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: Kartik Jolapara SAP ID: 60004200107

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

Year: **T.Y. B.Tech.** Sem: **VI** Batch: B1

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	4	4	5	5	4	5	4	4					5	5		
2. Describe (Factual/Conceptual/Procedural/ Metacognitive)	4	4	4	4	5	5	4	5	4					4	4		
3. Demonstration (Factual/Conceptual/Procedural/ Metacognitive)	4	4	5	5	4	5	5	5	4					5	5		
4. Strategy (Analyse & / or Evaluate) (Factual/Conceptual/ Procedural/Metacognitive)	4	4	4	4	4	5	4	4	4					4	5		
5. Interpret/ Develop (Factual/Conceptual/ Procedural/Metacognitive)	–	–	–	–	–	–	–	–	–					4	4		
6. Attitude towards learning (receiving, attending, responding, valuing, organizing, characterization by value)	4	4	4	5	5	5	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)	–	–	–	–	–	–	–	–	5					–	–		
Total	20	20	21	23	23	24	22	22	21					22	23		
Signature of the faculty member	k	k	k	k	k	k	k	k	k					k	k		

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: Kartik

Signature of the Faculty member: Ruhinale 04/05/23
Name of the Faculty member:

Signature of Head of the Department
Date:

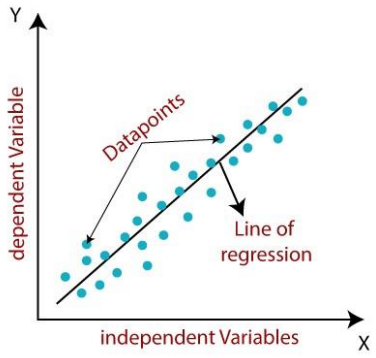
PLOT NO. U-15, JVPD SCHEME, MUKTI PATH, SWAMI VIVEK VILE PARLE (WEST), MUMBAI - 400056
Tel : 42335000/42335001 Email : info@djsce.ac.in / admin@djsce.ac.in Website : www.djsce.ac.in

Machine Learning
Experiment 1

SAP ID: 60004200107 Name: Kartik Jolapara
Division: B Batch: B1

AIM
To implement Linear Regression.

THEORY
Linear regression is one of the easiest and most popular supervised Machine Learning algorithms. It is a statistical method that is used for predictive analysis. Linear regression makes predictions for continuous/real or numeric variables such as sales, salary, age, product price, etc.
Linear regression algorithm shows a linear relationship between a dependent (y) and one or more independent (y) variables, hence called as linear regression. Since linear regression shows the linear relationship, which means it finds how the value of the dependent variable is changing according to the value of the independent variable.
The linear regression model provides a sloped straight line representing the relationship between the variables.



Mathematically, we can represent a linear regression as - y
= w0 + w1 * x
Here, y = Dependent Variable (Target Variable) x = Independent Variable (Predictor Variable) w0 = Intercept of the line (Gives an additional degree of freedom) w1 = Linear regression coefficient (Scale factor to each input value).

Linear regression can be further divided into two types –

1. Simple Linear Regression

If a single independent variable is used to predict the value of a numerical dependent variable, then such a Linear Regression algorithm is called Simple Linear Regression.

2. Multiple Linear regression

If more than one independent variable is used to predict the value of a numerical dependent variable, then such a Linear Regression algorithm is called Multiple Linear Regression.

CODE

Statistical Linear Regression

```
import numpy as np import
matplotlib.pyplot as plt

def estimate_coeff(x, y):
n = np.size(x)
mean_x = np.mean(x)
mean_y = np.mean(y)
   SS_xy = np.sum(y * x) - n * mean_y * mean_x
   SS_xx = np.sum(x * x) - n * mean_x * mean_x
   SS_yx2 = mean_y * np.sum(x * x) - mean_x * np.sum(x *
y)    SS_x = np.sum(x * x) - n * mean_x * mean_x    w_1 =
SS_xy / SS_xx    w_0 = SS_yx2 / SS_x    return (w_0, w_1)

def plot_regression_line(x, y, w):
   plt.scatter(x, y, color = "m", marker = "o", s =
30)    y_pred = w[0] + w[1] * x    plt.plot(x,
y_pred, color = "g")    plt.xlabel('x')
plt.ylabel('y')    plt.show()

def main():
   x = np.array([1, 2, 3, 4])    y = np.array([7, 3, 2, 4])    w =
estimate_coeff(x, y)    print("Estimated coefficients - \nw_0 = {}\nw_1 =
{}".format(w[0], w[1]))    print("The equation is : y = {} +
{}x\n".format(w[0], w[1]))    plot_regression_line(x, y, w)

if __name__ == "__main__":
   main()
```

Linear Regression using ML import

```
pandas as pd import numpy as np
import matplotlib.pyplot as plt data =
pd.read_csv('Salary_Data.csv')

x = data.iloc[:, 0]
y = data.iloc[:, -1]
```

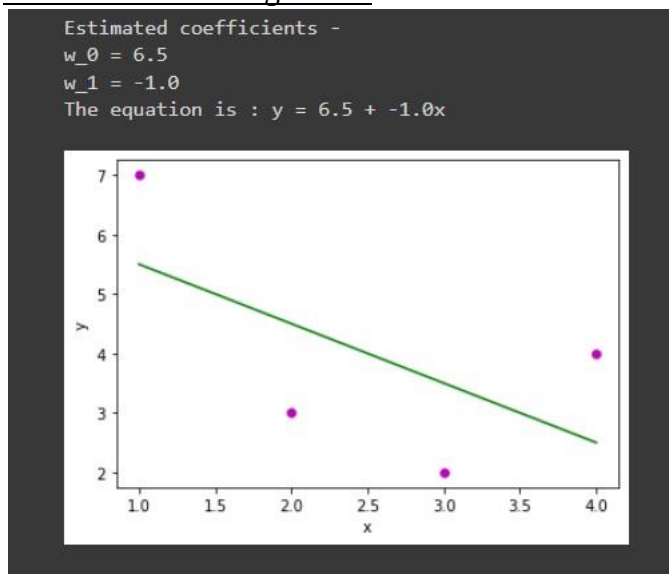
```
w_0 = 0.1
w_1 = 0.2
alpha = 0.01
epoch = 0
sumofdiff = 0
n = len(x)

for epoch in range (1000):
y1 = []   for i in x:
   y1.append(w_0 + w_1 * i)
for i in range(n):
   sumofdiff = sumofdiff + (y1[i] - y[i]) *
x[i]   delta = alpha * sumofdiff / n   w_0 =
w_0 - delta   w_1 = w_1 - delta   y = y1
print('w_0 = ', w_0) print('w_1 = ', w_1)

plt.scatter(x, y, color = "m", marker = "o", s =
30) plt.plot(x, y, color = "g") plt.xlabel('Years
Experience') plt.ylabel('Salary')
```

