Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Continuous Assessment for Laboratory / Assignment sessions

Academic Year 2022-23

Name: Ayush Jain SAP ID: 60004200132

Course: Machine Learning Laboratory Course Code: **DJ19CEEL6021**

Year: **T.Y. B.Tech.** Sem: **VII** Batch: B 3

Department: Computer Engineering

Performance Indicators (Any no. of Indicators) (Maximum 5 marks per indicator)	1	2	3	4	5	6	7	8	9	10	11	Σ	Avg	A1	A2	Σ	Avg
Course Outcome	2, 4	2, 4	2, 4	2, 4	2, 4	3	2, 4	2, 4	5								
1. Knowledge (Factual/Conceptual/Procedural/ Metacognitive)	4	5	5	5	5	5	5	5					5		5		
2. Describe (Factual/Conceptual/Procedural/ Metacognitive)	4	4	4	5	5	4	4	4					4		5		
3. Demonstration (Factual/Conceptual/Procedural/ Metacognitive)	5	5	5	5	5	5	5	4					5		5		
4. Strategy (Analyse & / or Evaluate) (Factual/Conceptual/ Procedural/Metacognitive)	4	4	4	4	4	5	5	5					4		4		
5. Interpret/ Develop (Factual/Conceptual/ Procedural/Metacognitive)	–	–	–	–	–	–	–	–	–				4		4		
6. Attitude towards learning (receiving, attending, responding, valuing, organizing, characterization by value)	4	4	4	4	5	4	4	5						–	–		
7. Non-verbal communication skills/ Behaviour or Behavioural skills (motor skills, hand-eye coordination, gross body movements, finely coordinated body movements speech behaviours)	–	–	–	–	–	–	–	–	–					–	–		
Total	21	22	22	23	24	23	23	23	22					22	23		
Signature of the faculty member																	

Outstanding (5), Excellent (4), Good (3), Fair (2), Needs Improvement (1)

Laboratory marks Σ Avg. = 22	Assignment marks Σ Avg. = 22	Total Term-work (25) = 22
Laboratory Scaled to (15) = 13	Assignment Scaled to (10) = 9	Sign of the Student:

Signature of the Faculty member: Rutuja
Name of the Faculty member: 03/05/23

Signature of Head of the Department
Date:

Scanned with CamScanner

Machine Learning

Experiment 1

Ayush Jain 60004200132 B3

Aim : To implement univariate linear regression in python with and without using any inbuilt function.

Theory:
Linear regression is a statistical method that is used to analyze the relationship between two continuous variables. In simple linear regression, the goal is to find a linear relationship between the predictor variable (X) and the response variable (Y), which can be expressed by the equation:

$Y = a + bX$

where Y is the predicted value of the response variable, X is the value of the predictor variable, a is the intercept or the value of Y when X is 0, and b is the slope or the change in Y for every unit increase in X.

The process of linear regression involves finding the best values for the intercept and slope that minimize the sum of the squared differences between the actual values of Y and the predicted values of Y based on the values of X. This is done using a technique called least squares regression.

Linear regression can be used to make predictions or to model the relationship between variables in order to gain insights and make decisions. It is commonly used in fields such as finance, economics, and social sciences.

Let's take an example of a simple linear regression, where we want to predict the salary of an employee based on their years of experience. We have a dataset of 10 employees with their corresponding years of experience and salaries:

Years of Experience	**Salary (in thousands)**
1	**20**
2	**22**

Years of Experience	Salary (in thousands)
3	25
4	28
5	30
6	32
7	35
8	38
9	40
10	42

We can plot this data on a scatter plot to see if there is a linear relationship between years of experience and salary:

scatter plot of years of experience and salary

From the scatter plot, we can see that there appears to be a positive linear relationship between years of experience and salary, meaning that as years of experience increase, salary tends to increase as well.

We can then perform linear regression on this data to find the line of best fit that represents the relationship between years of experience and salary. The equation for the line of best fit is:

Salary = 18.98 + 4.16*Years of Experience

The line of best fit represents the predicted salary for a given number of years of experience. For example, if an employee has 7 years of experience, we can predict their salary to be approximately $35,000.

Linear regression can be a powerful tool for analyzing and predicting relationships between variables. However, it's important to keep in mind that correlation does not always imply causation, and there may be other factors that influence the relationship between variables.

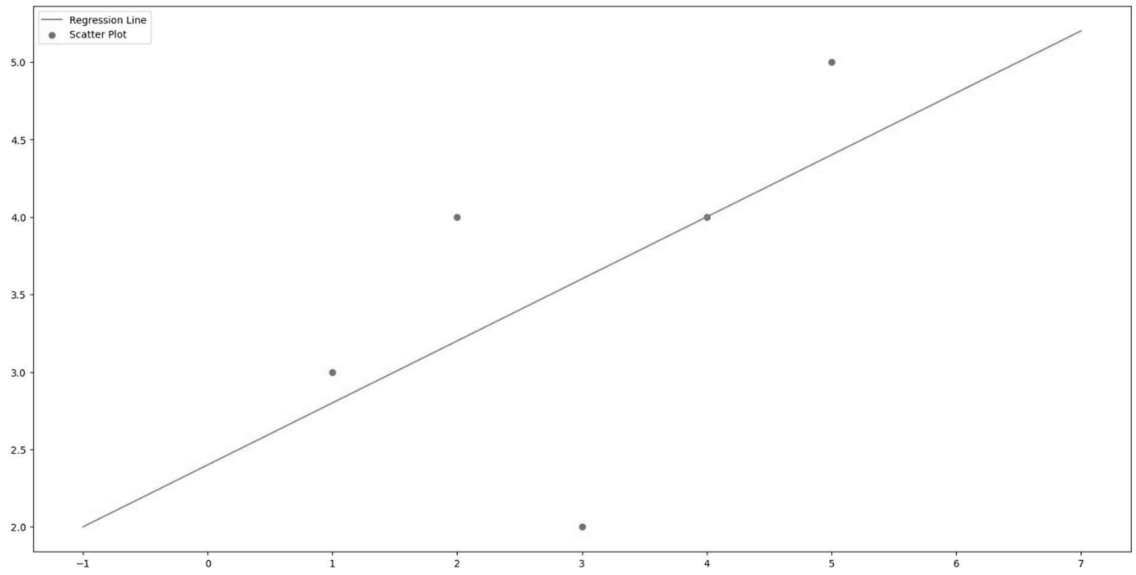
Code without inbuilt function:

```
x = [1,2,3,4,5]
y = [3,4,2,4,5]

import numpy as np
```

```
import pandas as pd
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = (20.0, 10.0)

X = np.array([1,2,3,4,5])#data['Head Size(cm^3)'].values
Y = np.array([3,4,2,4,5])#data['Brain Weight(grams)'].values

mean_x = np.mean(X)
mean_y = np.mean(Y)

n = len(X)

numer = 0
denom = 0
for i in range(n):
    numer += (X[i] - mean_x) * (Y[i] - mean_y)
    denom += (X[i] - mean_x) ** 2
    w0= numer / denom
    w1 = mean_y - (w0 * mean_x)

print(w0, w1)

max_x = np.max(X) + 2
min_x = np.min(X) - 2

# Calculating line values x and y
x = np.linspace(min_x, max_x, 1000)
y = w1 + w0 * x

plt.plot(x, y, color='#52b920', label='Regression Line')
plt.scatter(X, Y, c='#ef4423', label='Scatter Plot')


plt.legend()
plt.show()
```

Output:

Value of w0, w1: 0.4 2.4

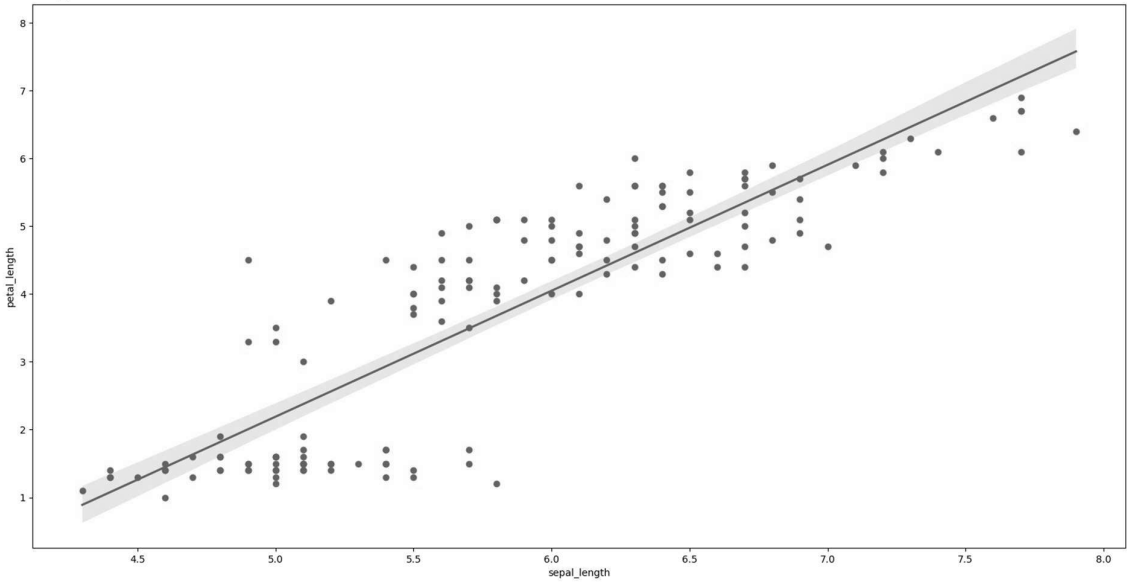


Code with inbuilt function:

```
import pandas as pd
import numpy as np
from sklearn.datasets import load_iris
from sklearn.linear_model import LinearRegression

iris = load_iris()

df = pd.DataFrame(data= np.c_[iris['data'], iris['target']],
                  columns= iris['feature_names'] + ['target'])

X = df[['sepal length (cm)']]
y = df['petal length (cm)']

reg = LinearRegression()

reg.fit(X, y)

print('Slope:', reg.coef_[0])
print('Intercept:', reg.intercept_)

new_X = [[6.5]]
print('Predicted Y:', reg.predict(new_X))

import matplotlib.pyplot as plt
import seaborn as sns

# load the Iris dataset
```

```
iris = sns.load_dataset('iris')

sns.scatterplot(x='sepal_length', y='petal_length', data=iris)
sns.regplot(x='sepal_length', y='petal_length', data=iris)
plt.show()
```

Output:



Conclusion: We successfully implemented Linear Regression.

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Continuous Assessment for Laboratory / Assignment sessions

Academic Year 2022-23

Name: Ayush Jain SAP ID: 60004200182

Course: Machine Learning Laboratory Course Code: **DJ19CEEL6021**

Year: **T.Y. B.Tech.** Sem: **VII** Batch: B 3

Department: Computer Engineering

Performance Indicators (Any no. of Indicators) (Maximum 5 marks per indicator)	1	2	3	4	5	6	7	8	9	10	11	Σ	Avg	A1	A2	Σ	Avg
Course Outcome	2, 4	2, 4	2, 4	2, 4	2, 4	3	2, 4	2, 4	5								
1. Knowledge (Factual/Conceptual/Procedural/ Metacognitive)	4	5	5	5													
2. Describe (Factual/Conceptual/Procedural/ Metacognitive)	4	4	4	5													
3. Demonstration (Factual/Conceptual/Procedural/ Metacognitive)	5	5	5	5													
4. Strategy (Analyse & / or Evaluate) (Factual/Conceptual/ Procedural/Metacognitive)	4	4	4	4													
5. Interpret/ Develop (Factual/Conceptual/ Procedural/Metacognitive)	–	–	–	–													
6. Attitude towards learning (receiving, attending, responding, valuing, organizing, characterization by value)	4	4	4	4													
7. Non-verbal communication skills/ Behaviour or Behavioural skills (motor skills, hand-eye coordination, gross body movements, finely coordinated body movements speech behaviours)	–	–	–	–													
Total	21	22	22	23													
Signature of the faculty member																	

Outstanding (5), Excellent (4), Good (3), Fair (2), Needs Improvement (1)

Laboratory marks Σ Avg. =	Assignment marks Σ Avg. =	Total Term-work (25) =
Laboratory Scaled to (15) =	Assignment Scaled to (10) =	Sign of the Student:

Signature of the Faculty member:
Name of the Faculty member:

Signature of Head of the Department
Date:

Machine Learning

Experiment 2

Ayush Jain 60004200132 B3

Aim : To implement univariate logistical regression in python with and without using any inbuilt function.

Theory:
Logistic regression is a type of regression analysis used to predict the probability of a binary outcome (i.e., 0 or 1). It is commonly used in machine learning and statistics to model relationships between a set of predictor variables (also called independent variables or features) and a binary outcome variable (also called a dependent variable or response).

The logistic regression model is based on the logistic function (also called the sigmoid function), which maps any real-valued input to an output between 0 and 1. This output represents the probability of the positive class (i.e., 1) given the input values. The formula for the logistic function is:

It is a special case of linear regression where the target variable is categorical in nature. It uses a log of odds as the dependent variable. Logistic Regression predicts the probability of occurrence of a binary event utilizing a logit function.

Linear Regression Equation:

$y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_n X_n$

Where, y is a dependent variable and x1, x2 ... and Xn are explanatory variables.

