Dayananda Sagar
College Of Engineering

# Department of Computer Science and Engineering

# Lab Manual - 22CS53

# Artificial Intelligence & Machine Learning Lab

## Academic Year - 2025-26

### Course Outcomes

| | |
|---|---|
| **CO1** | Comprehend the working of intelligent agents and their working environments |
| **CO2** | Make use of logic to represent knowledge for reasoning of AI. |
| **CO3** | Apply linear regression to build machine learning models to solve real life problems |
| **CO4** | Analyze and apply classification algorithms efficiently |
| **CO5** | Evaluate and select suitable clustering technique for problem solving. |
| **CO6** | Design and Implement different supervised, unsupervised machine learning techniques for real world applications |

# Table of Contents

# GUIDELINES & INSTRUCTIONS TO STUDENTS

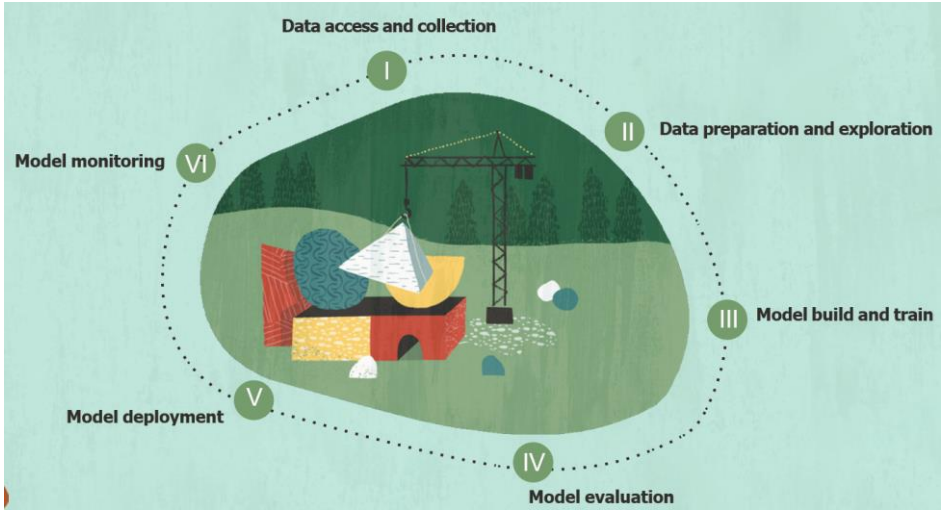
- **Bring your college ID, class notes, lab observation book, and lab record to each lab session.**
- **Sign in and out of the lab register.**
- **Arrive on time; late arrivals exceeding 15 minutes may not be permitted.**
- **100% lab attendance is mandatory.**
- **Adhere to the dress code.**
- **No food or drinks allowed.**
- **Leave bags in the designated area.**
- **Seek assistance from lab staff for any queries.**
- **Respect the lab and fellow students.**
- **Maintain a clean and tidy workspace.**
- **Do not use external storage devices (floppy disks, pen drives) without lab in-charge permission.**

# PREAMBLE

AIML laboratory manual has been designed to provide students of the fifth semester Computer Science & Engineering program with practical exposure to Artificial Intelligence and Machine Learning concepts. It complements theoretical learning by enabling students to implement algorithms, analyze datasets, and interpret model performance. The experiments included cover a wide spectrum of machine learning techniques, ranging from regression and classification to clustering, neural networks, and deep learning frameworks. Each exercise is structured to promote hands-on practice, critical thinking, and problem-solving skills aligned with real-world applications. The manual also emphasizes collaborative learning, adherence to laboratory discipline, and ethical use of computing resources. By following the guided programs and exercises, students will develop strong foundational skills to pursue advanced studies, industry projects, and innovative research in AI and ML. Ultimately, this manual aims to bridge classroom knowledge with practical competence, preparing students to become industry-ready professionals.

# Life Cycle of Machine Learning Models

Machine Learning model development follows a systematic lifecycle that ensures data is effectively utilized to build accurate and deployable solutions. The lifecycle begins with **data collection and access**, followed by **data preparation and exploration** to clean, transform, and analyze the dataset. Next, the **model is built and trained** using suitable algorithms, and its performance is assessed during the **evaluation stage** using appropriate metrics. Once validated, the model can be **deployed** into applications to make predictions on unseen data and further **monitored** to maintain accuracy and reliability over time. Each lab program in this course demonstrates how these lifecycle steps are applied to different machine learning techniques.




**1. Data Access and Collection**
This step involves gathering the right data from databases, sensors, surveys, or external sources. Data can be structured (tables) or unstructured (text, images, videos), and often requires ETL pipelines to make it usable.

**2. Data Preparation and Exploration**
Raw data is cleaned, transformed, and visualized to detect missing values, outliers, or correlations. This step ensures the dataset is accurate, well-structured, and ready for model building.

**3. Model Build and Train**
Here, the right algorithm is chosen (e.g., regression, classification, clustering). The dataset is split into training/testing parts, and hyperparameters may be tuned to improve performance.

**4. Model Evaluation**
The trained model is tested using metrics (e.g., $R^2$ score, accuracy, precision/recall, RMSE). This step checks if the model generalizes well to unseen data and meets the business goal.

**5. Model Deployment**
Once validated, the model is packaged and integrated into applications. Deployment can be batch-based (scheduled predictions) or real-time (instant predictions, e.g., fraud detection).

**6. Model Monitoring**
After deployment, the model's accuracy and system performance are continuously tracked. Monitoring detects data drift, performance drop, or operational issues and triggers retraining when needed.