OUTPUT

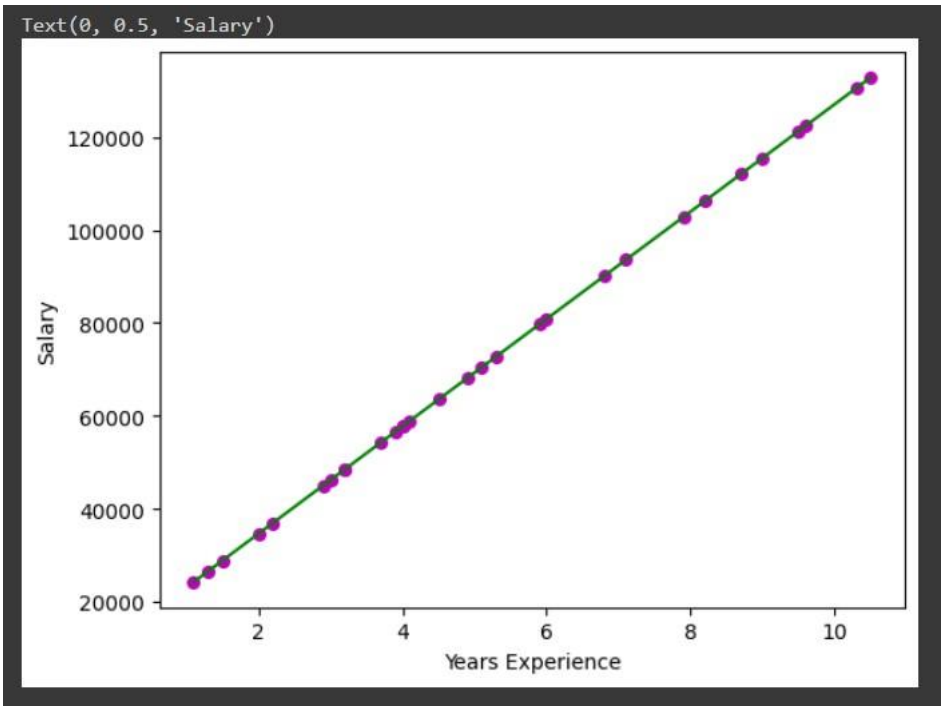
Statistical Linear Regression



Linear Regression using ML

```
w_0 =  11550.81292846198
w_1 =  11550.912928461981
```

```
Text(0, 0.5, 'Salary')
```



CONCLUSION

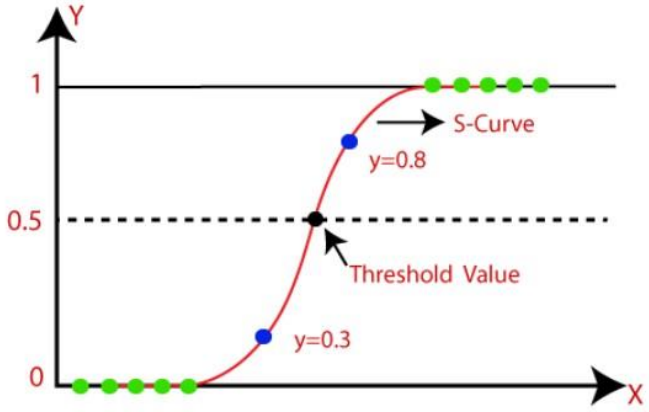
Thus, we have successfully implemented Linear Regression using both the statistical method and machine learning.

Machine Learning
Experiment 2

SAP ID: 60004200107 Name: Kartik Jolapara
Division: B Batch: B1

AIM
To implement Logistic Regression.

THEORY
Logistic regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique. It is used for predicting the categorical dependent variable using a given set of independent variables. It predicts the output of a categorical dependent variable. Therefore, the outcome must be a categorical or discrete value. It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, it gives the probabilistic values which lie between 0 and 1.
Logistic Regression is like the Linear Regression except that how they are used. Linear Regression is used for solving Regression problems, whereas Logistic regression is used for solving the classification problems. In Logistic regression, instead of fitting a regression line, we fit an "S" shaped logistic function, which predicts two maximum values (0 or 1). The curve from the logistic function indicates the likelihood of something such as whether the cells are cancerous or not, a mouse is obese or not based on its weight, etc.
It is a significant machine learning algorithm because it can provide probabilities and classify new data using continuous and discrete datasets. It can be used to classify the observations using different types of data and can easily determine the most effective variables used for the classification. The below image is showing the logistic function –



CODE import
pandas as pd import

```
math as m import
random as r import
numpy as np
import matplotlib.pyplot as plt

df = pd.read_csv('pima-indians-diabetes.csv')

x = df.iloc[ : , 0]
y = df.iloc[ : , -
1]

w = 1 alpha
= 0.01

epoch = 0 while
epoch <= 1000:
  sigmoidal_func = list(map(lambda x1 : (1/(1 + m.exp((-1) * (x1 * w)))), x))
sumation = list(map(lambda y1, y2, x1 : (y1 - y2) * x1, y, sigmoidal_func,
x))   total = sum(sumation)   gradient =  alpha * total   w += gradient   epoch
+= 1
print(w)

y_pred = list(map(lambda x1 : 1 if (1/(1 + m.exp((-1) * (x1 * w)))) > 0.5 else 0, x))
print(y_pred)
```

OUTPUT

```
11.181520306997749
[1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
```

CONCLUSION

Thus, we have successfully implemented Logistic Regression.