Sigmoid Function:

$$p = \frac{1}{1 + e^{-y}}$$

Apply Sigmoid function on linear regression:

$$p = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_n X_n)}}$$

Properties of Logistic Regression:

- The dependent variable in logistic regression follows Bernoulli Distribution.
- Estimation is done through maximum likelihood.

- No R Square, Model fitness is calculated through Concordance, KS-Statistics.

To apply logistic regression to a dataset, we typically first split the data into training and testing sets. Then, we fit a logistic regression model to the training data, using the predictor variables to predict the binary outcome variable. The logistic regression model estimates the probability of the positive class given the input values, based on the values of the predictor variables and the learned model parameters.

To evaluate the performance of the logistic regression model, we typically use metrics such as accuracy, precision, recall, and F1 score. These metrics provide an assessment of how well the model is able to predict the binary outcome, and can be used to compare different models or parameter settings.

Code without inbuilt function:

```
class LogisticRegression(object):

    def __init__(self, eta=0.01, n_iter=1000):
        self.eta = eta
        self.n_iter = n_iter

    def fit(self, X, y):
        self.w_ = np.zeros(1 + X.shape[1])
        self.cost_ = []
        for i in range(self.n_iter):
            y_val = self.activation(X)
            errors = (y - y_val)
            neg_grad = X.T.dot(errors)
            self.w_[1:] += self.eta * neg_grad
            self.w_[0] += self.eta * errors.sum()
            self.cost_.append(self._logit_cost(y, self.activation(X)))
        return self

    def _logit_cost(self, y, y_val):
        logit = -y.dot(np.log(y_val)) - ((1 - y).dot(np.log(1 - y_val)))
        return logit

    def _sigmoid(self, z):
        return 1.0 / (1.0 + np.exp(-z))

    def net_input(self, X):
        """Calculate net input"""
```

```
        return np.dot(X, self.w_[1:]) + self.w_[0]

    def activation(self, X):
        """ Activate the logistic neuron"""
        z = self.net_input(X)
        return self._sigmoid(z)

      def predict(self, X):

        # equivalent to np.where(self.activation(X) >= 0.5, 1, 0)
        return np.where(self.net_input(X) >= 0.0, 1, 0)

import pandas as pd

df = pd.read_csv('https://archive.ics.uci.edu/ml/'
        'machine-learning-databases/iris/iris.data', header=None)
df.tail()
```

	0	1	2	3	4
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

```
import numpy as np

# select setosa and versicolor
y = df.iloc[0:100, 4].values
y = np.where(y == 'Iris-setosa', 1, 0)

# extract sepal length and petal length
X = df.iloc[0:100, [0, 2]].values

# standardize features
X_std = np.copy(X)
X_std[:,0] = (X[:,0] - X[:,0].mean()) / X[:,0].std()
```

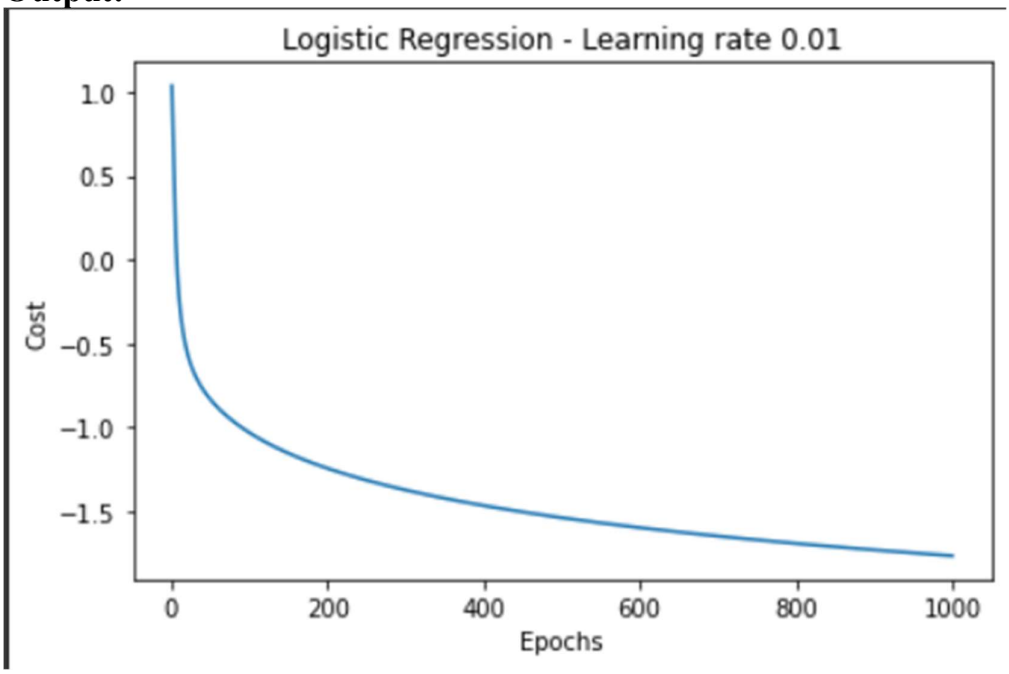
```
X_std[:,1] = (X[:,1] - X[:,1].mean()) / X[:,1].std()

from matplotlib.colors import ListedColormap

def plot_decision_regions(X, y, classifier, resolution=0.02):

    # setup marker generator and color map
    markers = ('s', 'x', 'o', '^', 'v')
    colors = ('red', 'blue', 'lightgreen', 'gray', 'cyan')
    cmap = ListedColormap(colors[:len(np.unique(y))])

    # plot the decision surface
    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution),
                           np.arange(x2_min, x2_max, resolution))
    Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)
    Z = Z.reshape(xx1.shape)
    plt.contourf(xx1, xx2, Z, alpha=0.4, cmap=cmap)
    plt.xlim(xx1.min(), xx1.max())
    plt.ylim(xx2.min(), xx2.max())

    # plot class samples
    for idx, cl in enumerate(np.unique(y)):
        plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1],
                    alpha=0.8, c=cmap(idx),
                    marker=markers[idx], label=cl)

score = logisticRegr.score(x_test, y_test)
print("Accuracy score: " + str(round(score*100.00,2)) + "%")

cm = metrics.confusion_matrix(y_test, predictions)
print("Confusion matrix: \n", cm)
```

```
Accuracy score: 95.93%
Confusion matrix:
 [[45  0  0  0  0  0  0  0  0  0]
 [ 0 48  0  0  0  0  1  0  2  1]
 [ 0  2 50  1  0  0  0  0  0  0]
 [ 0  0  0 53  0  0  0  0  1  0]
 [ 0  0  0  0 48  0  0  0  0  0]
 [ 0  0  0  0  0 53  1  0  0  3]
 [ 0  1  0  0  0  0 59  0  0  0]
 [ 0  0  0  0  1  0  0 52  0  0]
 [ 0  3  1  0  0  0  0  0 55  2]
 [ 0  0  0  1  0  1  0  0  0 55]]
```

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

```
lr = LogisticRegression(n_iter=1000, eta=0.2).fit(X_std, y)
plt.plot(range(1, len(lr.cost_) + 1), np.log10(lr.cost_))
plt.xlabel('Epochs')
plt.ylabel('Cost')
plt.title('Logistic Regression - Learning rate 0.01')

plt.tight_layout()
plt.show()
```

Output:



Code with inbuilt function:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing  import StandardScaler
from sklearn.pipeline import make_pipeline

data = pd.read_csv("data.csv")
data.drop(['Unnamed: 32',"id"], axis=1, inplace=True)
data.diagnosis = [1 if each == "M" else 0 for each in data.diagnosis]
y = data.diagnosis.values
x = data.drop(['diagnosis'], axis=1)

from sklearn.model_selection import train_test_split
```

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.15,
random_state=42)

from sklearn import linear_model
logreg =
make_pipeline(StandardScaler(),linear_model.LogisticRegression(random_state
= 42,max_iter= 150))
print("test accuracy: {} ".format(logreg.fit(x_train, y_train).score(x_test,
y_test)))
print("train accuracy: {} ".format(logreg.fit(x_train, y_train).score(x_train,
y_train)))
```

Output:

```
test accuracy: 0.9767441860465116
train accuracy: 0.9875776397515528
```

Conclusion: We successfully implemented univariate logistical regression in python with and without using any inbuilt function.

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Continuous Assessment for Laboratory / Assignment sessions

Academic Year 2022-23

Name: Ayush Jain SAP ID: 60004200182

Course: Machine Learning Laboratory Course Code: **DJ19CEEL6021**

Year: **T.Y. B.Tech.** Sem: **VII** Batch: B 3

Department: Computer Engineering

Performance Indicators (Any no. of Indicators) (Maximum 5 marks per indicator)	1	2	3	4	5	6	7	8	9	10	11	Σ	Avg	A1	A2	Σ	Avg
Course Outcome	2, 4	2, 4	2, 4	2, 4	2, 4	3	2, 4	2, 4	5								
1. Knowledge (Factual/Conceptual/Procedural/ Metacognitive)	4	5	5	5													
2. Describe (Factual/Conceptual/Procedural/ Metacognitive)	4	4	4	5													
3. Demonstration (Factual/Conceptual/Procedural/ Metacognitive)	5	5	5	5													
4. Strategy (Analyse & / or Evaluate) (Factual/Conceptual/ Procedural/Metacognitive)	4	4	4	4													
5. Interpret/ Develop (Factual/Conceptual/ Procedural/Metacognitive)	–	–	–	–													
6. Attitude towards learning (receiving, attending, responding, valuing, organizing, characterization by value)	4	4	4	4													
7. Non-verbal communication skills/ Behaviour or Behavioural skills (motor skills, hand-eye coordination, gross body movements, finely coordinated body movements speech behaviours)	–	–	–	–													
Total	21	22	22	23													
Signature of the faculty member																	

Outstanding (5), Excellent (4), Good (3), Fair (2), Needs Improvement (1)

Laboratory marks Σ Avg. =	Assignment marks Σ Avg. =	Total Term-work (25) =
Laboratory Scaled to (15) =	Assignment Scaled to (10) =	Sign of the Student:

Signature of the Faculty member:
Name of the Faculty member:

Signature of Head of the Department
Date:

PLOT NO. U-15, JVPD SCHEME, BHAKTIVEDANTA SWAMI MARG, VILE PARLE (WEST), MUMBAI - 400056
Tel.: 42335000/42335001 Email : info@djsce.ac.in / admin@djsce.ac.in Website : www.djsce.ac.in

Machine Learning

Experiment 3

Ayush Jain 60004200132 B3

Aim : To implement CART in python without using any inbuilt function and with inbuilt function.

Theory:

Introduction:
CART (Classification and Regression Tree) is a decision tree algorithm that can be used for both classification and regression tasks. In regression problems, CART creates a binary tree that recursively splits the data into subsets based on the value of a selected feature until a stopping criterion is met. The algorithm works by evaluating the impurity of each split and selecting the feature that produces the purest subsets possible.

Gini Index:
The Gini index is a metric used to evaluate the quality of a split in the CART algorithm. It measures the probability of incorrectly classifying a randomly chosen element from the set. In the CART algorithm, the Gini index is used to evaluate the impurity of a node. A split is made on the feature that produces the lowest Gini index, resulting in the purest subsets possible.
The formula for Gini index is as follows:

Gini Index = 1 - ∑(pi)^2

Where pi is the probability of an element being classified to a particular class. The Gini index ranges from 0 to 1, with a value of 0 indicating that all elements in the node belong to the same class, and a value of 1 indicating that the elements are uniformly distributed across all classes.

CART Algorithm for Classification
Here is the approach for most decision tree algorithms at their most simplest. The tree will be constructed in a top-down approach as follows:
Step 1: Start at the root node with all training instances
Step 2: Select an attribute on the basis of splitting criteria (Gain Ratio or other impurity metrics, discussed below)
Step 3: Partition instances according to selected attribute recursively

Partitioning stops when:

- There are no examples left
- All examples for a given node belong to the same class
- There are no remaining attributes for further partitioning – majority class is the leaf

Example:

Let's start with a simple example. Assume you have a bunch of oranges and mandrins with labels on them, and you want to identify a set of simple rules that you can use in the future to distinguish between these two types of fruit.

