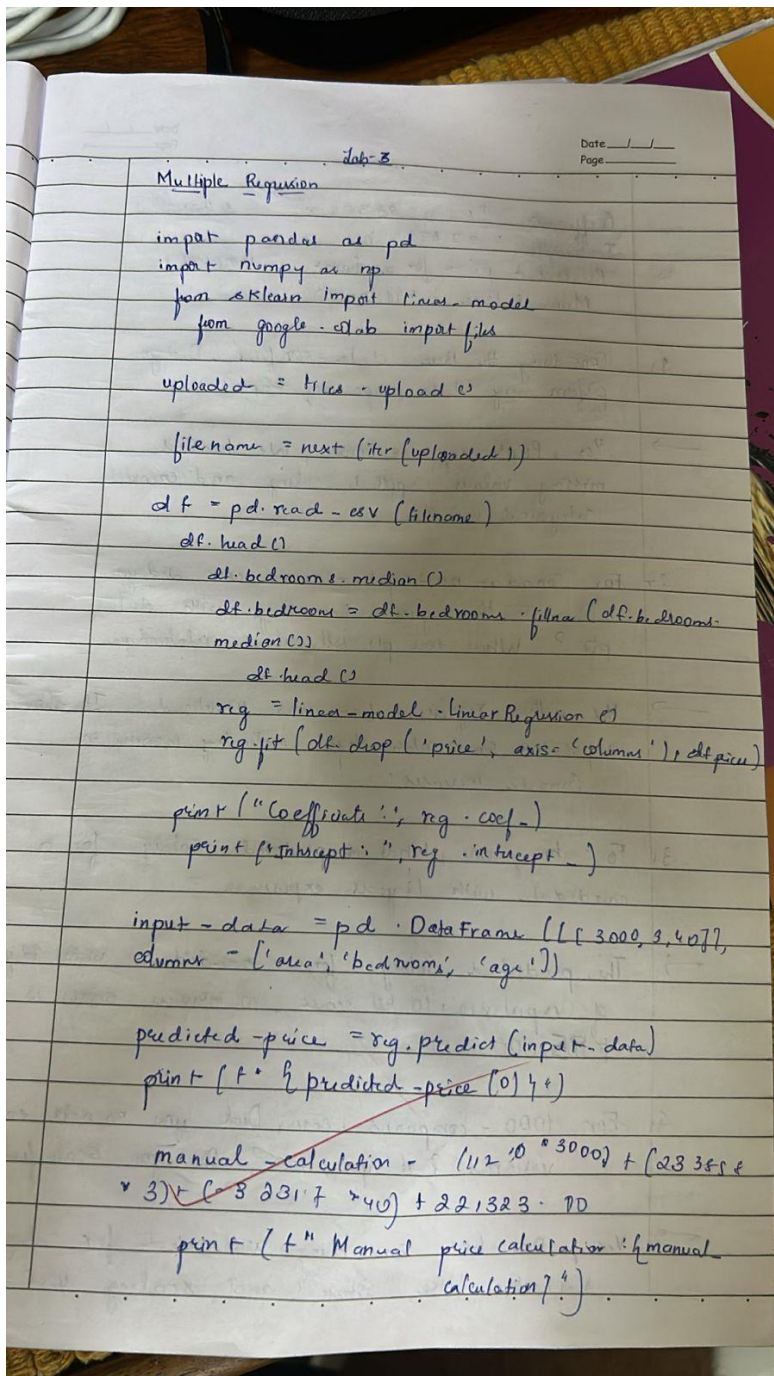
    state_col = f'State {State}'
    if state_col in input_data:
        input_data[state_col] = 1

    input_dt = pd.DataFrame([input_data])

    return f'Predicted Profit: $ {predicted_profit:2f}'

print(predict_profit(91640.48, 11931.24,
                    'Florida'))

Output:
```



Code

Linear Regression

```

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_boston
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

```

```
from sklearn.metrics import mean_squared_error, r2_score

# Load dataset
boston = load_boston()
df = pd.DataFrame(boston.data, columns=boston.feature_names)
df['PRICE'] = boston.target