Machine Learning
Experiment 3

SAP ID: 60004200107 Name: Kartik Jolapara
Division: B Batch: B1

AIM
To implement CART decision tree algorithm.

THEORY
CART (Classification and Regression Tree) is a variation of the decision tree algorithm. It can handle both classification and regression tasks. It is a predictive algorithm used in Machine learning and it explains how the target variable's values can be predicted based on other matters. It is a decision tree where each fork is split into a predictor variable and each node has a prediction for the target variable at the end.
In the decision tree, nodes are split into sub-nodes based on a threshold value of an attribute. The root node is taken as the training set and is split into two by considering the best attribute and threshold value. Further, the subsets are also split using the same logic. This continues till the last pure sub-set is found in the tree or the maximum number of leaves possible in that growing tree.
CART algorithm uses Gini Impurity to split the dataset into a decision tree .It does that by searching for the best homogeneity for the sub nodes, with the help of the Gini index criterion.

Gini index/Gini impurity
The Gini index is a metric for the classification tasks in CART. It stores the sum of squared probabilities of each class. It computes the degree of probability of a specific variable that is wrongly being classified when chosen randomly and a variation of the Gini coefficient. It works on categorical variables, provides outcomes either "successful" or "failure" and hence conducts binary splitting only. The degree of the Gini index varies from 0 to 1.

$$Gini = 1 - \sum_{i=1}^{n} (pi)^2$$

where pi is the probability of an object being classified to a particular class.

Advantages of CART
- Results are simplistic.
- Classification and regression trees implicitly perform feature selection.
- Outliers have no meaningful effect on CART.

Disadvantages of CART
- Overfitting.
- High Variance.

- The tree structure may be unstable.

CODE CART

```
import pandas as pd
import numpy as np

def variable_count(att):   types =
pd.unique(att)   no_of_types =
len(types)   counts =
att.value_counts()   return
no_of_types, counts, types

def gini_of_attribute(no_of_types, counts, rows, cla, types, att1, cl):
  gini_a = 0
type_cl_count = 0
type_count = 0
  gini = []
div_index = 0

  if no_of_types == 2:     for i in
range(len(types)):       temp =
df.loc[df[att1.name] == types[i]]
type_count = len(temp)
     p = 1       for j in
range(len(cla)):
       temp = df.loc[(df[att1.name] == types[i]) & (df[cl.name] ==
cla[j])]        type_cl_count = len(temp)         p -=
pow((type_cl_count/type_count), 2)       gini_a += (type_count/rows)
* p

  elif no_of_types > 2:     for i
in range(no_of_types):
     temp1 = df.loc[df[att1.name] ==
types[i]]       temp2 = df.loc[df[att1.name] !=
types[i]]       type_count1 = len(temp1)
type_count2 = len(temp2)       p1 = 1
     p2 = 1       for j in
range(len(cla)):
       temp3 = df.loc[(df[att1.name] == types[i]) & (df[cl.name] ==
cla[j])]        type_cl_count1 = len(temp3)         p1 -=
pow((type_cl_count1/type_count1), 2)         temp4 =
df.loc[(df[att1.name] != types[i]) & (df[cl.name] == cla[j])]
type_cl_count2 = len(temp4)         p2 -=
pow((type_cl_count2/type_count2), 2)
```

```
gini.append((type_count1/rows) * p1 + (type_count2/rows) * p2)
gini_a = min(gini)     div_index = gini.index(gini_a)   return gini_a,
div_index

df = pd.read_csv('CART.csv') col
= list(df.columns.values.tolist())

cl = df.iloc[ : , -1]
no_of_types, counts, cla = variable_count(cl) rows = len(cl)
gini = 1 - pow((counts[0]/rows), 2) - pow((counts[1]/rows),
2) print(gini)

gini_a = [] div = [] t = []
att = len(df.columns) - 1

for i in range(att):
att1 = df.iloc[ : , i]
  no_of_types, counts, types = variable_count(att1)   t.append(types)   gini_a1,
div_index =  gini_of_attribute(no_of_types, counts, rows, cla, types, att1, cl)
gini_a.append(gini_a1)   div.append(div_index)

print(gini_a)

delta_gini = list(map(lambda item : gini - item, gini_a))

print(delta_gini)

index = delta_gini.index(max(delta_gini)) print("\n") print(col[index], "is the root variable
and the variable on one side is ",t[index][div[index]])
```

CART using in-built function

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

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

df['Age']=df['Age'].apply(lambda x: 1 if x =='youth' else (2 if x == 'middle' else 3))
df['Income']=df['Income'].apply(lambda x: 1 if x =='low' else (2 if x == 'medium' else 3))
df['Student']=df['Student'].apply(lambda x: 1 if x=='no' else 2)
df['Credit_Rating']=df['Credit_Rating'].apply(lambda x: 1 if x=='fair' else 2)
df['Buys_Computer']=df['Buys_Computer'].apply(lambda x: 1 if x=='no' else 2)
```

```
X = df.iloc[ : , 0 : 3]
y = df.iloc[ : , -1]
# X_train, X_test, y_train, y_test = train_test_split(X, y)

clf = tree.DecisionTreeClassifier()
clf.fit(X, y)