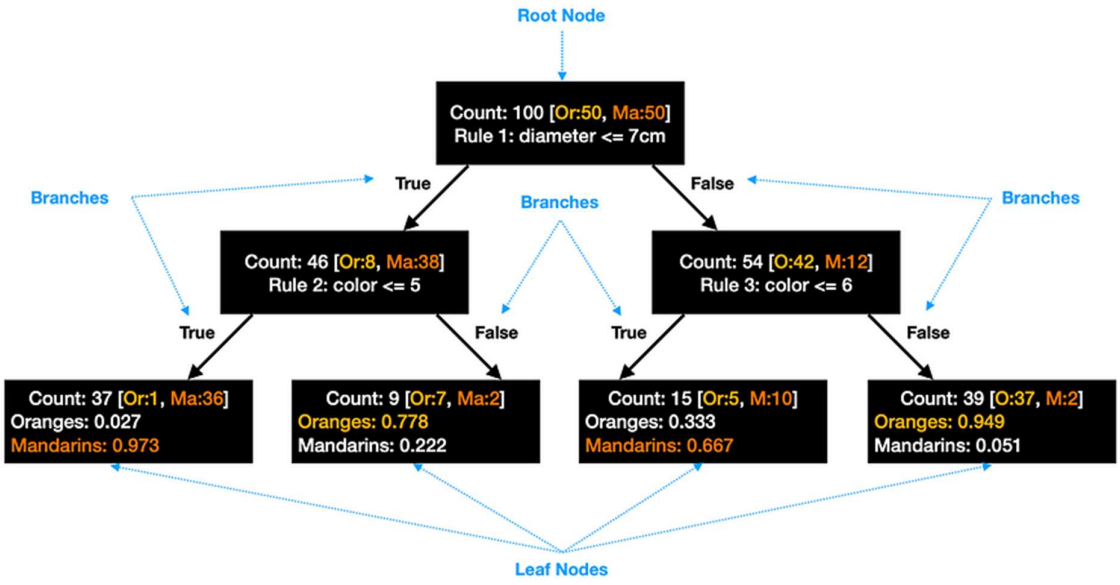
Typically, oranges (diameter 6–10cm) are bigger than mandarins (diameter 4–8cm), so the first rule found by your algorithm might be based on size:

- Diameter ≤ 7cm.

Next, you may notice that mandarins tend to be slightly darker in color than oranges. So, you use a color scale (1=dark to 10=light) to split your tree further:

- Color ≤5 for the left side of the sub-tree
- Color ≤6 for the right side of the sub-tree

Your final result is a tree that consists of 3 simple rules that help you to correctly distinguish between oranges and mandarins in the majority of the cases:



Code without inbuilt function:

```
import pandas as pd
import numpy as np
```

```
# Load the dataset
df = pd.read_csv("data.csv")

# Define a function to calculate the Gini index for a given attribute value
def calc_gini_for_attribute(class_name, col, df, target_col='Buys_computer'):
    total_count = len(df[df[col].isin([class_name])])
    count_of_1 = len(df[(df[col].isin([class_name])) & (df[target_col] == 1)])
    count_of_0 = len(df[(df[col].isin([class_name])) & (df[target_col] == 0)])
    prob_of_1 = count_of_1 / total_count
    prob_of_0 = count_of_0 / total_count
    gini = 1 - (prob_of_1 **2) - (prob_of_0**2)
    return gini, total_count

# Define a function to calculate the Gini index for all attributes
def calc_gini(cols: list, data, target_col='Buys_computer'):
    gini_dict = {}
    for col in cols:
        gini_for_attr = 0
        for value in list(data[col].unique()):
            gini_val, var_count = calc_gini_for_attribute(value, col, data)
            gini_for_attr += var_count/len(data) * gini_val
        gini_dict[col] = round(gini_for_attr, 3)
    return gini_dict

# Convert the "Buys_computer" feature to a categorical variable
df["Buys_computer"] = pd.Categorical(df["Buys_computer"])

# Get the unique classes of the "Buys_computer" feature
classes = df["Buys_computer"].unique()

# Loop through each attribute in the dataset
for attribute in df.columns[:-1]:
    # Convert the attribute to a categorical variable
    df[attribute] = pd.Categorical(df[attribute])
    # Calculate the Gini index for the attribute
    gini_dict = calc_gini([attribute], df, target_col='Buys_computer')
    # Print the Gini index for the attribute
    print("Gini Index of", attribute + ":", gini_dict[attribute])
```

Output:

```
Gini Index of Age: 1.0
Gini Index of Income: 1.0
Gini Index of Student: 1.0
Gini Index of Credit_rating: 1.0
Gini Index of Buys_computer: 1.0
```

Code with inbuilt function:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn import tree
from sklearn import metrics

pd.options.display.max_columns=50
df=pd.read_csv('weatherAUS.csv', encoding='utf-8')
df=df[pd.isnull(df['RainTomorrow'])==False]
df=df.fillna(df.mean())
df['RainTodayFlag']=df['RainToday'].apply(lambda x: 1 if x=='Yes' else 0)
df['RainTomorrowFlag']=df['RainTomorrow'].apply(lambda x: 1 if x=='Yes'
else 0)

X=df[['MinTemp', 'MaxTemp', 'Rainfall', 'Evaporation', 'Sunshine', 'WindGustS
peed',
    'WindSpeed9am', 'WindSpeed3pm', 'Humidity9am', 'Humidity3pm', 'Pressur
e9am',
    'Pressure3pm', 'Cloud9am', 'Cloud3pm', 'Temp9am', 'Temp3pm', 'RainToday
Flag']]
y=df['RainTomorrowFlag'].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_st
ate=0)

model = tree.DecisionTreeClassifier(criterion="gini",
                                    splitter="best")
clf = model.fit(X_train, y_train)


train = model.predict(X_train)
print(train)
test = model.predict(X_test)
```

```
print(test)
print("Training accuracy : ",metrics.accuracy_score(y_train , train))
print("Testing accuracy  : ",metrics.accuracy_score(y_test , test))
```

Output:

```
[0 0 0 ... 0 0 1]
[0 0 0 ... 0 0 1]
Training accuracy :  0.9999648364013574
Testing accuracy  :  0.7862090790815429
```

Conclusion: We implemented CART in python without using any inbuilt function and with inbuilt function.

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Continuous Assessment for Laboratory / Assignment sessions

Academic Year 2022-23

Name: Ayush Jain SAP ID: 60004200182

Course: Machine Learning Laboratory Course Code: **DJ19CEEL6021**

Year: **T.Y. B.Tech.** Sem: **VII** Batch: B 3

Department: Computer Engineering

Performance Indicators (Any no. of Indicators) (Maximum 5 marks per indicator)	1	2	3	4	5	6	7	8	9	10	11	Σ	Avg	A1	A2	Σ	Avg
Course Outcome	2, 4	2, 4	2, 4	2, 4	2, 4	3	2, 4	2, 4	5								
1. Knowledge (Factual/Conceptual/Procedural/ Metacognitive)	4	5	5	5													
2. Describe (Factual/Conceptual/Procedural/ Metacognitive)	4	4	4	5													
3. Demonstration (Factual/Conceptual/Procedural/ Metacognitive)	5	5	5	5													
4. Strategy (Analyse & / or Evaluate) (Factual/Conceptual/ Procedural/Metacognitive)	4	4	4	4													
5. Interpret/ Develop (Factual/Conceptual/ Procedural/Metacognitive)	–	–	–	–													
6. Attitude towards learning (receiving, attending, responding, valuing, organizing, characterization by value)	4	4	4	4													
7. Non-verbal communication skills/ Behavior or Behavioural skills (motor skills, hand-eye coordination, gross body movements, finely coordinated body movements speech behaviours)	–	–	–	–													
Total	21	22	22	23													
Signature of the faculty member																	

Outstanding (5), Excellent (4), Good (3), Fair (2), Needs Improvement (1)

Laboratory marks Σ Avg. =	Assignment marks Σ Avg. =	Total Term-work (25) =
Laboratory Scaled to (15) =	Assignment Scaled to (10) =	Sign of the Student:

Signature of the Faculty member:
Name of the Faculty member:

Signature of Head of the Department
Date:

Machine Learning

Experiment 4

Ayush Jain 60004200132 B3

Aim : To implement PCA in python without using any inbuilt function and without inbuilt function.

Theory:

What is Principal Component Analysis (PCA)?

In short, PCA is a dimensionality reduction technique that transforms a set of features in a dataset into a smaller number of features called principal components while at the same time trying to retain as much information in the original dataset as possible:

The key aim of PCA is to reduce the number of variables of a data set, while preserving as much information as possible.

Instead of explaining the theory of how PCA works in this article, I shall leave the explanation to the following video, which provides an excellent walkthrough of how PCA works.

So what are the advantages of using PCA on your dataset? Here are several reasons why you want to use PCA:

- Removes correlated features. PCA will help you remove all the features that are correlated, a phenomenon known as *multi-collinearity*. Finding features that are correlated is time consuming, especially if the number of features is large.
- Improves machine learning algorithm performance. With the number of features reduced with PCA, the time taken to train your model is now significantly reduced.
- Reduce overfitting. By removing the unnecessary features in your dataset, PCA helps to overcome overfitting.

On the other hand, PCA has its disadvantages:

- Independent variables are now less interpretable. PCA reduces your features into smaller number of components. Each component is now a

linear combination of your original features, which makes it less readable and interpretable.
- Information loss. Data loss may occur if you do not exercise care in choosing the right number of components.
- Feature scaling. Because PCA is a *variance maximizing* exercise, PCA requires features to be scaled prior to processing.

PCA is useful in cases where you have a large number of features in your dataset.

Steps for PCA algorithm

1. **Getting the dataset**
   Firstly, we need to take the input dataset and divide it into two subparts X and Y, where X is the training set, and Y is the validation set.
2. **Representing data into a structure**
   Now we will represent our dataset into a structure. Such as we will represent the two-dimensional matrix of independent variable X. Here each row corresponds to the data items, and the column corresponds to the Features. The number of columns is the dimensions of the dataset.
3. **Standardizing the data**
   In this step, we will standardize our dataset. Such as in a particular column, the features with high variance are more important compared to the features with lower variance.
   If the importance of features is independent of the variance of the feature, then we will divide each data item in a column with the standard deviation of the column. Here we will name the matrix as Z.
4. **Calculating the Covariance of Z**
   To calculate the covariance of Z, we will take the matrix Z, and will transpose it. After transpose, we will multiply it by Z. The output matrix will be the Covariance matrix of Z.
5. **Calculating the Eigen Values and Eigen Vectors**
   Now we need to calculate the eigenvalues and eigenvectors for the resultant covariance matrix Z. Eigenvectors or the covariance matrix are the directions of the axes with high information. And the coefficients of these eigenvectors are defined as the eigenvalues.
6. **Sorting the Eigen Vectors**
   In this step, we will take all the eigenvalues and will sort them in decreasing order, which means from largest to smallest. And

simultaneously sort the eigenvectors accordingly in matrix P of eigenvalues. The resultant matrix will be named as P*.

7. **Calculating the new features Or Principal Components**
   Here we will calculate the new features. To do this, we will multiply the P* matrix to the Z. In the resultant matrix Z*, each observation is the linear combination of original features. Each column of the Z* matrix is independent of each other.
8. **Remove less or unimportant features from the new dataset.**
   The new feature set has occurred, so we will decide here what to keep and what to remove. It means, we will only keep the relevant or important features in the new dataset, and unimportant features will be removed out.

Code without inbuilt function:

```
import numpy as np

def PCA(X, num_components):

    X = (X - np.mean(X, axis=0)) / np.std(X, axis=0)

    cov_matrix = np.cov(X.T)

    eigenvalues, eigenvectors = np.linalg.eig(cov_matrix)


    indices = np.argsort(eigenvalues)[::-1][:num_components]
    top_eigenvectors = eigenvectors[:, indices]

    transformed_data = np.dot(X, top_eigenvectors)

    return transformed_data

X = np.array([
    [7, 2, 3],
    [4, 5, 6],
    [7, 8, 9],
    [10, 11, 12]
])

X_pca = PCA(X, 2)
print(X_pca)
```

Output:

```
[[-1.63655433  0.96004684]
 [-1.26109477 -0.89979996]
 [ 0.54551811 -0.32001561]
 [ 2.35213099  0.25976874]]
```

Code with inbuilt function:

```
import numpy as np
from sklearn.decomposition import PCA

data = np.array([[ 0.6, -0.2,  0.4,  0.4,  0.3],
                 [-0.2,  0.7, -0.5, -0.1, -0.3],
                 [ 0.3, -0.5,  0.9,  0.3,  0.2],
                 [ 0.2, -0.1,  0.3,  0.8,  0.7],
                 [ 0.5, -0.3,  0.2,  0.7,  0.6],
                 [-0.1,  0.8, -0.7, -0.3, -0.5],
                 [ 0.2, -0.5,  0.3,  0.6,  0.8],
                 [ 0.4, -0.1,  0.2,  0.7,  0.6],
                 [-0.2,  0.6, -0.4, -0.2, -0.4],
                 [ 0.1, -0.4,  0.6,  0.2,  0.2]])
n_train = 8
X_train = data[:n_train, :] # replace n_train with the number of training samples
X_val = data[n_train:, :] # replace n_train with the number of validation sample
s

X_train_std = (X_train - X_train.mean(axis=0)) / X_train.std(axis=0)

cov_mat = np.cov(X_train_std.T)
eig_vals, eig_vecs = np.linalg.eig(cov_mat)
eig_pairs = [(np.abs(eig_vals[i]), eig_vecs[:, i]) for i in range(len(eig_vals))]
eig_pairs.sort(reverse=True, key=lambda x: x[0])
sorted_eig_vecs = np.stack([eig_pairs[i][1] for i in range(len(eig_vals))], axis=1
)
X_train_pca = X_train_std.dot(sorted_eig_vecs)

n_components =2
X_train_pca = X_train_pca[:, :n_components]
X_val_std = (X_val - X_train.mean(axis=0)) / X_train.std(axis=0)
X_val_pca = X_val_std.dot(sorted_eig_vecs[:, :n_components])
X_val_pca
```

Output:

```
array([[-3.24368083, -0.19499751],
       [ 0.24929579, -0.91519733]])
```

Conclusion: We successfully implemented PCA in python without using any inbuilt function and without inbuilt function.

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Continuous Assessment for Laboratory / Assignment sessions

Academic Year 2022-23

Name: Ayush Jain SAP ID: 60004200182

Course: Machine Learning Laboratory Course Code: **DJ19CEEL6021**

Year: **T.Y. B.Tech.** Sem: **VII** Batch: B 3

Department: Computer Engineering

Performance Indicators (Any no. of Indicators) (Maximum 5 marks per indicator)	1	2	3	4	5	6	7	8	9	10	11	Σ	Avg	A1	A2	Σ	Avg
Course Outcome	2, 4	2, 4	2, 4	2, 4	2, 4	3	2, 4	2, 4	5								
1. Knowledge (Factual/Conceptual/Procedural/ Metacognitive)	4	5	5	5													
2. Describe (Factual/Conceptual/Procedural/ Metacognitive)	4	4	4	5													
3. Demonstration (Factual/Conceptual/Procedural/ Metacognitive)	5	5	5	5													
4. Strategy (Analyse & / or Evaluate) (Factual/Conceptual/ Procedural/Metacognitive)	4	4	4	4													
5. Interpret/ Develop (Factual/Conceptual/ Procedural/Metacognitive)	–	–	–	–													
6. Attitude towards learning (receiving, attending, responding, valuing, organizing, characterization by value)	4	4	4	4													
7. Non-verbal communication skills/ Behaviour or Behavioural skills (motor skills, hand-eye coordination, gross body movements, finely coordinated body movements speech behaviours)	–	–	–	–													
Total	21	22	22	23													
Signature of the faculty member																	

Outstanding (5), Excellent (4), Good (3), Fair (2), Needs Improvement (1)

Laboratory marks Σ Avg. =	Assignment marks Σ Avg. =	Total Term-work (25) =
Laboratory Scaled to (15) =	Assignment Scaled to (10) =	Sign of the Student:

Signature of the Faculty member:
Name of the Faculty member:

Signature of Head of the Department
Date:

PLOT NO. U-15, JVPD SCHEME, BHAKTIVEDANTA SWAMI MARG, VILE PARLE (WEST), MUMBAI - 400056
Tel.: 42335000/42335001 Email : info@djsce.ac.in / admin@djsce.ac.in Website : www.djsce.ac.in

Machine Learning

Experiment 5

Ayush Jain 60004200132 B3

Aim : To implement KNN in python without using any inbuilt function and without inbuilt function.

Theory:

A **classification problem** has a discrete value as its output. For example, “likes pineapple on pizza” and “does not like pineapple on pizza” are discrete. There is no middle ground. The analogy above of teaching a child to identify a pig is another example of a classification problem.

Age	Likes Pinapple on Pizza
42	1
65	1
50	1
76	1
96	1
50	1
91	0
58	1
25	1
23	1
75	1
46	0
87	0
96	0
45	0
32	1
63	0
21	1
26	1
93	0
68	1
96	0

This image shows a basic example of what classification data might look like. We have a predictor (or set of predictors) and a label. In the image, we might be trying to predict whether someone likes pineapple (1) on their pizza or not (0) based on their age (the predictor).

It is standard practice to represent the output (label) of a classification algorithm as an integer number such as 1, -1, or 0. In this instance, these numbers are purely representational. Mathematical operations should not be performed on them because doing so would be meaningless. Think for a moment. What is "likes pineapple" + "does not like pineapple"? Exactly. We cannot add them, so we should not add their numeric representations.

A **regression problem** has a real number (a number with a decimal point) as its output. For example, we could use the data in the table below to estimate someone's weight given their height.

Height(Inches)	Weight(Pounds)
65.78	112.99
71.52	136.49
69.40	153.03
68.22	142.34
67.79	144.30
68.70	123.30
69.80	141.49
70.01	136.46
67.90	112.37
66.78	120.67
66.49	127.45
67.62	114.14
68.30	125.61
67.12	122.46
68.28	116.09

Data used in a regression analysis will look similar to the data shown in the image above. We have an independent variable (or set of independent variables) and a dependent variable (the thing we are trying to guess given our independent variables). For instance, we could say height is the independent variable and weight is the dependent variable.

Also, each row is typically called an **example, observation, or data point**, while each column (not including the label/dependent variable) is often called a **predictor, dimension, independent variable, or feature.**

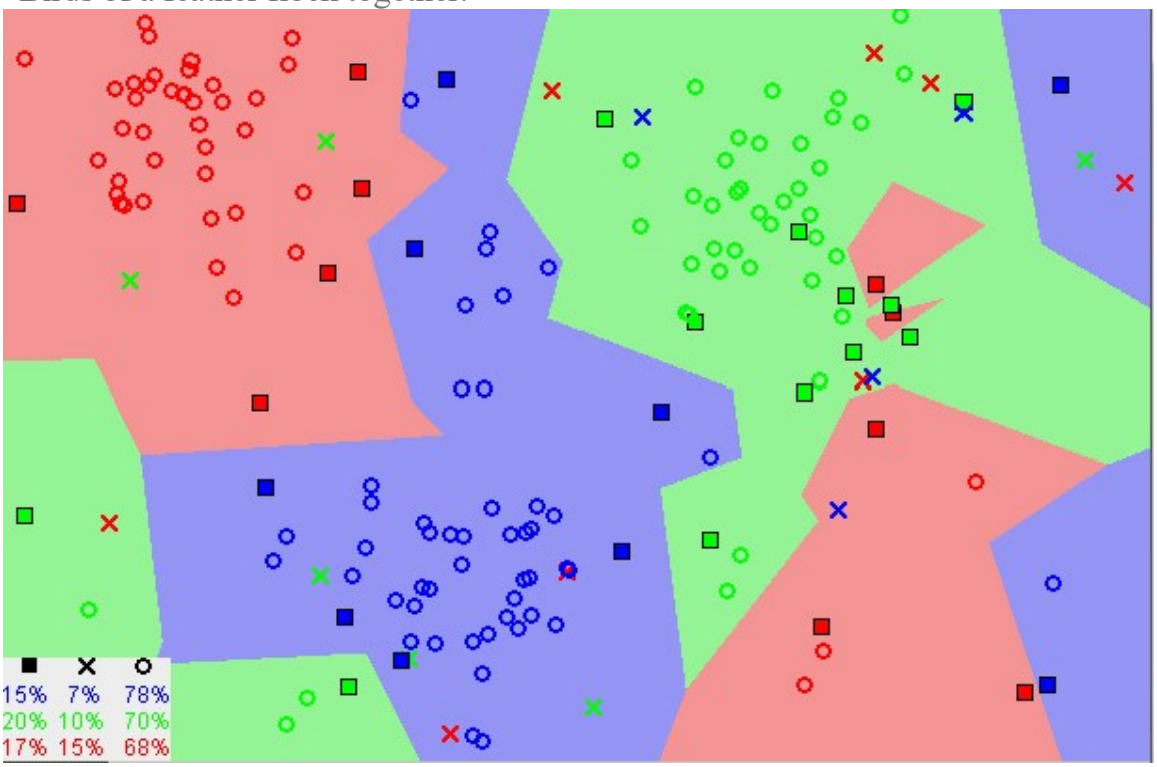
An **unsupervised machine learning** algorithm makes use of input data without any labels —in other words, no teacher (label) telling the child (computer) when it is right or when it has made a mistake so that it can self-correct.

Unlike supervised learning that tries to learn a function that will allow us to make predictions given some new unlabeled data, unsupervised learning tries to learn the basic structure of the data to give us more insight into the data.

K-Nearest Neighbors

The KNN algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each other.

"Birds of a feather flock together."



Notice in the image above that most of the time, similar data points are close to each other. The KNN algorithm hinges on this assumption being true enough for the algorithm to be useful. KNN captures the idea of similarity (sometimes called distance, proximity, or closeness) with some mathematics we might have learned in our childhood— calculating the distance between points on a graph.

There are other ways of calculating distance, and one way might be preferable depending on the problem we are solving. However, the straight-line distance (also called the Euclidean distance) is a popular and familiar choice.

The KNN Algorithm

1. Load the data
2. Initialize K to your chosen number of neighbors
3. For each example in the data

3.1 Calculate the distance between the query example and the current example from the data.

3.2 Add the distance and the index of the example to an ordered collection

4. Sort the ordered collection of distances and indices from smallest to largest (in ascending order) by the distances

5. Pick the first K entries from the sorted collection

6. Get the labels of the selected K entries

7. If regression, return the mean of the K labels

8. If classification, return the mode of the K labels

Code without inbuilt function:

```
import numpy as np
from collections import Counter

class KNN:

    def __init__(self, k):
        self.k = k

    def fit(self, X, y):
        self.X_train = X
        self.y_train = y

    def euclidean_distance(self, X):
        return np.sqrt(np.sum((self.X_train - X)**2, axis=1))

    def predict(self, X):
        y_pred = []

        for i in range(len(X)):
            distances = self.euclidean_distance(X[i])
            indices = np.argsort(distances)[:self.k]
            k_nearest_labels = [self.y_train[idx] for idx in indices]
            most_common_label = Counter(k_nearest_labels).most_common(1)[0][
0]
            y_pred.append(most_common_label)

        return np.array(y_pred)
```

```
InterviewS = [70, 70, 30, 10]
ExamR = [70, 40, 40, 40]
TypeH = ['Not Hired', 'Hired', 'Not Hired', 'Not Hired']

X = np.array(list(zip(InterviewS, ExamR)))
y = np.array(TypeH)

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

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

knn = KNN(k=3)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)

# Calculate the accuracy of the model
# accuracy = accuracy_score(y_test, y_pred)
# print(f"Accuracy: {accuracy:.2f}")

knn = KNN(k=3)
knn.fit(X, y)

# Input new data to classify
new_data = np.array([[80, 40]])

predicted_label = knn.predict(new_data)

print(f"Predicted label: {predicted_label[0]}")
```

Output:

```
Predicted label: Not Hired
```

Code with inbuilt function:

```
import numpy as np

InterviewS = [90, 60, 70, 50, 80, 70, 60, 40, 90, 80]
ExamR = [80, 50, 60, 40, 90, 50, 70, 60, 60, 70]
```

```
TypeH = ['Hired', 'Not Hired', 'Hired', 'Not Hired', 'Hired', 'Not Hired', 'Hired', '
Not Hired', 'Hired', 'Hired']

X = np.array(list(zip(InterviewS, ExamR)))
y = np.array(TypeH)

from sklearn.neighbors import KNeighborsClassifier

# Create KNN classifier
knn = KNeighborsClassifier(n_neighbors=3)

# Fit the classifier to the data
knn.fit(X, y)

# Predict if a person with interview score 70 and exam rank 80 will be hired or
not
X_new = np.array([[70, 80]])
prediction = knn.predict(X_new)

print("Prediction:", prediction)
```

Output:

```
Prediction: ['Hired']
```

Conclusion: We implemented KNN in python without using any inbuilt function and without inbuilt function.

Shri Vile Parle Kelavani Mandal's

DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING

(Autonomous College Affiliated to the University of Mumbai)

NAAC Accredited with "A" Grade (CGPA : 3.18)

Continuous Assessment for Laboratory / Assignment sessions

Academic Year 2022-23

Name: Ayush Jain SAP ID: 60004200132

Course: Machine Learning Laboratory Course Code: **DJ19CEEL6021**

Year: **T.Y. B.Tech.** Sem: **VII** Batch: B 3

Department: Computer Engineering

Performance Indicators (Any no. of Indicators) (Maximum 5 marks per indicator)	1	2	3	4	5	6	7	8	9	10	11	Σ	Avg	A1	A2	Σ	Avg
Course Outcome	2, 4	2, 4	2, 4	2, 4	2, 4	3	2, 4	2, 4	5								
1. Knowledge (Factual/Conceptual/Procedural/ Metacognitive)	4	5	5	5	5	5	5	5									
2. Describe (Factual/Conceptual/Procedural/ Metacognitive)	4	4	4	5	5	4	4	4									
3. Demonstration (Factual/Conceptual/Procedural/ Metacognitive)	5	5	5	5	5	5	5	4									
4. Strategy (Analyse & / or Evaluate) (Factual/Conceptual/ Procedural/Metacognitive)	4	4	4	4	4	5	5	5									
5. Interpret/ Develop (Factual/Conceptual/ Procedural/Metacognitive)	–	–	–	–	–	–	–	–									
6. Attitude towards learning (receiving, attending, responding, valuing, organizing, characterization by value)	4	4	4	4	5	4	4	5									
7. Non-verbal communication skills/ Behaviour or Behavioural skills (motor skills, hand-eye coordination, gross body movements, finely coordinated body movements speech behaviours)	–	–	–	–	–	–	–	–									
Total	21	22	22	23	24	23	23	23									
Signature of the faculty member																	

Outstanding (5), Excellent (4), Good (3), Fair (2), Needs Improvement (1)

Laboratory marks Σ Avg. =	Assignment marks Σ Avg. =	Total Term-work (25) =
Laboratory Scaled to (15) =	Assignment Scaled to (10) =	Sign of the Student:

Signature of the Faculty member:
Name of the Faculty member:

Signature of Head of the Department
Date:

Machine Learning

Experiment 6

Ayush Jain 60004200132 B3

Aim : To implement Backpropagation in python without using any inbuilt function.

Theory:

Backpropagation is a widely used algorithm for training artificial neural networks (ANNs). It is a supervised learning algorithm that involves propagating errors backwards through the layers of the network, in order to adjust the weights and biases of the neurons. The goal of backpropagation is to minimize the difference between the predicted output of the network and the actual output, which is known as the error or loss.

The backpropagation algorithm works by first making a forward pass through the network, in which the inputs are propagated through the layers of neurons until they reach the output layer. The output of the network is then compared to the desired output, and the error or loss is calculated. This error is then propagated backwards through the layers of neurons, starting at the output layer and working backwards towards the input layer.

During the backward pass, the algorithm calculates the gradient of the error with respect to each weight and bias in the network. The gradient is a measure of how much the error changes when the weight or bias is changed, and it is used to update the weights and biases in a way that reduces the error.

The backpropagation algorithm can be broken down into the following steps:

1. Initialize the weights and biases: The weights and biases of the neurons in the network are randomly initialized before training begins.

2. Make a forward pass: The inputs are propagated through the layers of neurons until they reach the output layer, and the output of the network is calculated.

3. Calculate the error: The difference between the predicted output and the actual output is calculated, and this is used to calculate the error or loss.

4. Propagate the error backwards: The error is propagated backwards through the layers of neurons, starting at the output layer and working backwards towards the input layer.

5. Calculate the gradient: The gradient of the error with respect to each weight and bias in the network is calculated.

6. Update the weights and biases: The weights and biases are updated in a way that reduces the error, using the gradients calculated in step 5.

7. Repeat: Steps 2-6 are repeated for a specified number of epochs or until the error is minimized to a satisfactory level.

Advantages:

Sure, here are some points elaborating on the advantages of backpropagation:

1. Backpropagation is a powerful algorithm that can be used to train a wide range of neural network architectures, including feedforward neural networks, convolutional neural networks, and recurrent neural networks. This flexibility makes it a useful tool for a variety of machine learning tasks.
2. Backpropagation is relatively simple to implement, especially when compared to other optimization algorithms like genetic algorithms or simulated annealing. This simplicity makes it accessible to a wider range of users, including those with less experience in machine learning.
3. Backpropagation can be parallelized, which allows it to take advantage of the computational power of modern hardware. This makes it well-suited to large-scale training tasks, where the processing of large amounts of data can be a bottleneck.
4. Backpropagation is a gradient-based optimization algorithm, which means that it can be used to find the global minimum of the error function, as long as the error function is convex. This makes it a powerful tool for optimization tasks.
5. Backpropagation is an iterative algorithm, which means that it can continue to improve the performance of a neural network over time. This is particularly useful when dealing with complex problems or when the quality of the data is variable. Overall, the advantages of backpropagation make it a widely-used and powerful algorithm for training neural networks. While it does have some limitations, such as its tendency to get stuck in local minima, the benefits of backpropagation make it a key tool in the machine learning toolkit.

Disadvantages:

1. Backpropagation is a fundamental algorithm for training neural networks. It is a supervised learning algorithm that uses gradient descent to minimize the difference between the predicted output and the actual output.

2. Backpropagation has been used extensively in a wide range of machine learning tasks, including image recognition, natural language processing, and speech recognition. It has been shown to be effective in achieving state-of-the-art performance on many of these tasks.

3. While backpropagation is a powerful algorithm, it does have some limitations. For example, it can get stuck in local minima, which can prevent it from finding the global minimum of the error function. It can also suffer from the vanishing gradient problem, which can make it difficult to train deep neural networks.

4. Despite its limitations, backpropagation remains one of the most effective methods for training neural networks. Researchers have developed various techniques to mitigate

its limitations, such as adding regularization to the cost function or using alternative activation functions.

5. Backpropagation can be computationally expensive, particularly for large datasets or deep neural networks. However, it can be parallelized and optimized to take advantage of modern hardware, making it feasible to train large models.

6. Overall, backpropagation is a key tool in the machine learning toolkit. While it may not be the best choice for every problem, it remains a powerful and widely-used algorithm that has enabled significant advances in the field of machine learning. Overall, backpropagation is a powerful and widely-used algorithm for training neural networks. While it has some limitations, it remains one of the most effective methods for achieving state-of-the-art performance on a wide range of machine learning tasks.

Here is a visual representation of the backpropagation algorithm:

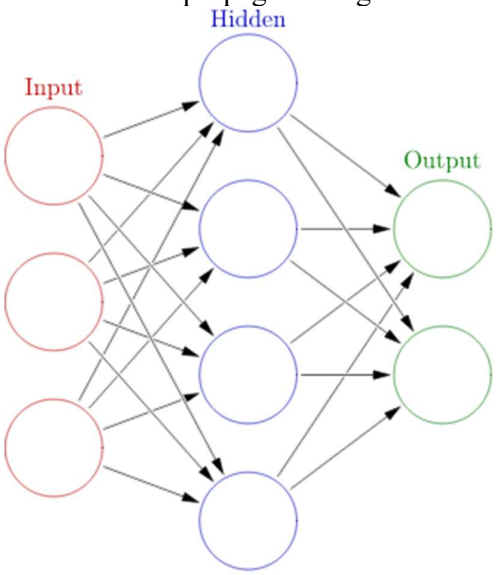


Fig: Backpropagation Algorithm

Algorithm:

1. Initialize the weights and biases
2. Make a forward pass and calculate the output of the network
3. Calculate the error or loss
4. Propagate the error backwards through the layers of neurons
5. Calculate the gradient of the error with respect to each weight and bias
6. Update the weights and biases using the gradients calculated in step 5 7. Repeat steps 2-6 for a specified number of epochs or until the error is minimized to a satisfactory level

Code:

```
import numpy as np

def sigmoid(x):
   return 1 / (1 + np.exp(-x))

def sigmoid_derivative(x):     return
sigmoid(x) * (1 - sigmoid(x))

def backpropagation(X, y, weights, learning_rate, epochs):
for i in range(epochs):         layer_1 = sigmoid(np.dot(X,
weights[0]))         output = sigmoid(np.dot(layer_1,
weights[1]))


       error = y - output
       delta_output = error * sigmoid_derivative(output)
       delta_layer_1 = np.dot(delta_output, weights[1].T) * sigmoid_derivative(la
yer_1)

       weights[1] += learning_rate * np.dot(layer_1.T, delta_output)
weights[0] += learning_rate * np.dot(X.T, delta_layer_1)

   return weights

X = np.array([[0, 0, 1], [0, 1, 1], [1, 0, 1], [1, 1, 1]]) y
= np.array([[0], [1], [1], [0]])

weights = [np.random.randn(3, 4), np.random.randn(4, 1)] learning_rate
= 0.5
epochs = 10000

# Call backpropagation function with sigmoid activation function
updated_weights_sigmoid = backpropagation(X, y, weights,learning_rate, epoc
hs)

updated_weights_sigmoid
# Call backpropagation function with ReLU activation function
updated_weights_relu = backpropagation(X, y, weights, learning_rate, epochs)
updated_weights_relu
```

Output:

```
[array([[  79.01230725,   55.95127262, -139.80242202,   -2.30674911],
        [ -79.84634984,    2.39269949,  138.98222477,   -1.57091725],
        [   0.94195722,   56.86632426,    1.01953494,    2.31077579]]),
 array([[-26.15611331],
        [ 22.97071048],
        [-28.69179373],
        [ 14.12472595]])]
```

Conclusion: We implemented Backpropogation in python without using any inbuilt function and without inbuilt function.

Shri Vile Parle Kelavani Mandal's

DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING

(Autonomous College Affiliated to the University of Mumbai)

NAAC Accredited with "A" Grade (CGPA : 3.18)

Continuous Assessment for Laboratory / Assignment sessions

Academic Year 2022-23

Name: Ayush Jain SAP ID: 60004200132

Course: Machine Learning Laboratory Course Code: **DJ19CEEL6021**

Year: **T.Y. B.Tech.** Sem: **VII** Batch: B 3

Department: Computer Engineering

Performance Indicators (Any no. of Indicators) (Maximum 5 marks per indicator)	1	2	3	4	5	6	7	8	9	10	11	Σ	Avg	A1	A2	Σ	Avg
Course Outcome	2, 4	2, 4	2, 4	2, 4	2, 4	3	2, 4	2, 4	5								
1. Knowledge (Factual/Conceptual/Procedural/ Metacognitive)	4	5	5	5	5	5	5	5									
2. Describe (Factual/Conceptual/Procedural/ Metacognitive)	4	4	4	5	5	4	4	4									
3. Demonstration (Factual/Conceptual/Procedural/ Metacognitive)	5	5	5	5	5	5	5	4									
4. Strategy (Analyse & / or Evaluate) (Factual/Conceptual/ Procedural/Metacognitive)	4	4	4	4	4	5	5	5									
5. Interpret/ Develop (Factual/Conceptual/ Procedural/Metacognitive)	–	–	–	–	–	–	–	–									
6. Attitude towards learning (receiving, attending, responding, valuing, organizing, characterization by value)	4	4	4	4	5	4	4	5									
7. Non-verbal communication skills/ Behaviour or Behavioural skills (motor skills, hand-eye coordination, gross body movements, finely coordinated body movements speech behaviours)	–	–	–	–	–	–	–	–									
Total	21	22	22	23	24	23	23	23									
Signature of the faculty member																	

Outstanding (5), Excellent (4), Good (3), Fair (2), Needs Improvement (1)

Laboratory marks Σ Avg. =	Assignment marks Σ Avg. =	Total Term-work (25) =
Laboratory Scaled to (15) =	Assignment Scaled to (10) =	Sign of the Student:

Signature of the Faculty member:
Name of the Faculty member:

Signature of Head of the Department
Date:

PLOT NO. U.15, JVPD SCHEME, BHAKTIVEDANTA SWAMI MARG, VILE PARLE (WEST), MUMBAI - 400056
Tel : 42335000/42335001 Email : info@djsce.ac.in / admin@djsce.ac.in Website : www.djsce.ac.in

Machine Learning

Experiment 7

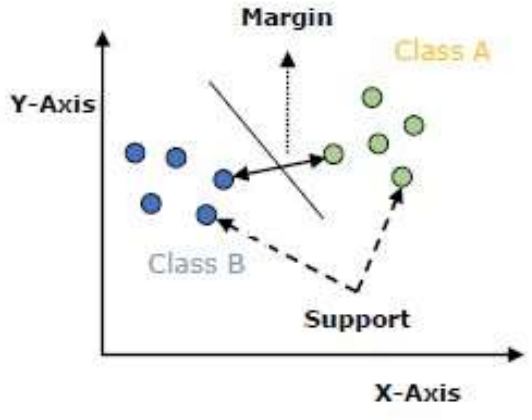
Ayush Jain 60004200132 B3

Aim : Implement SVM

Theory:

Support vector machines (SVMs) are powerful yet flexible supervised machine learning algorithms which are used both for classification and regression. But generally, they are used in classification problems. In 1960s, SVMs were first introduced but later they got refined in 1990. SVMs have their unique way of implementation as compared to other machine learning algorithms. Lately, they are extremely popular because of their ability to handle multiple continuous and categorical variables.

An SVM model is basically a representation of different classes in a hyperplane in multidimensional space. The hyperplane will be generated in an iterative manner by SVM so that the error can be minimized. The goal of SVM is to divide the datasets into classes to find a maximum marginal hyperplane (MMH).



The followings are important concepts in SVM −
Support Vectors − Datapoints that are closest to the hyperplane is called support vectors. Separating line will be defined with the help of these data points.

Hyperplane − As we can see in the above diagram, it is a decision plane or space which is divided between a set of objects having different classes.
Margin − It may be defined as the gap between two lines on the closet data points of different classes. It can be calculated as the perpendicular distance from the line to the support vectors. Large margin is considered as a good margin and small margin is considered as a bad margin.
The main goal of SVM is to divide the datasets into classes to find a maximum marginal hyperplane (MMH) and it can be done in the following two steps −
First, SVM will generate hyperplanes iteratively that segregates the classes in best way.

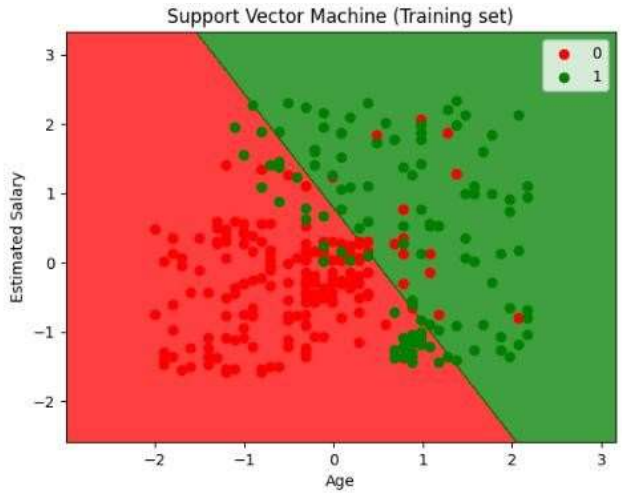
Then, it will choose the hyperplane that separates the classes correctly.

Code:

```
import pandas as pd import numpy as np import
matplotlib.pyplot as plt from sklearn.datasets import
load_digits from sklearn.model_selection import
train_test_split
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix

# Load the dataset digits
= load_digits()

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(digits.data, digits.target,
test_size=0.3, random_state=42)