# Use only one feature for simple linear regression (e.g., RM = average number of rooms)
X = df[['RM']]
y = df['PRICE']

# Split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)

# Train model
lr = LinearRegression()
lr.fit(X_train, y_train)
```

```

# Predict
y_pred = lr.predict(X_test)

# Output
print("Linear Regression Results")
print("Coefficients:", lr.coef_)
print("Intercept:", lr.intercept_)
print("MSE:", mean_squared_error(y_test, y_pred))
print("R2 Score:", r2_score(y_test, y_pred))

# Plot
plt.scatter(X_test, y_test, color='blue')
plt.plot(X_test, y_pred, color='red')
plt.xlabel('Average Number of Rooms (RM)')
plt.ylabel('House Price')
plt.title('Simple Linear Regression')
plt.show()

# Multiple Linear Regression

# Use all features
X = df.drop('PRICE', axis=1)
y = df['PRICE']

# Split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)

# Train model
mlr = LinearRegression()
mlr.fit(X_train, y_train)

# Predict
y_pred = mlr.predict(X_test)

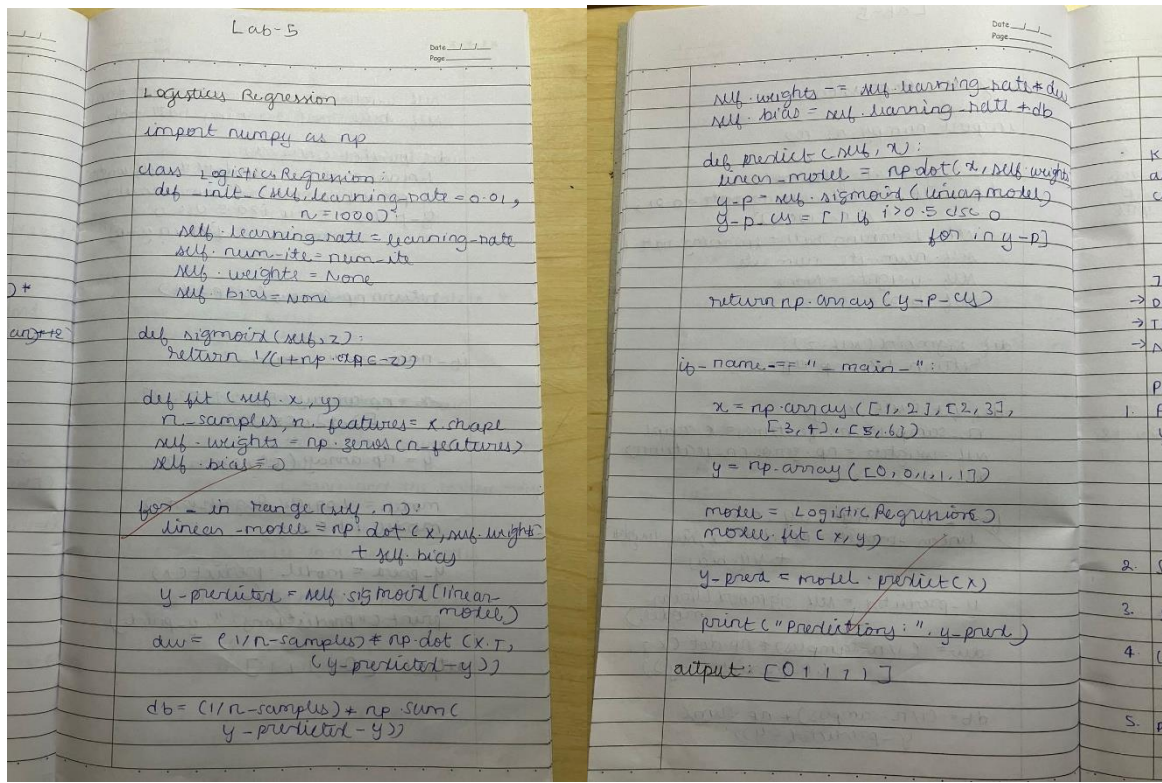
# Output
print("\nMultiple Linear Regression Results")
print("Coefficients:", mlr.coef_)
print("Intercept:", mlr.intercept_)
print("MSE:", mean_squared_error(y_test, y_pred))
print("R2 Score:", r2_score(y_test, y_pred))

```

Program 4

Build Logistic Regression Model for a given dataset

Screenshot's



Code