tree.plot_tree(clf)
```

OUTPUT

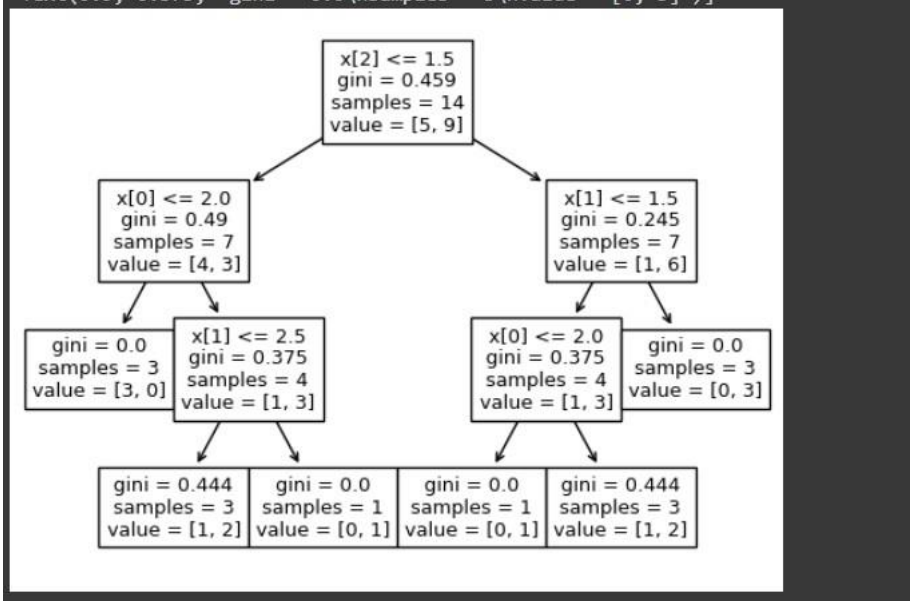
CART

```
0.4591836734693877
[0.35714285714285715, 0.44285714285714295, 0.3673469387755103, 0.42857142857142855]
[0.10204081632653056, 0.01632653061224476, 0.09183673469387743, 0.030612244897959162]


Age is the root variable and the variable on one side is  middle-aged
```

CART using in-built function

```
[Text(0.5, 0.875, 'x[2] <= 1.5\ngini = 0.459\nsamples = 14\nvalue = [5, 9]'),
 Text(0.2, 0.625, 'x[0] <= 2.0\ngini = 0.49\nsamples = 7\nvalue = [4, 3]'),
 Text(0.1, 0.375, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
 Text(0.3, 0.375, 'x[1] <= 2.5\ngini = 0.375\nsamples = 4\nvalue = [1, 3]'),
 Text(0.2, 0.125, 'gini = 0.444\nsamples = 3\nvalue = [1, 2]'),
 Text(0.4, 0.125, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.8, 0.625, 'x[1] <= 1.5\ngini = 0.245\nsamples = 7\nvalue = [1, 6]'),
 Text(0.7, 0.375, 'x[0] <= 2.0\ngini = 0.375\nsamples = 4\nvalue = [1, 3]'),
 Text(0.6, 0.125, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.8, 0.125, 'gini = 0.444\nsamples = 3\nvalue = [1, 2]'),
 Text(0.9, 0.375, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]')]
```



CONCLUSION

Thus, we have successfully implemented CART from scratch and using the in-built functions.

Machine Learning
Experiment 4

SAP ID: 60004200107 Name: Kartik Jolapara
Division: B Batch: B1

AIM
To implement PCA.

THEORY
Principal Component Analysis (PCA) is a statistical technique used to reduce the dimensionality of a large dataset. It is a commonly used method in machine learning, data science, and other fields that deal with large datasets. This method was introduced by Karl Pearson. It works on a condition that while the data in a higher dimensional space is mapped to data in a lower dimension space, the variance of the data in the lower dimensional space should be maximum.
PCA works by identifying patterns in the data and then creating new variables that capture as much of the variation in the data as possible. These new variables, known as principal components, are linear combinations of the original variables in the dataset. It reduces the dimensionality of a data set by finding this new set of variables, smaller than the original set of variables, retains most of the sample's information and useful for the compression and classification of data.
The PCA algorithm is based on some mathematical concepts such as -

- Variance and Covariance
- Eigenvalues and Eigen factors

PCA can be used for a variety of purposes, including data visualization, feature selection, and data compression. In data visualization, PCA can be used to plot high-dimensional data in two or three dimensions, making it easier to interpret. In feature selection, PCA can be used to identify the most important variables in a dataset. In data compression, PCA can be used to reduce the size of a dataset without losing important information.
Applications

- PCA is mainly used as the dimensionality reduction technique in various AI applications such as computer vision, image compression, etc.
- It can also be used for finding hidden patterns if data has high dimensions. Some fields where PCA is used are Finance, data mining, Psychology, etc.

CODE import pandas as pd
import numpy as np from
numpy.linalg import eig

```
df = pd.read_csv('Salary_Data.csv')

x = df.iloc[ : , 0]
y = df.iloc[ : , 1]

mew1 = round((sum(x)/len(x)), 2) mew2
= round((sum(y)/len(y)), 2)

mean = [mew1, mew2]

print(mean)

x_new = list(map(lambda x1 : round((x1 - mew1), 2), x))
y_new = list(map(lambda y1 : round((y1 - mew2), 2), y))

cov = np.array([[0, 0], [0, 0]])

for i in range(0, len(x_new)):   temp =
np.array([[float(x_new[i])], [float(y_new[i])]])
temp1 = temp.transpose()   temp2 = temp@temp1
cov = cov + temp2

cov = cov/len(x_new)

eigenvalues, eigenvectors = eig(cov)

feat = eigenvectors[ : , 1]

new_values = []

for i in range(0 , len(x_new)):
  temp = np.array([[float(x_new[i])], [float(y_new[i])]])
temp1 = feat @ temp   new_values.append(temp1.tolist())
print(new_values)
```

OUTPUT

```
[5.31, 76003.0]
[[36660.000238359156], [29798.000253290018], [38272.00018958758], [32478.000168662715], [36112.00012977673], [19361.00014477929], [15853.000152639528],
```

CONCLUSION

Thus, we have successfully implemented PCA Algorithm.

Machine Learning
Experiment 5

SAP ID: 60004200107 Name: Kartik Jolapara
Division: B Batch: B1

AIM
To implement K-Nearest Neighbour.

THEORY
K-Nearest Neighbours (KNN) is one of the most basic yet essential classification algorithms in Machine Learning. It is based on supervised learning technique. It assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories. It stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.
K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems. It is a non-parametric algorithm, which means it does not make any assumption on underlying data. It is also called a lazy learner algorithm because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset. At the training phase, it just stores the dataset and when it gets new data, then it classifies that data into a category that is much like the new data.
Example - Suppose, we have an image of a creature that looks similar to cat and dog, but we want to know either it is a cat or dog. So, for this identification, we can use the KNN algorithm, as it works on a similarity measure. Our KNN model will find the similar features of the new data set to the cats and dogs images and based on the most similar features it will put it in either cat or dog category.

CODE
```
import math

interview = [70, 70, 30, 10] exam_rank = [70, 40,
40, 40] classes = ['not hired', 'hired', 'not hired', 'not
hired'] data = {
    'first' : [70, 70, 'not hired'],
    'second' : [70, 40, 'hired'],
    'third' : [30, 40, 'not hired'],
    'fourth' : [10, 40, 'not hired'],
}
# print(data)
```