# Define the list of kernel functions to be used
kernels = ['linear', 'poly', 'rbf', 'sigmoid']
best_accuracy = 0

# Loop through each kernel function and train the SVM for
kernel in kernels:
   # Create an SVM with the chosen kernel function
svm = SVC(kernel=kernel)

   # Fit the SVM to the training data
svm.fit(X_train, y_train)
```

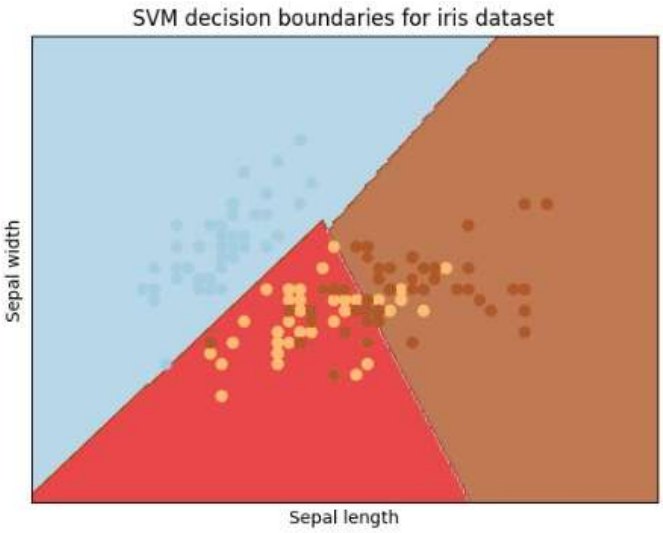
```
    # Evaluate the performance of the SVM on the testing data
y_pred = svm.predict(X_test)     cm =
confusion_matrix(y_test, y_pred)

    # Update the best-performing kernel     if
np.mean(np.diag(cm)) > best_accuracy:
best_accuracy = np.mean(np.diag(cm))
        best_kernel = kernel
        best_svm = svm

# Plot the decision boundary for the best-performing kernel
plt.figure(figsize=(8, 6))
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap=plt.cm.get_cmap('jet', 10),
edgecolors='k') plt.colorbar(ticks=range(10))
plt.title('Decision boundary for SVM with {} kernel'.format(best_kernel))
plt.show()
```

Output:



Multi-dimensional:

```
import numpy as np import
matplotlib.pyplot as plt from
sklearn import svm, datasets

data='/content/Test.csv'
X = iris.data[:, :2] y =
iris.target

C = 1.0
clf = svm.SVC(kernel='linear', C=C, decision_function_shape='ovr') clf.fit(X,
y)

x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1 xx,
yy = np.meshgrid(np.arange(x_min, x_max, 0.02),
                 np.arange(y_min, y_max, 0.02))

Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape) plt.contourf(xx, yy, Z,
cmap=plt.cm.Paired, alpha=0.8) plt.scatter(X[:, 0], X[:,
1], c=y, cmap=plt.cm.Paired) plt.xlabel('Sepal length')
plt.ylabel('Sepal width') plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.xticks(()) plt.yticks(())
plt.title('SVM decision boundaries for iris dataset') plt.show()
```



Conclusion: We successfully implemented SVM.

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Continuous Assessment for Laboratory / Assignment sessions

Academic Year 2022-23

Name: Ayush Jain SAP ID: 60004200132

Course: Machine Learning Laboratory Course Code: **DJ19CEEL6021**

Year: **T.Y. B.Tech.** Sem: **VII** Batch: B 3

Department: Computer Engineering

Performance Indicators (Any no. of Indicators) (Maximum 5 marks per indicator)	1	2	3	4	5	6	7	8	9	10	11	Σ	Avg	A1	A2	Σ	Avg
Course Outcome	2, 4	2, 4	2, 4	2, 4	2, 4	3	2, 4	2, 4	5								
1. Knowledge (Factual/Conceptual/Procedural/ Metacognitive)	4	5	5	5	5	5	5	5									
2. Describe (Factual/Conceptual/Procedural/ Metacognitive)	4	4	4	5	5	4	4	4									
3. Demonstration (Factual/Conceptual/Procedural/ Metacognitive)	5	5	5	5	5	5	5	4									
4. Strategy (Analyse & / or Evaluate) (Factual/Conceptual/ Procedural/Metacognitive)	4	4	4	4	4	5	5	5									
5. Interpret/ Develop (Factual/Conceptual/ Procedural/Metacognitive)	–	–	–	–	–	–	–	–									
6. Attitude towards learning (receiving, attending, responding, valuing, organizing, characterization by value)	4	4	4	4	5	4	4	5									
7. Non-verbal communication skills/ Behaviour or Behavioural skills (motor skills, hand-eye coordination, gross body movements, finely coordinated body movements speech behaviours)	–	–	–	–	–	–	–	–									
Total	21	22	22	23	24	23	23	23									
Signature of the faculty member																	

Outstanding (5), Excellent (4), Good (3), Fair (2), Needs Improvement (1)

Laboratory marks Σ Avg. =	Assignment marks Σ Avg. =	Total Term-work (25) =
Laboratory Scaled to (15) =	Assignment Scaled to (10) =	Sign of the Student:

Signature of the Faculty member:
Name of the Faculty member:

Signature of Head of the Department
Date:

Machine Learning

Experiment 8

Ayush Jain 60004200132 B3

Aim : Implement Bayesian Classification

Theory:

Bayesian classification uses Bayes theorem to predict the occurrence of any event. Bayesian classifiers are the statistical classifiers with the Bayesian probability understandings. The theory expresses how a level of belief, expressed as a probability.

Bayes theorem came into existence after Thomas Bayes, who first utilized conditional probability to provide an algorithm that uses evidence to calculate limits on an unknown parameter.

Bayes's theorem is expressed mathematically by the following equation that is given below.

$$P(X/Y) = \frac{P(Y/X)P(X)}{P(Y)}$$

Where X and Y are the events and P (Y) ≠ 0

P(X/Y) is a **conditional probability** that describes the occurrence of event **X** is given that **Y** is true.

P(Y/X) is a **conditional probability** that describes the occurrence of event **Y** is given that **X** is true.

P(X) and P(Y) are the probabilities of observing X and Y independently of each other. This is known as the **marginal probability**.

Bayesian interpretation:

In the Bayesian interpretation, probability determines a "**degree of belief**." Bayes theorem connects the degree of belief in a hypothesis before and after accounting

for evidence. For example, Lets us consider an example of the coin. If we toss a coin, then we get either heads or tails, and the percent of occurrence of either heads and tails is 50%. If the coin is flipped numbers of times, and the outcomes are observed, the degree of belief may rise, fall, or remain the same depending on the outcomes.

For proposition X and evidence Y,

- P(X), the prior, is the primary degree of belief in X
- P(X/Y), the posterior is the degree of belief having accounted for Y.
- The quotient $\frac{P(Y/X)}{P(Y)}$ represents the supports Y provides for X.

Bayes theorem can be derived from the conditional probability:

$$P(X/Y) = \frac{P(X \cap Y)}{P(Y)}, \text{ if } P(Y) \neq 0$$

$$P(Y/X) = \frac{P(Y \cap X)}{P(X)}, \text{ if } P(X) \neq 0$$

Where P (X⋂Y) is the **joint probability** of both X and Y being true, because

$$P(Y \cap X) = P(X \cap Y)$$

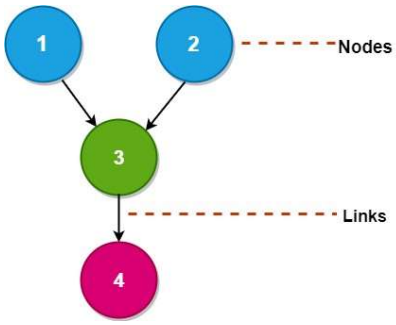
$$\text{or, } P(X \cap Y) = P(X/Y)P(Y) = P(Y/X)P(X)$$

$$\text{or, } P(X/Y) = \frac{P(Y/X)P(X)}{P(Y)}, \text{ if } P(Y) \neq 0$$

Bayesian network:

A Bayesian Network falls under the classification of Probabilistic Graphical Modelling (PGM) procedure that is utilized to compute uncertainties by utilizing the probability concept. Generally known as **Belief Networks, Bayesian Networks** are used to show uncertainties using **Directed Acyclic Graphs** (DAG)

A **Directed Acyclic Graph** is used to show a Bayesian Network, and like some other statistical graph, a DAG consists of a set of nodes and links, where the links signify the connection between the nodes.



The nodes here represent random variables, and the edges define the relationship between these variables.

A DAG models the uncertainty of an event taking place based on the Conditional Probability Distribution (CDP) of each random variable. A **Conditional Probability Table** (CPT) is used to represent the CPD of each variable in a network

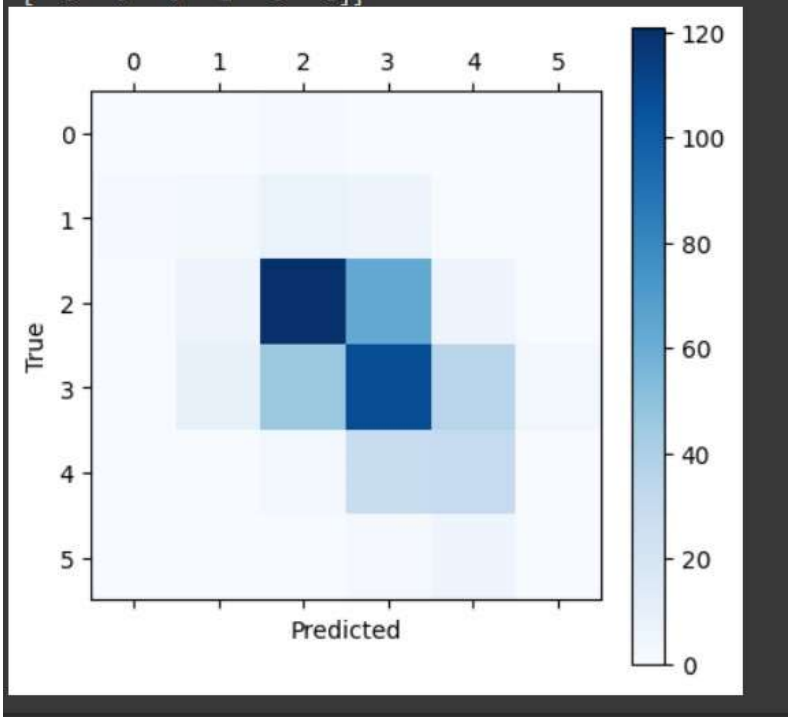
Code:

```
import pandas as pd from sklearn.model_selection
import train_test_split from sklearn.naive_bayes
import GaussianNB
from sklearn.metrics import accuracy_score, confusion_matrix import
matplotlib.pyplot as plt

# Load the dataset
data = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-
databases/winequality/winequality-red.csv', delimiter=';')

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(data.drop('quality', axis=1), data['qua
lity'], test_size=0.3, random_state=42)

# Create a Gaussian Naive Bayes classifier
clf = GaussianNB()

# Train the classifier on the training data
clf.fit(X_train, y_train)

# Test the classifier on the testing data
y_pred = clf.predict(X_test)

# Evaluate the accuracy of the classifier
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy:', accuracy)
```

```
# Print the confusion matrix cm =
confusion_matrix(y_test, y_pred)
print('Confusion Matrix:') print(cm)

# Plot the confusion matrix plt.matshow(cm,
cmap=plt.cm.Blues)
plt.colorbar() plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```

Output:

```
Accuracy: 0.5416666666666666
Confusion Matrix:
[[  0   0   1   0   0   0]
 [  1   2   8   6   0   0]
 [  0   6 121  63   5   0]
 [  0   9  46 107  35   3]
 [  0   0   3  28  30   0]
 [  0   0   0   1   5   0]]
```



Conclusion: We successfully implemented Bayesian Classification.

Department of Computer Engineering
Academic Year 2022-2023

Breast Cancer Prediction

Machine Learning Laboratory

By

Preet Gada	**60004200101**
Jiya Patel	**60004200105**
Ayush Jain	**60004200132**

Guide:

Prof. (Dr.) Ruhina Karani

Assistant Professor

Breast Cancer Prediction

1. PROBLEM STATEMENT

The problem statement of this report is to develop and compare the performance of three machine learning algorithms, namely SVM, KNN, and Naive Bayes, for the accurate prediction of breast cancer. The study aims to train the models on a dataset of patient information, including clinical and imaging data, and evaluate their ability to classify breast tumors as either benign or malignant. The key objective is to identify the most effective algorithm for breast cancer prediction, which can assist healthcare professionals in the early detection and diagnosis of breast cancer, leading to improved patient outcomes and survival rates.

2. INTRODUCTION

a. Need

The need for breast cancer prediction using machine learning is driven by the desire to improve early detection and diagnosis of breast cancer, which is one of the most common cancers affecting women worldwide. Machine learning algorithms offer the potential to accurately classify breast tumors as either benign or malignant based on their characteristics, such as size, shape, and texture. This can help healthcare professionals to identify cases of breast cancer at an early stage, when treatment is more effective and outcomes are better.