```

import pandas as pd

from sklearn.datasets import load_iris

from sklearn.linear_model import LogisticRegression

from sklearn.model_selection import train_test_split

from sklearn.metrics import confusion_matrix, accuracy_score, classification_report

iris = load_iris()

df = pd.DataFrame(iris.data, columns=iris.feature_names)

df['species'] = iris.target
    
```

```

df_binary = df[df['species'] != 2] # Remove class 2 (Virginica)

X = df_binary.iloc[:, :-1] # Features
y = df_binary['species']    # Target (0 or 1)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = LogisticRegression()
model.fit(X_train, y_train)

# Step 5: Predict and evaluate
y_pred = model.predict(X_test)

print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nAccuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))

```

Program 5

Use an appropriate data set for building the decision tree (ID3) and apply this knowledge to classify a new sample

Screenshots

Date: ___/___/___
 Page: ___

Lab 4

Decision Tree

Instance	a ₂	a ₃	Classification
1	Hot	High	No
2	Hot	High	No
3	Cool	High	No
4	Hot	High	No
5	Hot	Normal	Yes
6			
7			
8			

→ Entropy(S) = $-\frac{4}{5} \log(\frac{4}{5}) - \frac{1}{5} \log(\frac{1}{5})$

For a₂
 S_{Hot} [1+3] = $-\frac{1}{4} \log(\frac{1}{4}) - \frac{3}{4} \log(\frac{3}{4})$
 = 0.9189

Gain = 0.011286

For a₃
 S_{High} = [0+0] = 0
 S_{Normal} = [1+10] = 0
 Gain (S a₃) = 0.7419

Decision Tree Diagram:
 Root Node: a₃
 - High branch leads to Node a₂
 - Normal branch leads to Node a₂
 Node a₂ (under High):
 - Hot branch leads to Leaf 1,2,3
 - Cool branch leads to Leaf 4
 Node a₂ (under Normal):
 - Hot branch leads to Leaf 5
 - Cool branch leads to Leaf 6,7,8

Code for Decision Tree

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
sklearn metric = accuracy_score
```

```
dt = pd.read_csv('data.csv')
x = dt.drop('Spam', axis=1)
y = dt['Spam']
```

```
x_train, x_test, y_train, y_test
```

```
dt_classifier = DecisionTreeClassifier()
dt_classifier.fit(x_train, y_train)
```

```
y_pred = dt_classifier.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
```

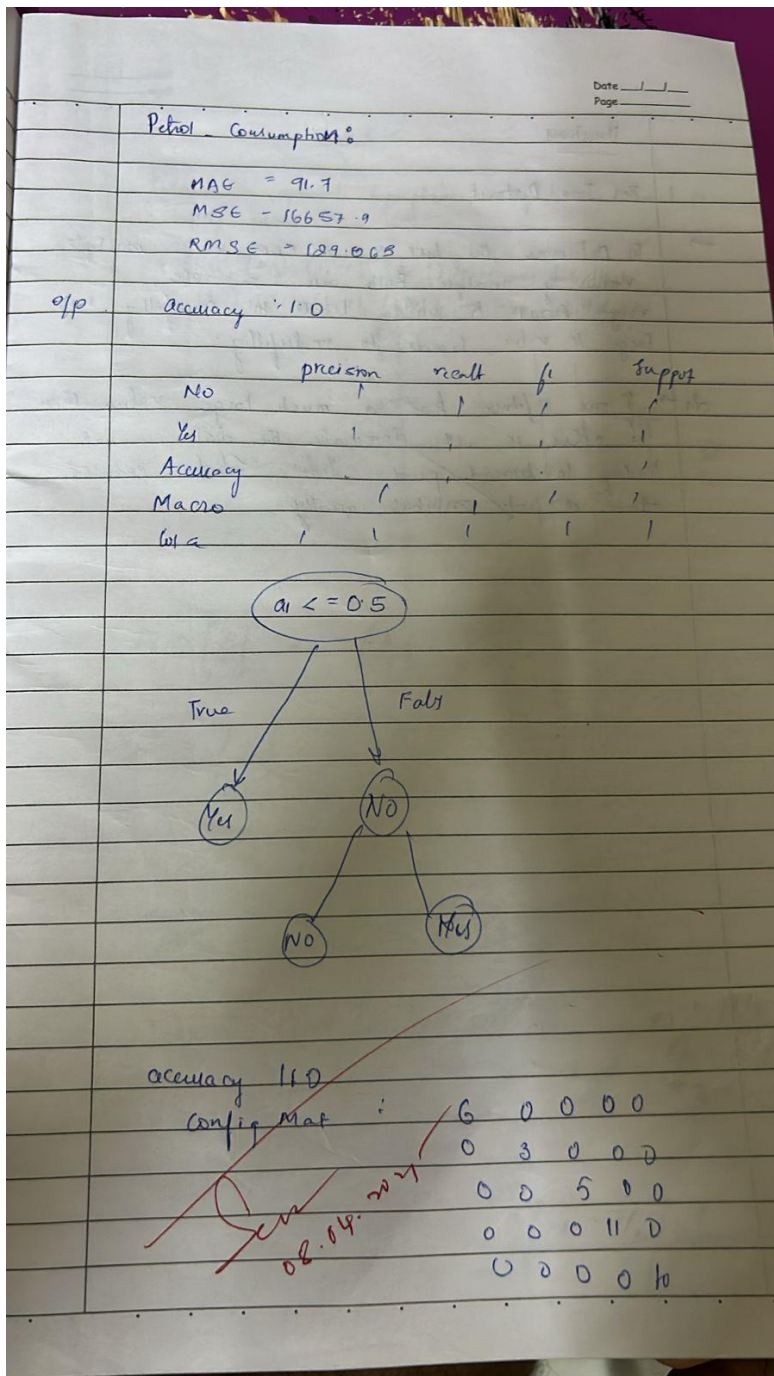
```
print("accuracy score", accuracy)
print("conf matrix", conf_matrix)
```