```
x = 30
y = 70

distance1 = []
distance2 = [] response
= []

for i in range(len(interview)) :
  s = (interview[i] - x)**2 + (exam_rank[i] -
y)**2   s = math.sqrt(s)   # print(s)
distance1.append(s)   distance2.append(s)

# print('Euclidean Distance - ', distance1) distance2.sort()
# print(distance2)

for i in range(len(distance1)):
for j in range(len(distance2)):
if(distance1[i] == distance2[j]):
if j == 0:
        response.append(data['first'][2])
elif j == 1:
        response.append(data['second'][2])
elif j == 2:
        response.append(data['third'][2])
elif j == 3:
        response.append(data['fourth'][2])

# print(response)

k = 3 countNH = countH = 0
for i in range(k):   if
response[i] == 'not hired':
    countNH += 1
  else:
    countH += 1

# print(countNH, countH)

if countNH > countH:
  print('Type for {} and {} will be not hired.'.format(x, y))
else:
  print('Type for {} and {} will be hired.'.format(x, y))
```

OUTPUT

```
Type for 30 and 70 will be not hired.
```

CONCLUSION

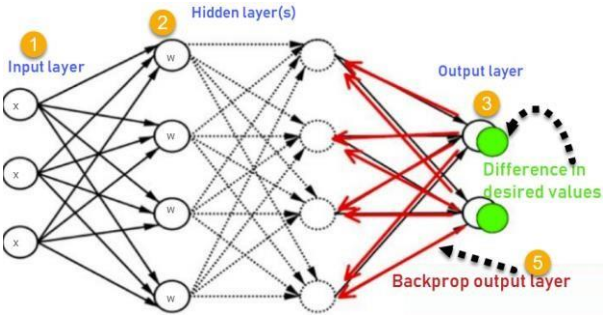
Hence, we have successfully implemented K-Nearest Neighbours in classification.

Machine Learning
Experiment 6

SAP ID: 60004200107 Name: Kartik Jolapara
Division: B Batch: B1

AIM
To implement Backpropagation.

THEORY
A neural network is a group of connected I/O units where each connection has a weight associated with its computer programs. It helps you to build predictive models from large databases. This model builds upon the human nervous system. It helps you to conduct image understanding, human learning, computer speech, etc.
Backpropagation is the essence of neural network training. It is the method of fine-tuning the weights of a neural network based on the error rate obtained in the previous epoch (i.e., iteration). Proper tuning of the weights allows you to reduce error rates and make the model reliable by increasing its generalization.
Backpropagation in neural network is a short form for "backward propagation of errors." It is a standard method of training artificial neural networks. This method helps calculate the gradient of a loss function with respect to all the weights in the network. The working -



1. Inputs X, arrive through the preconnected path
2. Input is modeled using real weights W. The weights are usually randomly selected.
3. Calculate the output for every neuron from the input layer, to the hidden layers, to the output layer.
4. Calculate the error in the outputs

   ErrorB = Actual Output – Desired Output
5. Travel back from the output layer to the hidden layer to adjust the weights such that the error is decreased.

Keep repeating the process until the desired output is achieved.

Advantages of Backpropagation

- It is fast, simple, and easy to program.
- It has no parameters to tune apart from the numbers of input.

- It is a flexible method as it does not require prior knowledge about the network.

CODE

```
import numpy as np

class NeuralNetwork:     def _init_(self, input_dim,
hidden_dim, output_dim):
     self.input_dim = input_dim
self.hidden_dim = hidden_dim
self.output_dim = output_dim

     # Initialize weights and biases       self.weights1 =
np.random.randn(self.input_dim, self.hidden_dim)       self.bias1 =
np.random.randn(self.hidden_dim)       self.weights2 =
np.random.randn(self.hidden_dim, self.output_dim)       self.bias2 =
np.random.randn(self.output_dim)

  def sigmoid(self, z):
     return 1/(1+np.exp(-z))

  def sigmoid_derivative(self, z):
     return z * (1 - z)

  def train(self, X, y, epochs):       for i in range(epochs):
# Forward propagation         z1 = np.dot(X,
self.weights1) + self.bias1         hidden_layer =
self.sigmoid(z1)         z2 = np.dot(hidden_layer,
self.weights2) + self.bias2         output_layer =
self.sigmoid(z2)

       # Backpropagation         output_error = y - output_layer
output_delta = output_error * self.sigmoid_derivative(output_layer)
hidden_error = np.dot(output_delta, self.weights2.T)         hidden_delta =
hidden_error * self.sigmoid_derivative(hidden_layer)

       # Update the weights and biases
       self.weights2 += np.dot(hidden_layer.T, output_delta)
self.bias2 += np.sum(output_delta, axis=0)
self.weights1 += np.dot(X.T, hidden_delta)         self.bias1
+= np.sum(hidden_delta, axis=0)