Moreover, the existing methods of breast cancer prediction rely on the subjective interpretation of mammograms by radiologists, which can result in a high rate of false positives or false negatives. Machine learning algorithms, on the other hand, can provide a more objective and consistent approach to breast cancer prediction by analyzing large amounts of data and identifying patterns and relationships that are not easily discernible by human experts.

Therefore, the use of machine learning algorithms for breast cancer prediction has the potential to improve the accuracy and efficiency of breast cancer diagnosis, leading to better patient outcomes and reduced healthcare costs.

b. Working

The breast cancer prediction model using machine learning algorithms SVM, KNN, and Naive Bayes works by analyzing a dataset of patient information and training the algorithms to predict whether a breast tumor is benign or malignant based on the tumor's characteristics.

The dataset used for this model is the Wisconsin Diagnostic Breast Cancer (WDBC) dataset, which contains 569 instances of breast tumor data. Each instance includes information on various features, such as radius, texture, and symmetry, as well as a diagnosis label indicating whether the tumor is benign (B) or malignant (M).

To develop the breast cancer prediction model, the dataset is split into training and testing sets. The algorithms are then trained on the training set using the features as input and the diagnosis label as the output. Once trained, the algorithms are tested on the testing set to evaluate their accuracy in predicting the diagnosis of new, unseen cases.

SVM, KNN, and Naive Bayes are all supervised machine learning algorithms. SVM is a linear classification algorithm that separates data points into different classes using a hyperplane. KNN is a non-parametric algorithm that assigns a new data point to the class with the majority of its k nearest neighbors. Naive Bayes is a probabilistic algorithm that calculates the probability of a data point belonging to each class and assigns the class with the highest probability.

After training and testing the algorithms on the WDBC dataset, their performance is compared based on various evaluation metrics such as accuracy, precision, recall, and F1 score. The algorithm with the highest performance is selected as the best model for predicting breast cancer.

Overall, the breast cancer prediction model using machine learning algorithms SVM, KNN, and Naive Bayes provides an objective and accurate approach to breast cancer diagnosis that can assist healthcare professionals in making informed decisions about patient care.

c. Applications

The application of a breast cancer prediction model in healthcare can be significant and widespread. Here are some examples:

- Early detection: A breast cancer prediction model can assist healthcare professionals in detecting breast cancer at an early stage, leading to better treatment options and improved patient outcomes.
- Risk assessment: The model can help in assessing the risk of developing breast cancer in women who have a family history of the disease or other risk factors.

- Personalized treatment: By predicting the likelihood of a breast tumor being malignant, the model can assist healthcare professionals in developing personalized treatment plans for individual patients.
- Clinical decision making: The model can provide additional information to support clinical decision-making, such as whether a biopsy is necessary or if further diagnostic tests are required.
- Resource allocation: The model can help in allocating healthcare resources by identifying patients who require urgent care and prioritizing their treatment.

Overall, the breast cancer prediction model can improve the accuracy and efficiency of breast cancer diagnosis, leading to better patient outcomes and reduced healthcare costs. It can also assist healthcare professionals in making informed decisions about patient care and resource allocation.

3. ALGORITHMS USED

SUPPORT VECTOR MACHINE (SVM)

Support Vector Machine (SVM) is a popular machine learning algorithm used for classification and regression analysis. SVM tries to find the best decision boundary that can separate the data into two classes such that the margin between the two classes is maximized. Here is how SVM works:

Data Preparation: First, the data is prepared by splitting it into training and testing sets. Then, the features are normalized or standardized to ensure that they have a similar scale and range.

Model Training: Next, SVM finds the best decision boundary that can separate the data into two classes. SVM tries to find a hyperplane that maximizes the margin between the two classes. The margin is the distance between the hyperplane and the nearest data points of each class. SVM finds the hyperplane that maximizes this margin, making it the best decision boundary.

Classification: Once the hyperplane is found, SVM uses it to classify new, unseen data points. SVM assigns a new data point to one of the two classes based on which side of the hyperplane it falls.

SVM has several advantages over other machine learning algorithms. One of its strengths is that it can handle high-dimensional data and can work well with small to medium-sized datasets. SVM can also handle noisy or overlapping data, which is often the case in medical datasets. Moreover, SVM is less prone to overfitting and can generalize well to new data, making it suitable for developing accurate and reliable machine learning models.

Applications of SVM:

Image Classification: SVM is used to classify images based on their features, such as color, texture, and shape.

Text Classification: SVM is used to classify text data, such as email spam detection, sentiment analysis, and document classification.

Bioinformatics: SVM is used to analyze biological data, such as gene expression data and protein structure data.

Finance: SVM is used for credit scoring, stock price forecasting, and fraud detection.

Healthcare: SVM is used for medical diagnosis, such as breast cancer detection and diagnosis of other diseases.

Overall, SVM is a powerful and versatile machine learning algorithm that has many applications in various fields.

K-NEAREST NEIGHBOURS (KNN)

K-Nearest Neighbors (KNN) is a popular machine learning algorithm used for classification and regression analysis. KNN classifies new data points based on their proximity to the closest data points in the training set. Here is how KNN works:

Data Preparation: First, the data is prepared by splitting it into training and testing sets. Then, the features are normalized or standardized to ensure that they have a similar scale and range.

Choosing K: Next, a value of K is chosen, which is the number of nearest neighbors to consider when classifying a new data point. This value is typically chosen by trial and error or cross-validation.

Finding Neighbors: Once K is chosen, KNN finds the K nearest neighbors of a new data point in the training set based on their distance. The distance can be measured using various distance metrics, such as Euclidean distance or Manhattan distance.

Classification: Finally, KNN classifies the new data point based on the majority class of its K nearest neighbors. If most of the neighbors belong to class A, the new data point is classified as class A, and vice versa.

KNN has several advantages over other machine learning algorithms. One of its strengths is that it is simple and easy to implement. KNN can also handle non-linear and complex decision boundaries, making it suitable for a wide range of applications. Moreover, KNN does not require a training phase, making it a suitable choice for real-time and streaming applications.

Applications of KNN:

Image Classification: KNN is used to classify images based on their features, such as color, texture, and shape.

Recommendation Systems: KNN is used to recommend products or services to users based on their similarity to other users.

Anomaly Detection: KNN is used to detect anomalies in data, such as fraudulent transactions or network intrusions.

Healthcare: KNN is used for medical diagnosis, such as predicting the risk of developing certain diseases or classifying medical images.

Social Network Analysis: KNN is used to find similar users or groups in social networks based on their profiles or behavior.

Overall, KNN is a simple and effective machine learning algorithm that has many applications in various fields.

NAIVE BAYES CLASSIFICATION

Naive Bayes is a popular machine learning algorithm used for classification tasks. Naive Bayes is based on Bayes' theorem, which states that the probability of a hypothesis (class) is updated as new evidence (features) is observed. Naive Bayes assumes that all features are independent of each other, which is a naive assumption but often works well in practice. Here is how Naive Bayes works:

Data Preparation: First, the data is prepared by splitting it into training and testing sets. Then, the features are normalized or standardized to ensure that they have a similar scale and range.

Model Training: Next, Naive Bayes calculates the probability of each class based on the frequency of each feature in the training set. Naive Bayes calculates the conditional probability of each feature given each class and the prior probability of each class.

Classification: Once the probabilities are calculated, Naive Bayes classifies new data points based on the highest probability. Naive Bayes calculates the posterior probability of each class given the features of the new data point and chooses the class with the highest probability.

Naive Bayes has several advantages over other machine learning algorithms. One of its strengths is that it is fast and scalable, making it suitable for large datasets. Naive Bayes can also handle high-dimensional data and can work well with small to medium-sized datasets. Moreover, Naive

Bayes is less prone to overfitting and can generalize well to new data, making it suitable for developing accurate and reliable machine learning models.

Applications of Naive Bayes:

Text Classification: Naive Bayes is used to classify text data, such as email spam detection, sentiment analysis, and document classification.

Recommendation Systems: Naive Bayes is used to recommend products or services to users based on their preferences.

Healthcare: Naive Bayes is used for medical diagnosis, such as predicting the risk of developing certain diseases based on patient data.

Finance: Naive Bayes is used for credit scoring and fraud detection.

Image Classification: Naive Bayes is used to classify images based on their features, such as color, texture, and shape.

Overall, Naive Bayes is a simple yet effective machine learning algorithm that has many applications in various fields.

4. IMPLEMENTATION

a. Code of important functions

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report
from yellowbrick.classifier import ClassificationReport
cancer = datasets.load_breast_cancer()
cancer.keys()
print(cancer['DESCR'])
print(cancer['target_names'])
print(cancer['feature_names'])
```

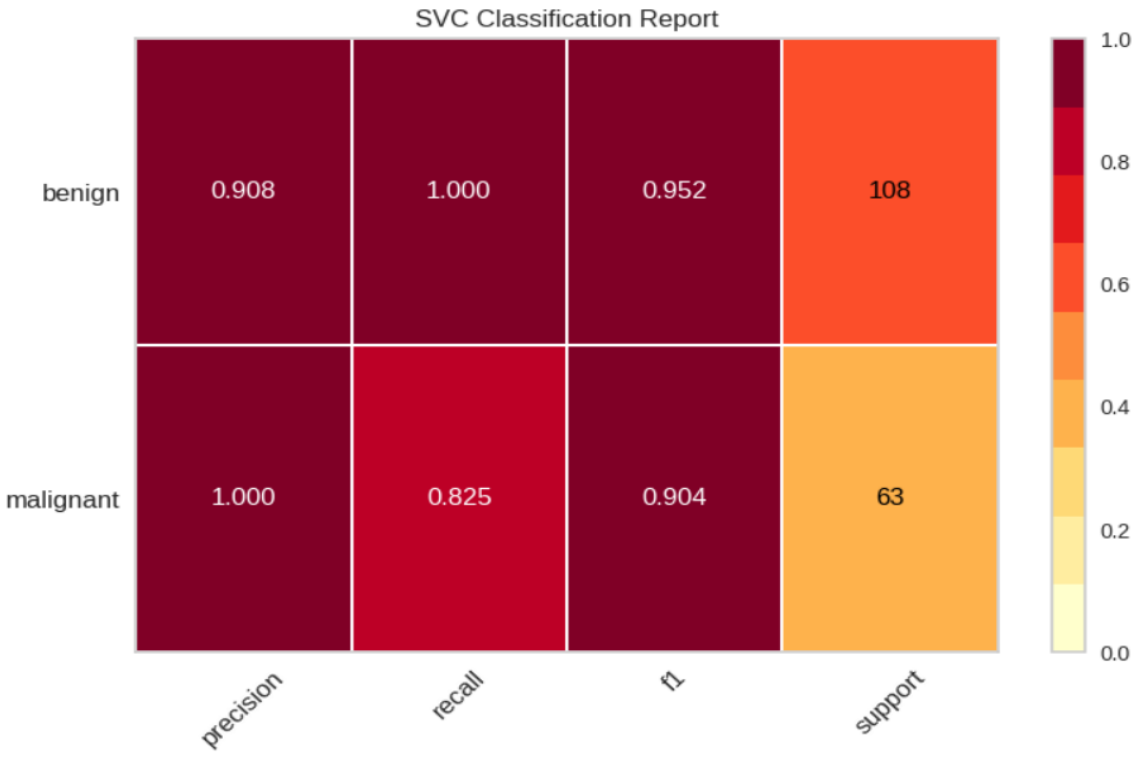
```
      cancer['data'].shape
      df_cancer = pd.DataFrame(np.c_[cancer['data'], cancer['target']],
columns= np.append(cancer['feature_names'],['target']))
      df_cancer.head()
      sns.scatterplot(x='mean area',y='mean smoothness',hue='target',data
=df_cancer)
      X = cancer.data
      y = cancer.target
      # Split the data into training and testing sets
      X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)
      # Train an SVM classifier with a radial basis function (RBF) kernel
      clf = svm.SVC(kernel='rbf', gamma='scale',probability=True)
      clf.fit(X_train, y_train)
      # Make predictions on the testing set
      y_pred = clf.predict(X_test)
      # Generate a confusion matrix
      cm = confusion_matrix(y_test, y_pred)
      print('Confusion Matrix:')
      print(cm)
      sns.heatmap(cm,annot=True)
      # Generate a classification report
      svm_report = classification_report(y_test, y_pred)
      print('Classification Report:')
      print(svm_report)
      # Plot classification reports for each model
      visualizer = ClassificationReport(clf, classes=['malignant',
'benign'], support=True)
      visualizer.score(X_test, y_test)
      visualizer.poof()
      from sklearn.neighbors import KNeighborsClassifier
      from sklearn.model_selection import train_test_split
      from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report
      from sklearn.datasets import load_breast_cancer
      from sklearn.preprocessing import StandardScaler
      # Load the breast cancer dataset
      data = load_breast_cancer()
      # Split the dataset into training and test sets
```

```
      X_train, X_test, y_train, y_test = train_test_split(data.data,
data.target, test_size=0.3, random_state=42)
      # Scale the data
      scaler = StandardScaler()
      X_train = scaler.fit_transform(X_train)
      X_test = scaler.transform(X_test)
      # Create a KNN classifier and train it on the training data
      knn = KNeighborsClassifier(n_neighbors=5)
      knn.fit(X_train, y_train)
      # Use the trained classifier to make predictions on the test data
      y_pred = knn.predict(X_test)
      cm = confusion_matrix(y_test, y_pred)
      print("Confusion matrix:\n", cm)
      sns.heatmap(cm,annot=True)
      knn_report = classification_report(y_test, y_pred)
      print("Classification report:\n", knn_report)
      visualizer = ClassificationReport(knn, classes=['malignant',
'benign'], support=True)
      visualizer.score(X_test, y_test)
      visualizer.poof()
      from sklearn.datasets import load_breast_cancer
      from sklearn.naive_bayes import GaussianNB
      from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, roc_curve, auc
      import matplotlib.pyplot as plt
      import numpy as np
      from sklearn.model_selection import train_test_split