True

acc score : 1.0

conf mat :

10	0	0
0	0	0
0	0	1



Code

```
import pandas as pd
```

```

from sklearn.preprocessing import LabelEncoder

from sklearn.tree import DecisionTreeClassifier

from sklearn import tree

data= {

    'Outlook': ['Sunny', 'Sunny', 'Overcast', 'Rain', 'Rain', 'Rain', 'Overcast',
               'Sunny', 'Sunny', 'Rain', 'Sunny', 'Overcast', 'Overcast', 'Rain'],
    'Temperature': ['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool', 'Cool',
                   'Mild', 'Cool', 'Mild', 'Mild', 'Mild', 'Hot', 'Mild'],
    'Humidity': ['High', 'High', 'High', 'High', 'Normal', 'Normal', 'Normal',
                'High', 'Normal', 'Normal', 'Normal', 'High', 'Normal', 'High'],
    'Wind': ['Weak', 'Strong', 'Weak', 'Weak', 'Weak', 'Strong', 'Strong',
            'Weak', 'Weak', 'Weak', 'Strong', 'Strong', 'Weak', 'Strong'],
    'PlayTennis': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes',
                  'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No']
}

```

```

df=pd.DataFrame(data)

le = LabelEncoder()

for column in df.columns:

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

# Step 3: Separate features and label

X = df.drop('PlayTennis', axis=1)

y = df['PlayTennis']

clf= DecisionTreeClassifier(criterion='entropy') # ID3 uses 'entropy'

```

```

clf = clf.fit(X, y)

print("\nDecision Tree Rules:")

tree_text = tree.export_text(clf, feature_names=X.columns.tolist())

print(tree_text)

# Example: Outlook=Rain, Temperature=Mild, Humidity=High, Wind=Weak

# Encode input sample with same label encoding order used earlier

sample = pd.DataFrame({

    'Outlook': [le.transform(['Rain'])[0]],

    'Temperature': [le.transform(['Mild'])[0]],

    'Humidity': [le.transform(['High'])[0]],

    'Wind': [le.transform(['Weak'])[0]]

})

# Predict

prediction = clf.predict(sample)

result = 'Yes' if prediction[0] == 1 else 'No'

print(f"\nPrediction for new sample (Rain, Mild, High, Weak): {result}")

```

Program 6 Build KNN Classification model for a given dataset

Lab 5
KNN :-

Person	Age	Salaryk	Leapt	distance
A	18	50	N	59.8
B	23	55	N	46.57
C	24	70	N	31.95
D	24	60	Y	40.44
E	43	70	Y	36.04
F	38	60	Y	60.07
X	35	100	Y	?