       # Print the loss every 100 epochs
if i % 100 == 0:
```

```
        loss = np.mean(np.square(y - output_layer))
print(f"Epoch {i}: Loss = {loss}")

  def predict(self, X):
     # Make a prediction for a new input       z1 =
np.dot(X, self.weights1) + self.bias1       hidden_layer =
self.sigmoid(z1)       z2 = np.dot(hidden_layer,
self.weights2) + self.bias2       output_layer =
self.sigmoid(z2)

     return output_layer

# Create a dataset
X = np.array([[0,0,1], [0,1,1], [1,0,1], [1,1,1]])
y = np.array([[0], [1], [1], [0]])

# Create a neural network with 3 input nodes, 4 hidden nodes, and 1 output node
nn = NeuralNetwork() nn._init_(3, 4, 1)

# Train the neural network for 1000 epochs nn.train(X,
y, 1000)

# Make a prediction for a new input new_input
= np.array([[1, 0, 0]])
print(nn.predict(new_input)) OUTPUT
```

```
Epoch 0: Loss = 0.3874651821847994
Epoch 100: Loss = 0.22308014757336586
Epoch 200: Loss = 0.06550820285020104
Epoch 300: Loss = 0.01803979072695123
Epoch 400: Loss = 0.008996793353942182
Epoch 500: Loss = 0.00574453349169824
Epoch 600: Loss = 0.004145272677354018
Epoch 700: Loss = 0.003212858570125505
Epoch 800: Loss = 0.002608610663163388
Epoch 900: Loss = 0.0021879221875389784
[[0.23915884]]
```

CONCLUSION

Hence, we have successfully implemented Backpropagation.

Machine Learning
Experiment 7

SAP ID: 60004200107 Name: Kartik Jolapara
Division: B Batch: B1

AIM
To implement SVM.

THEORY
Support Vector Machines (SVMs) are a type of supervised learning algorithm used for classification, regression, and outlier detection. SVMs are based on the idea of finding a hyperplane that separates the data points in a high-dimensional space with the maximum margin, where the margin is defined as the distance between the hyperplane and the closest data points from each class. SVMs are particularly effective for solving binary classification problems, where the goal is to separate the data into two classes, such as "spam" and "not spam".
The basic idea of SVMs is to transform the input data into a higher dimensional space, where it becomes more separable by a hyperplane. This transformation is done using a kernel function, which maps the input data to a higher dimensional feature space. The kernel function can be chosen based on the type of data and the problem at hand. Some common types of kernel functions include linear, polynomial, and radial basis function (RBF) kernels.
Once the data has been transformed, the SVM algorithm finds the hyperplane that separates the data points with the maximum margin. The hyperplane is defined as the set of all points x in the feature space that satisfy the equation - $w^T x + b = 0$
where w is a vector perpendicular to the hyperplane and b is the intercept. The distance between the hyperplane and the closest data points from each class is given by the margin, which is proportional to the length of the vector w.

The SVM algorithm aims to find the values of w and b that maximize the margin, subject to the constraint that all data points are classified correctly. This is done by solving a quadratic optimization problem, which involves finding the Lagrange multipliers that maximize the margin.
One of the main advantages of SVMs is their ability to handle non-linearly separable data by using kernel functions to transform the input data into a higher dimensional space. SVMs are also robust to overfitting and can generalize well to new data. However, SVMs can be sensitive to the choice of kernel function and its parameters, and the optimization problem can become computationally expensive for large datasets.

CODE AND OUTPUT

```
from sklearn.datasets import load_breast_cancer from
sklearn.model_selection import train_test_split from sklearn.svm
import SVC from sklearn.metrics import
```

```
confusion_matrix,classification_report import seaborn as sns
import matplotlib.pyplot as plt import pandas as pd

data =
pd.read_csv("UniversalBank.csv") X =
data.iloc[:, :-1] y = data.iloc[:, -1]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
svm_model = SVC(kernel='linear', C=1.0, random_state=0) svm_model.fit(X_train,
y_train)
```

```
SVC
SVC(kernel='linear', random_state=0)
```

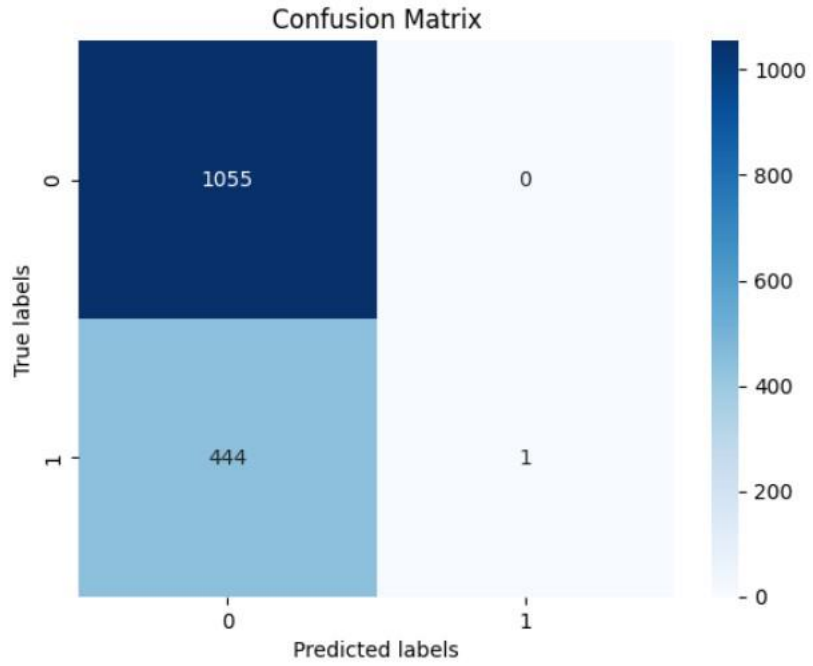
```
from sklearn.metrics import accuracy_score
y_pred = svm_model.predict(X_test) accuracy
= accuracy_score(y_test, y_pred)
print('Accuracy:', accuracy)
```

```
Accuracy: 0.704
```

```
cm = confusion_matrix(y_test, y_pred)
print(f"Confusion Matrix : ") print(cm)
```

```
Confusion Matrix :
[[1055    0]
 [ 444    1]]
```

```
sns.heatmap(cm, annot=True, cmap='Blues', fmt='g')
plt.xlabel('Predicted labels') plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()
```



from sklearn.metrics import confusion_matrix,classification_report print("Classification report")
print(classification_report(y_test,y_pred))