      # Load the breast cancer dataset
      data = load_breast_cancer()
      X = data.data
      y = data.target
      # Split the dataset into training and test sets
      X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
      # Train a Naive Bayes classifier
      nb = GaussianNB()
      nb.fit(X_train, y_train) # Fit the classifier on the training data
      # Make predictions on the test set
      y_pred = nb.predict(X_test)
```

```
        cm = confusion_matrix(y_test, y_pred)
        print("Confusion matrix:\n", cm)
        sns.heatmap(cm,annot=True)
        nb_report = classification_report(y_test, y_pred)
        print("Classification report:\n", nb_report)
        visualizer = ClassificationReport(nb, classes=['malignant',
'benign'], support=True)
        visualizer.score(X_test, y_test)
        visualizer.poof()
        # Print the performance analysis
        print("Performance Analysis:\n")
        print("Algorithm\tClassification Report")
        print("-"*40)
        print(f"SVM{svm_report}")
        print(f"KNN{knn_report}")
        print(f"Naive Bayes{nb_report}")
```

b. Important screenshots

Confusion matrix using SVM



Performance Analysis:

Algorithm Classification Report

SVM

	precision	recall	f1-score	support
0	1.00	0.83	0.90	63
1	0.91	1.00	0.95	108
accuracy			0.94	171
macro avg	0.95	0.91	0.93	171
weighted avg	0.94	0.94	0.93	171

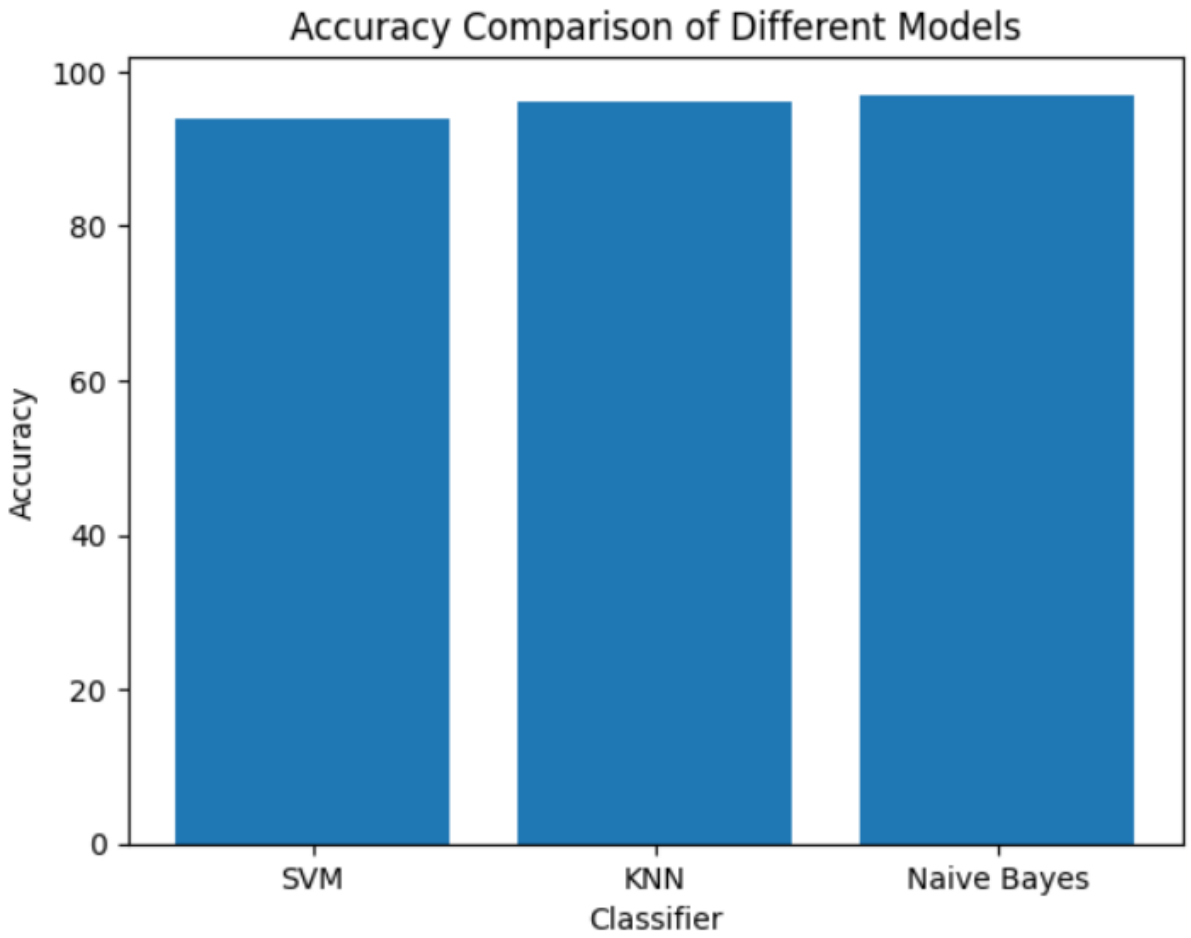
KNN

	precision	recall	f1-score	support
0	0.95	0.94	0.94	63
1	0.96	0.97	0.97	108
accuracy			0.96	171
macro avg	0.96	0.95	0.96	171
weighted avg	0.96	0.96	0.96	171

Naive Bayes

	precision	recall	f1-score	support
0	1.00	0.93	0.96	43
1	0.96	1.00	0.98	71
accuracy			0.97	114
macro avg	0.98	0.97	0.97	114
weighted avg	0.97	0.97	0.97	114

5. PERFORMANCE ANALYSIS



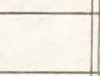
6. CONCLUSION

Based on the results of the comparison analysis using SVM, KNN, and Naive Bayes algorithms on the breast cancer dataset, it can be concluded that all three models performed well with high accuracy.The Naive Bayes model achieved the highest accuracy of 97%, followed by the KNN model with an accuracy of 96%, and the SVM model with an accuracy of 94%.When considering other metrics such as precision, recall, and F1-score, Naive Bayes again performed the best, followed by KNN and SVM.Therefore, it can be recommended that Naive Bayes is a suitable algorithm for breast cancer classification. However, the other models may still be useful depending on the specific requirements of the problem at hand.

Name : Ayush Jain
SAP ID: 60004200132
Div : B
Computer Engineering

ML - Assignment 1

Q.1) Consider the set of training data below, and two clustering algorithms : K Means and a Gaussian Matrix Model (GMM) trained using EM. Will these two clustering algorithms produce the same cluster centres for this data set? If yes, Justify the answer.



→ 1) Both algorithms - K-means and GMM are capable of identifying the cluster effectively. However, the key distinction between them is the assignment method used for each point.

2) In K-Means, a hard assignment approach is utilized, where each point is assigned exclusively to one cluster. On the other hand, GMM employs a soft assignment technique, where every point has a non-zero probability of belonging to each cluster.

3) As a result, the calculations of the means for each cluster differs between two algorithms. In K-means, the cluster means are determined by averaging the points assigned to that specific cluster. In contrast, GMM calculates the means of each cluster based on differently weighted averages of all points.

4) This discrepancy leads to a noticeable effect, where the centre of the left cluster is skewed to the right, and the centre of the right cluster is skewed to the left.

5) Whether you appriciate or disapprove of this characteristic, it is crucial to recognise its existence and comprehend its origin.

FOR EDUCATIONAL USE

Scanned with CamScanner

Q. 2) Describe any five real time applications where Hidden markov model can be used?

→ 1) Speech Recognition: HMMs are widely used in speech recognition system to model the time-varying characteristics of spoken language. They can be used to identify and predict the sequence of phonemes or words in a given audio signal, enabling the conversion of speech into text.

2) Bioinformatics: HMMs are used to model and predict the secondary structure of proteins or the functional elements in DNA sequences. They help in identifying genes, predicting protein folds and detting homologus sequences.

3) Finance: HMM's can be applied to financial time series data, such as stock prices or currency exchanging rates, to model and predict trends or hidden states.

4) Gesture Recognition: HMM's can model the temporal dynamics of human gestures, making them suitable for gesture recognition in real-life applications.

5) Natural Language Processing: HMM's are used in NLP's tasks like parts-of-speech tagging, which involves assigning a grammatical category to each word in a sentence. Real-time applications include chatbots, machine translation, and sentiment analysis, where understanding the structure and meaning of text is crucial for accurate and timely responses.

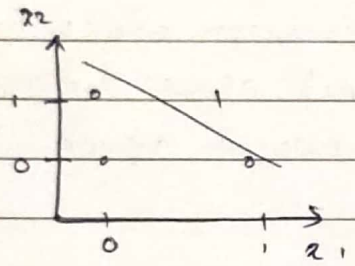
Sundaram
FOR EDUCATIONAL USE

Scanned with CamScanner

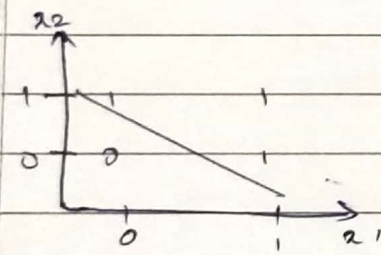
Q. 3 Apply RBF on two input XOR gate and show how it converts non linearly seperable problem to linearly seperable by applying Gaussian kernel and increasing the dimensions to four.

→ Radial Basis Function Network (RBFN)

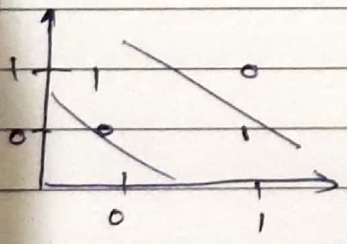
Consider a linearly seperable examples of AND and OR.



This is linearly seperable since a single line can seperate both clases 0 and 1



Linearly seperable since a straight line can seperate both classes.



Non-linearly seperable data since we can't seperate both the classes using a single straight line.

- RBFN performs linear formation over the i/p vector before the i/p vectors are feed for classification.
- Using such non-linear formation, it is possible to convert a non-linearly seperable data into linearly seperable data.

FOR EDUCATIONAL USE

Scanned with CamScanner

- RBFN also increases the dimensionality of the i/p feature vectors to convert non-linearly seperable problem into linearly seperable problem.

- case I] Converting a non-linearly seperable problem into linearly seperable problem by only applying formation function.

- RBFN also increase the dimensionality of a i/p feature vectors since it is found that if we increase the dimensionality of i/p vectors, a problem which is non-linear seperable in lower dimensional space becomes linearly seperable in higher dimensional space.

Sundaram
FOR EDUCATIONAL USE

Scanned with CamScanner

Name: Ayush Jain
SAP ID: 60004200132
Div: B
Computer Science

ML - Assignment 2

Q. 1) Describe Model based learning and temporal difference learning methods of reinforcement learning.

→ 1) Model based learning and temporal difference learning are two distinct methods used in reinforcement learning.

2) Model based learning is a method in which the agent learns to predict the dynamics of its environment, building an internal model of the environment's state transition and rewards. The internal model enables the agent to simulate possible future scenarios and plan actions to optimize the performance.

3) In model based learning, the agent uses it experience to update its internal model, which typically consist of two main components a) State transistion Model. b) Reward Model

4) Once the agent has an accurate internal Model, it can use planning algorithms like Monte-Carlo Tree search, value iteration or policy iteration.

5) Temporal difference learning is a model free method, which means that the agent does not learn an explicit model of the environments dynamics. Instead, it directly learns an optimal policy or value function by updating it estimates using difference between current and predicted future rewards, known as temporal difference error.

6) There are two primary TD learning algorithms: a) SARSA (State-Action-Reward State Action). b) Q-learning.

Sundaram
FOR EDUCATIONAL USE

Scanned with CamScanner

Q. 2) Explain how ML can be used for video surveillance. Also describe which technique of ML is most suitable for designing it.

→ 1) ML can be used for video surveillance to analyze and process video data and automating various tasks. By using ML techniques, it can perform tasks such as object recognition, motion detection, behaviour analysis and anomaly detection.

2) Some applications of ML in video surveillance:

(a) Object recognition and tracking: ML algo can be trained to identify and track objects, such as people, vehicle and animals. This allows the surveillance system to monitor specific objects of interest and track their movement across different cameras feeds.

(b) Crowd analysis: ML can be applied to analyze crowd behaviours, density and movement patterns, which can be useful for public safety, traffic control, or event management.

3) The most suitable ML tech for designing video surveillance depends on the specific task and req. However, deep learning tech, particularly CNN are used.

4) In summary, ML, especially deep learning techniques like CNN can significantly enhance video surveillance system by automating task, such as object recognition, motion detection and behaviour analysis.

5) The choice of ML technique depends on the specific application and requirements of the video surveillance system.

Sundaram
FOR EDUCATIONAL USE

Scanned with CamScanner