(X = 35, 100)

K = 3

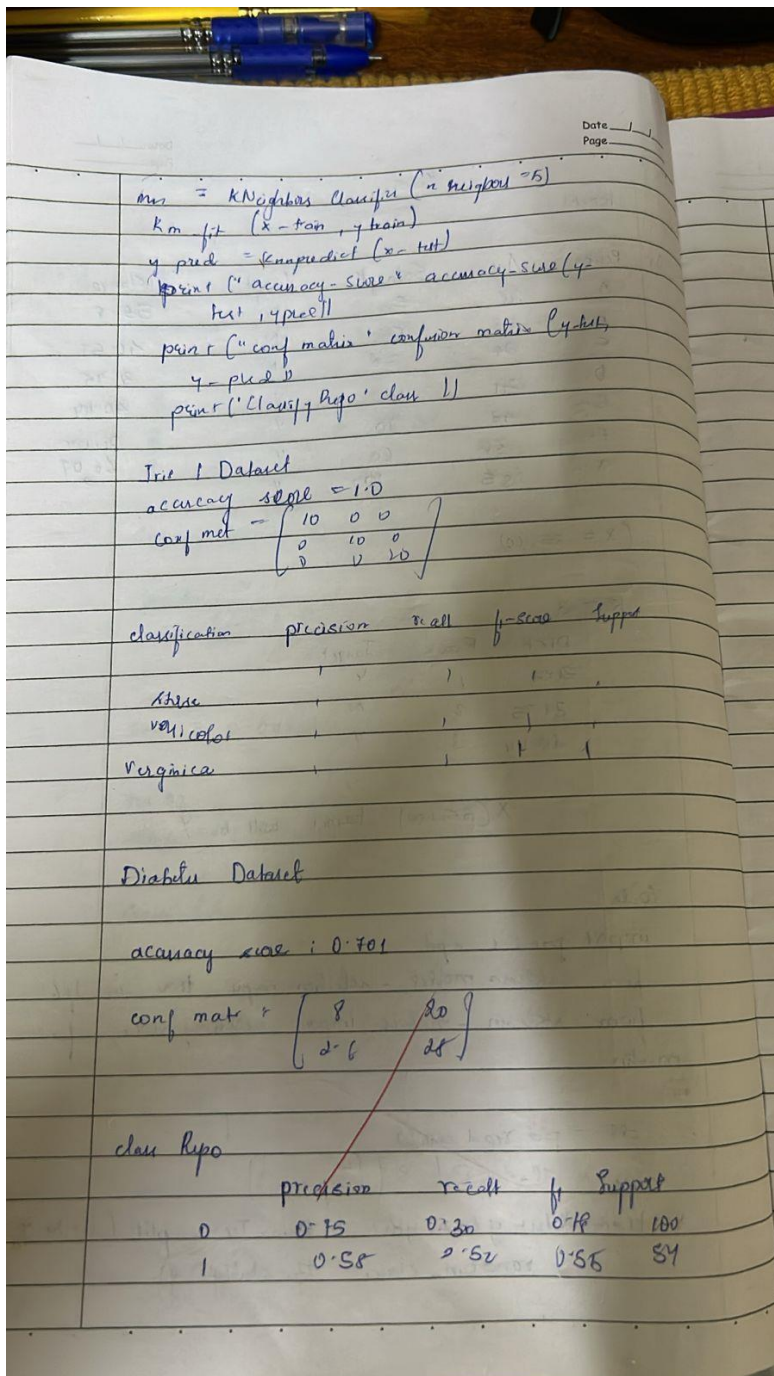
Dist	Rank	Target
31.95	1	Y
36.04	2	N
40.44	3	Y

X(35, 100) Target will be Y

Code!

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
```

```
df = pd.read_csv(' ')
x, y = df[['Age', 'Salary', 'Leapt']]
X_train, X_test, y_train, y_test = train_test_split(x, y,
                                                    random_state=42, stratify=y)
```

Code

```
import pandas as pd
```

```
from sklearn.datasets import load_iris
```

```
from sklearn.model_selection import train_test_split
```

```
from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy_score, classification_report, confusion_matrix


# Step 1: Load the Iris dataset

iris = load_iris()

X = pd.DataFrame(iris.data, columns=iris.feature_names)
```

```

y = pd.Series(iris.target)

# Step 2: Split the data (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 3: Build KNN model (k = 3)
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)