```
Classification report
              precision    recall  f1-score   support

           0       0.70      1.00      0.83      1055
           1       1.00      0.00      0.00       445

    accuracy                           0.70      1500
   macro avg       0.85      0.50      0.42      1500
weighted avg       0.79      0.70      0.58      1500
```

CONCLUSION

Hence, we have successfully implemented SVM.

Machine Learning
Experiment 8

SAP ID: 60004200107 Name: Kartik Jolapara
Division: B Batch: B1

`AIM
To implement Bayesian Classification.

THEORY
Bayesian classification is a probabilistic approach to machine learning that is used for classification tasks. It is based on Bayes' theorem, which describes the probability of an event occurring based on prior knowledge of conditions that might be related to the event. In Bayesian classification, the goal is to assign a given data point to one of several predefined classes. The classification is based on a probabilistic model that is learned from a training set of labeled data. The model assigns a probability to each class for a given data point, and the class with the highest probability is chosen as the predicted class for the data point.
The probabilistic model used in Bayesian classification is typically a Bayesian network, which is a graphical model that represents the dependencies between variables in a probabilistic way. Each node in the network represents a random variable, and the edges between nodes represent the conditional dependencies between variables. The network is learned from the training data using a maximum likelihood or maximum a posteriori estimation approach. Once the network is learned, Bayesian classification works by computing the posterior probability of each class for a given data point using Bayes' theorem:

$$P(C|X) = P(X|C) * P(C) / P(X)$$

where P(C|X) is the posterior probability of class C given the data point X, P(X|C) is the likelihood of the data point X given class C, P(C) is the prior probability of class C, and P(X) is the evidence, which is the probability of observing the data point X.

The likelihood of the data point X given class C is typically modeled using a probability density function, such as a Gaussian distribution. The prior probability of class C is the probability of observing class C in the training data. The evidence P(X) is a normalizing constant that ensures that the probabilities sum to 1 over all classes.
Once the posterior probabilities are computed for each class, the class with the highest probability is chosen as the predicted class for the data point.
Bayesian classification has several advantages over other classification methods, such as decision trees and neural networks. It is a probabilistic method, which means that it provides a measure of uncertainty in the classification results. It also handles missing data and noisy data well, and can be updated easily as new data becomes available. However, Bayesian classification can be computationally expensive, especially when dealing with highdimensional data, and it requires a significant amount of training data to learn an accurate model.

CODE AND OUTPUT import pandas as pd import numpy as np from sklearn.model_selection import train_test_split from sklearn.naive_bayes import GaussianNB from sklearn.metrics import confusion_matrix, roc_curve, roc_auc_score import matplotlib.pyplot as plt

df = pd.read_csv("BankLoan.csv") df.drop(df.columns[[0, 1, 2, 4, 5, 11]], axis=1, inplace=True) df = df.dropna()

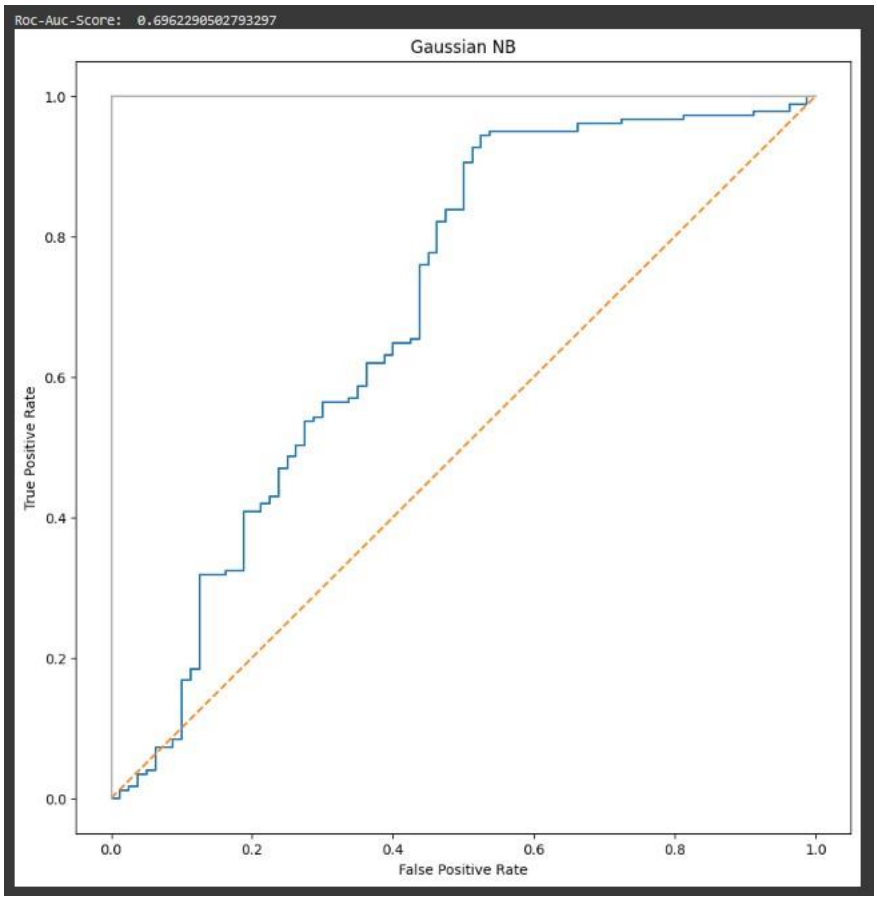
X = df.iloc[:, 1 : 6]
y = df.iloc[: , -1]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5)

gnb = GaussianNB() gnb.fit(X_train, y_train) y_pred = gnb.predict(X_test)
nb_loan = gnb.score(X_train, y_train) * 100

print("Bank Loan") print("Naive Baeyes Classifier") print(f"Accuracy - ", gnb.score(X_train, y_train)) print(f"Confusion Matrix : ")
print(confusion_matrix(y_test, y_pred))