# Step 4: Predict and evaluate
y_pred = knn.predict(X_test)

# Results
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
print("Accuracy Score:", accuracy_score(y_test, y_pred))

```

Program 7

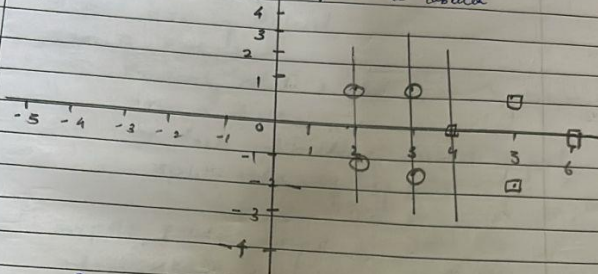
Build Support vector machine model for a given dataset

Screenshots

Lab 7:-

Draw an optimal hyperplane using linear sum to classify points.

$\{(1,1), (2,1), (1,-1), (2,-1)\} \rightarrow +ve \text{ labelled}$
 $\{(4,0), (5,1), (5,-1), (6,0)\} \rightarrow -ve \text{ labelled}$



$$s_1 = \begin{bmatrix} 2 \\ 1 \end{bmatrix} \quad s_2 = \begin{bmatrix} 2 \\ -1 \end{bmatrix} \quad s_3 = \begin{bmatrix} 4 \\ 0 \end{bmatrix}$$

$$\bar{s}_1 = \begin{bmatrix} 2 \\ 1 \end{bmatrix} \quad \bar{s}_2 = \begin{bmatrix} 2 \\ -1 \end{bmatrix} \quad \bar{s}_3 = \begin{bmatrix} 4 \\ 0 \end{bmatrix}$$

$$\alpha_1 \bar{s}_1 + \alpha_2 \bar{s}_2 + \alpha_3 \bar{s}_3 = +1$$

$$\alpha_1 \bar{s}_1 + \alpha_2 \bar{s}_2 + \alpha_3 \bar{s}_3 = +1$$

$$\alpha_1 \bar{s}_1 + \alpha_2 \bar{s}_2 + \alpha_3 \bar{s}_3 = -1$$

$$\alpha_1 (6) + \alpha_2 (4) + \alpha_3 (9) = +1 \quad \alpha_1 = 13/4$$

$$\alpha_1 (4) + \alpha_2 (6) + \alpha_3 (9) = +1 \quad \alpha_2 = 13/4$$

$$\alpha_1 (9) + \alpha_2 (9) + \alpha_3 (13) = -1 \quad \alpha_3 = -7/2$$

$$w = \alpha_1 \bar{s}_1 + \alpha_2 \bar{s}_2 + \alpha_3 \bar{s}_3$$

$$= 13/4 \begin{bmatrix} 2 \\ 1 \end{bmatrix} + 13/4 \begin{bmatrix} 2 \\ -1 \end{bmatrix} + (-7/2) \begin{bmatrix} 4 \\ 0 \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \end{bmatrix}$$

Code

```
import pandas as pd

from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split

from sklearn.svm import SVC

from sklearn.metrics import confusion_matrix, classification_report, accuracy_score


# Step 1: Load the Iris dataset

iris = load_iris()

X = pd.DataFrame(iris.data, columns=iris.feature_names)

y = pd.Series(iris.target)


# Step 2: Split the data into train and test sets (80% train, 20% test)
```

```
X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=0.2, random_state=42)

# Step 3: Build and train the SVM model(linear kernel)

svm_model = SVC(kernel='linear')

svm_model.fit(X_train, y_train)

# Step 4: Predict and evaluate

y_pred=svm_model.predict(X_test)

# Output results

print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))

print("\nClassification Report:\n", classification_report(y_test, y_pred))

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

Program 8

Implement Random Forest ensemble method on a given dataset

Screenshots

Lab 8 :-

Parameters of Random Forest Classifier :-

- n - estimate no. of trees
- criterion - function to measure quality of split
- max-depth - min depth of tree
- min sample split - min sample size to split node

I/p : Training dataset

for n trees:

Randomly select samples with replacement
grow a decision tree
at each split choose subset of features
split nodes using gini

import pandas as pd
from sklearn.ensemble import RandomForestClassifier

iris = pd.read_csv('iris.csv')

x = iris[['sepal.length', 'sepal.width', 'petal.length', 'petal.width', 'species']]

y = iris['species']

x_train, y_train, x_test, y_test = train_test_split(x, y)

y_pred = rf.predict(x_test)

accuracy = accuracy_score(y_test, y_pred)

print(accuracy)

for n in iterations:

rf = RandomForestClassifier(n_estimators=n, random_state=42)
rf.fit(x_train, y_train)

y_pred = rf.predict(x_test)

score = accuracy_score(y_test, y_pred)

print(score)

Code

```
import pandas as pd

from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score


# Step 1: Load Iris dataset

iris = load_iris()

X = pd.DataFrame(iris.data, columns=iris.feature_names)

y = pd.Series(iris.target)
```

Step 2: Split into train and test sets (80% train, 20% test)

```
X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=0.2, random_state=42)
```

Step 3: Train Random Forest model

```
rf=RandomForestClassifier(n_estimators=100, random_state=42) # 100 trees
```

```
rf.fit(X_train, y_train)
```

Step 4: Predict and evaluate

```
y_pred = rf.predict(X_test)
```

```
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

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

Program 9

Implement Boosting ensemble method on a given dataset

Screenshots

Lab 9 :-

Boosting :-

Combine Multiple weak learners to create a strong learner. It works by training models sequentially where each model focuses on what was misclassified by the previous one.

Pseudo Algorithm :-

1. Start with equal weight for all training samples.
2. Train a weak model.
3. Calculate error of updated samples.
4. Add weak model to ensemble with weight based on its accuracy.
5. Repeat of n estimators.
6. Final prediction.

Code

```
import pandas as pd

from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split

from sklearn.ensemble import AdaBoostClassifier

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

from sklearn.tree import DecisionTreeClassifier


# Step 1: Load the Iris dataset

iris = load_iris()
```



```

X=pd.DataFrame(iris.data, columns=iris.feature_names)

y = pd.Series(iris.target)


# Step 2: Split into train and test sets (80% train, 20% test)

X_train, X_test, y_train, y_test =train_test_split(X, y, test_size=0.2, random_state=42)


# Step 3: Build AdaBoost modelwith DecisionTreeClassifier as base estimator

base_estimator = DecisionTreeClassifier(max_depth=1)

model= AdaBoostClassifier(base_estimator=base_estimator, n_estimators=50, learning_rate=1.0,
random_state=42)

model.fit(X_train, y_train)


# Step 5: Predict and evaluate

y_pred= model.predict(X_test)

print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))

print("\nClassification Report:\n", classification_report(y_test, y_pred))

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

```

Program 10

Build k-Means algorithm to cluster a set of data stored in a .CSV file

Screenshots

Lab-10 KMeans

1. Write K-means algo

1. choose no of clusters K
2. initialize K centroids
3. Assign each point to nearest centroid.
4. Recompute centroids of the mean of assigned points.
5. Repeat step 3-4 until centroids stop changing.

2. How to determine no of clusters

- * Use elbow method
- * true diff value of K & calculate SSE
- * Plot SSE vs K

3. Parameters of K-Means

- n - cluster : no of clusters init initialized
 - ~~method~~ n -init = no of times algo will run
 - max iter : max iteration per run for n .
 - ~~tolerance~~ tolerance to declare convergence.
- ~~Ques~~ $n=10, n=100$

Code

```
import pandas as pd

from sklearn.cluster import KMeans

import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler


# Step 1: Load dataset from CSV

df = pd.read_csv('your_dataset.csv') # Replace with your file path


# Optional: View first few rows

print("Data Preview:\n", df.head())
```

```

# Step 2: Select relevant numeric columns for clustering

# You can specify specific columns like: df[['column1', 'column2']]

X = df.select_dtypes(include='number')


# Step 3: Scale the data (important for K-Means)

scaler = StandardScaler()

X_scaled = scaler.fit_transform(X)


# Step 4: Apply K-Means clustering (e.g., 3 clusters)

kmeans = KMeans(n_clusters=3, random_state=42)

df['Cluster'] = kmeans.fit_predict(X_scaled)


# Step 5: Print cluster centers

print("Cluster Centers:\n", kmeans.cluster_centers_)