```
Bank Loan
Naive Baeyes Classifier
Accuracy -  0.813953488372093
Confusion Matrix :
[[ 38  42]
 [ 13 166]]
```

y_score = gnb.predict_proba(X_test)[: , 1] false_pos, true_pos, threshold = roc_curve(y_test, y_score, pos_label = 'Y') print("\nRoc-Auc-Score: ", roc_auc_score(y_test, y_score)) plt.subplots(1, figsize = (10, 10)) plt.title("Gaussian NB") plt.plot(false_pos, true_pos) plt.plot([0, 1], ls = "--") plt.plot([0,0], [1,0], c="0.7"), plt.plot([1,1], c="0.7") plt.ylabel("True Positive Rate") plt.xlabel("False Positive Rate") plt.show()



CONCLUSION

Hence, we have successfully implemented Bayesian Classification.

Assignment - 1

6000420010 7

Q1
- Both algorithms k means and GMM are capable of identifying clusters effectively. However, key distinction is assignment method used for a point
- K means uses hard clustering assignment; point is assigned to one cluster.
- GMM applies soft clustering., each point has non zero probability of belonging to a cluster
- As a result calculation of means for each cluster differs for two.
- In K means, clusters are determined by all of the points assigned to specific cluster.
- GMM calculates cluster based on differently weighted average of all points. This discrepancy has notable effect.
- Center of left cluster is skewed to right and center of right cluster is skewed to left.
- Some may view this as drawback of EM algorithm but skewing is not entirely unjustified.

Q2

1. Speech recognition

- HMM are widely used in speech recognition systems to model the varying characteristics of spoken language..
- They can be use to identify and predict sequence of phonemes of words in given audio signal, enable conversion of speech to text.
- Real time speech recognition found in Siri, Alexa

2. Bio-informatics :

- HMM is used to model and predict secondary structure of proteins or functional elements in DNA sequence. They
- They help in detecting genes, protein folds and homologous sequences
- Real time application personalized medicine

3. Finance

- HMM can be applied to finance sectors dealing such as stock prices or currency exchange rates, to model and predict market trends or hidden states (eg: bullish bearish regimes)
- Real time trading algorithm

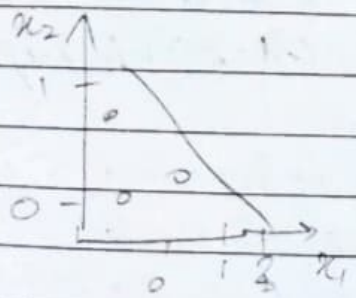
4. Gesture recognition.

- HMM can model temporal dynamics of human gestures, making them suitable for gesture

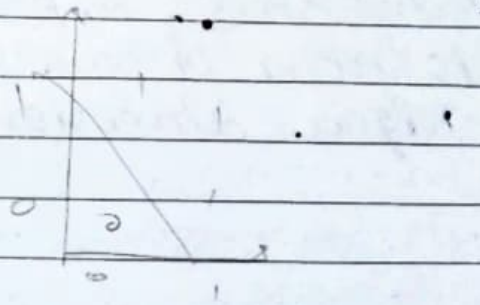
recognition in real time applications.
- Eg. : VR, gaming, HMI.

5. NLP
- HMM is used in NLP part of speech tagging, assigning grammatical category.
- chatbots, machine translation.

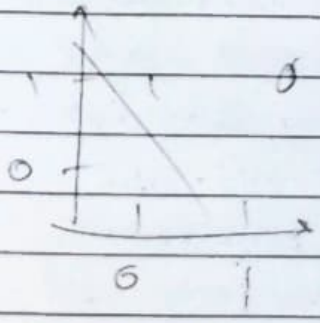
Q3.



AND func
↓
Linearly separable



OR func
↓
Linearly Separable



← XOR
Non linearly separable

- Radial basis function performs linear transformation over ip vectors before the ip vectors are feed for classification.
- Using such non linear transformation, it is possible to convert a non linearly separable data ip to linearly separable data.

- RBFN also increase the dimensionality of i/p feature vectors to convert non linearly separable problem into linear space

case 1] convert by apply xx transformation function

- RBFN increase dimensionality of i/p. vector space, separable in lower dimension is non linear becomes linear in higher dimension.

Assign 2

60004200107

Q1. 1. Model base learning

- Agent learns to predict dynamics of its environment, building an internal model of environment's state transition and reward
- In model based learning, agents used experience to update its internal model, which typically consists of two main components

a) State transition model: This component predicts next state given current state and action. It captures the environment dynamics and can be represented as probability distribution over next state

b) Reward model: The component estimates the expected reward given current state and action. It captures environment's reward structure

- Once agent has an accurate model, it can use planning algorithms.

2. Temporal based learning

- Temporal difference learning is a model free method which means that agent does not learn an explicit model of environment's dynamics.
- Instead it directly learns an optimal policy or value function by updating its estimates using the difference between current and

predicted future rewards, known as temporal difference error

- TD learning is combination of two other reinforcement learning : monte carlo and dynamic programming.
- It combines idea of sampling from MC learning with bootstrapping from DP.
- There are two primary techniques.

a) SARSA (State Action Reward and State Action)
This is an on policy TD learning algorithm, which means it learns the value of the policy being followed. The agent updates its action value based on current state

b) Q Learning : based on off policy TD algorithm, which means it learns value of optimal policy regardless of policy being followed. The agent updates its function based on the current state

Q2

- ML can be used effectively for video surveillance to analyze and process video data, enabling intelligent decision making and automating various tasks
- It can perform task like:

1) Object recognition & tracking :
ML algorithms can be trained to identify and track objects, such as people, vehicles and animals in real time. This allows surveillance system to monitor specific objects of interest

2) Motion detection
ML can be used to detect and analyze motion in video streams, enabling system to identify unusual activity and trigger alerts or other actions when necessary

3) Behaviour analysis
ML algorithms can be trained to recognize and analyze specific behaviours or actions, such as people loitering, fights or thefts., allowing system to respond to such dangerous situations

4) Anomaly detection
ML can be used to establish normal patterns of activity in a video stream and identify any deviations from these patterns, signaling potential security threats

5. Facial recognition

ML techniques can be used to identify individuals in video streams, allowing for personalized security measures, access control.

6. Crowd analysis.

ML can be applied to analyze crowd behaviour, density and movement patterns can be used for public safety.