# Optional Step 6: Visualize (works well for 2D or PCA-reduced data)

if X.shape[1] >= 2:

    plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=df['Cluster'], cmap='viridis')

    plt.title("K-Means Clustering")

    plt.xlabel("Feature 1")

    plt.ylabel("Feature 2")

    plt.show(

```

Program 11

Implement Dimensionality reduction using Principal Component Analysis (PCA) method

Screenshots

Lab 10 : PCA

1. Calculate Mean
2. Calculate covariance matrix
3. Eigen val of covariance matrix
4. Computation of eigen vectors with eigen vectors
5. Computation of first principle component
6. Geometric meaning of 1st principle component

x_1	x_2	x_3	x_4	x_5
4	8	13	13	7
11	4	5	5	14

calculate mean

$$\bar{x}_1 = 1/4 (4 + 11 + 13 + 13 + 11)$$

$$\bar{x}_2 = 1/4 (8 + 4 + 5 + 5 + 14) = 8.5$$

$$\begin{aligned} \text{Cov}(x_1, x_2) &= 1/N \sum_{i=1}^N (x_{1i} - \bar{x}_1)(x_{2i} - \bar{x}_2) \\ &= 1/5 [(4-8)(8-8.5) + (11-8)(4-8.5) + (13-8)(5-8.5) + (13-8)(5-8.5) + (11-8)(14-8.5)] \\ &= -11 \end{aligned}$$

$$S = \begin{bmatrix} \text{Cov}(x_1, x_1) & \text{Cov}(x_1, x_2) \\ \text{Cov}(x_2, x_1) & \text{Cov}(x_2, x_2) \end{bmatrix}$$

$$S = \begin{bmatrix} 14 & -11 \\ -11 & 23 \end{bmatrix}$$

ii)

$$\text{Eigen value} = \begin{bmatrix} 14 - \lambda & -11 \\ -11 & 23 - \lambda \end{bmatrix}$$

Date: / /
Page:

$$= \lambda^2 - 37\lambda + 20$$

$$\lambda = \frac{37 \pm \sqrt{565}}{2}$$

$$\lambda_1 = \frac{37 + \sqrt{565}}{2}, \lambda_2 = \frac{37 - \sqrt{565}}{2}$$

$$\lambda_1 = 30.3842, \lambda_2 = 6.6151$$

$$\lambda_1 > \lambda_2$$

iv) Eigen vector for Target eigen value

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix} = (S - \lambda I) X$$

$$= \begin{bmatrix} 14 - \lambda_1 & -11 \\ 11 & 23 - \lambda_1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} (14 - \lambda_1) v_1 - 11 v_2 \\ -11 v_1 + (23 - \lambda_1) v_2 \end{bmatrix}$$

$$\frac{v_1}{11} = \frac{v_2}{14 - \lambda_1} = c_1$$

$$\|v\| = \sqrt{11^2 + (14 - \lambda_1)^2} = 10.7348$$

unit eigen vector to λ

$$p_1 = \begin{bmatrix} 0.5574 \\ 0.8303 \end{bmatrix}$$

$$e_4 = \begin{bmatrix} 0.8303 \\ 0.5574 \end{bmatrix}$$

$$e_1 = \begin{bmatrix} 0.5574 & -0.8303 \\ 0.8303 & 0.5574 \end{bmatrix}$$

(x₁₁ x₂₁)
(x₁₂ x₂₂)

Code

```
import pandas as pd

from sklearn.datasets import load_iris

from sklearn.preprocessing import StandardScaler
```

```
from sklearn.decomposition import PCA
```

```
import matplotlib.pyplot as plt
```

```
# Step 1: Load the Iris dataset
```

```
iris = load_iris()
```

```
X = iris.data
```



```

y = iris.target

feature_names = iris.feature_names


# Step 2: Standardize the data

scaler = StandardScaler()

X_scaled = scaler.fit_transform(X)


# Step 3: Apply PCA (reduce to 2 components for visualization)

pca = PCA(n_components=2)

X_pca = pca.fit_transform(X_scaled)


# Step 4: Plot the 2D PCA result

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

scatter = plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, cmap='viridis', edgecolor='k', s=60)

plt.xlabel("Principal Component 1")

plt.ylabel("Principal Component 2")

plt.title("PCA - Iris Dataset")

plt.legend(handles=scatter.legend_elements()[0], labels=iris.target_names)

plt.grid(True)

plt.show()


# Explained variance

print("Explained variance ratio:", pca.explained_variance_ratio_)

```