## Program 1:

**Apply: -**
**Simple linear regression model for head Brain dataset and predict brain weight based on head size using the least square method.**
**Find out**

**i. $R^2$ score for the predicted model.**
**ii. Display all the data points along with the fitting the data points to the model.**

```
#importing libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

# Reading Data

data = pd.read_csv('headbrain.csv')

print(data.shape)

data.head()
```

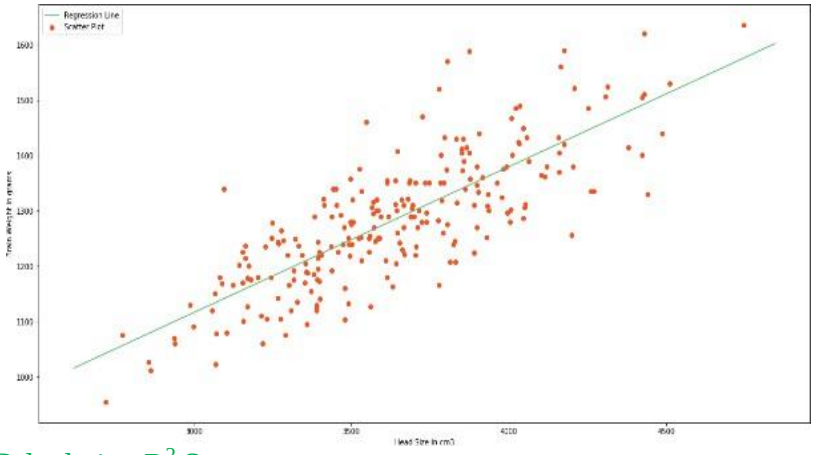
(237, 4)

| | Gender | Age Range | Head Size(cm^3) | Brain Weight(grams) |
|---|---|---|---|---|
| 0 | 1 | 1 | 4512 | 1530 |
| 1 | 1 | 1 | 3738 | 1297 |
| 2 | 1 | 1 | 4261 | 1335 |
| 3 | 1 | 1 | 3777 | 1282 |
| 4 | 1 | 1 | 4177 | 1590 |

```
# Collecting X and Y

X = data['Head Size(cm^3)'].values

Y = data['Brain Weight(grams)'].values

# Calculating coefficient

# Mean X and Y

mean_x = np.mean(X)

mean_y = np.mean(Y)

print(mean_x)
```

```
print(mean_y)

# Total number of values

n = len(X)

print(n)

3633.9915611814345
1282.873417721519
237
# Using the formula to calculate b1 and b0

numer = 0

denom = 0

for i in range(n):

numer += (X[i] - mean_x) * (Y[i] - mean_y)

denom += (X[i] - mean_x) ** 2

b1 = numer / denom

b0 = mean_y - (b1 * mean_x)

# Printing coefficients

print("Coefficients")

print(b1, b0)

Coefficients
b1:0.26342933948939945 b0:325.57342104944223

# Plotting Values and Regression Line

max_x = np.max(X) + 100

min_x = np.min(X) - 100

# Calculating line values x and y

x = np.linspace(min_x, max_x, 1000)

y = b0 + b1 * x

# Ploting Line

plt.plot(x, y, color='#58b970', label='Regression Line')
# Ploting Scatter Points

plt.scatter(X, Y, c='#ef5423', label='Scatter Plot')
plt.xlabel('Head Size in cm3')
```

```
plt.ylabel('Brain Weight in grams')

plt.legend()

plt.show()
```




```
# Calculating R2 Score

ss_tot = 0

ss_res = 0

for i in range(n):

y_pred = b0 + b1 * X[i]

ss_tot += (Y[i] - mean_y) ** 2

ss_res += (Y[i] - y_pred) ** 2

r2 = 1 - (ss_res/ss_tot)

print("R2 Score")

print(r2)
```

$R^2$ Score

0.6393117199570003

**Conclusion: The simple linear regression model gives average accuracy depending on the $R^2$ score value.**

**2b. Simple linear regression model for housing_ prices_ SLR dataset and predict house price based on the area of the house using the library scikit_ learn.**

**Find out**

**i. Analyze the $R^2$score of predicted training and test models score.**

**ii. Display all the data points along with the fitting the data points to the model.**

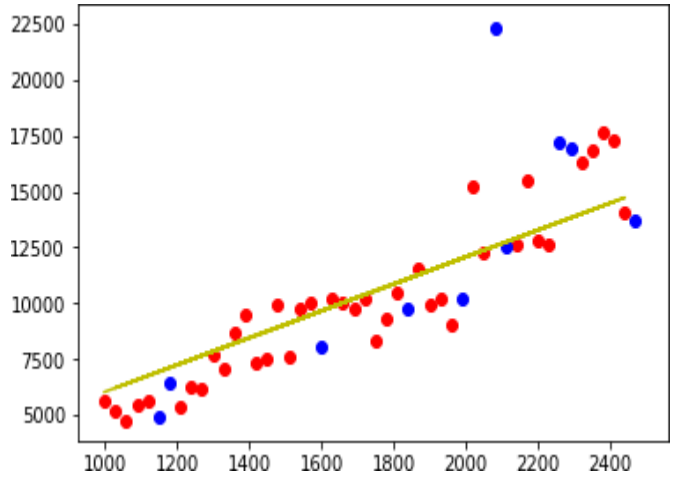
```
# Step1:importing all the libraries
import numpy as np

import pandas as pd

importmatplotlib.pyplot as plt

%matplotlib  inline

# Step2:load dataset

df=pd.read_csv("housing_prices_SLR.csv",delimiter=',')

df.head()
```

| | AREA | PRICE |
|---|---|---|
| 0 | 1000 | 5618 |
| 1 | 1030 | 5201 |
| 2 | 1060 | 4779 |
| 3 | 1090 | 5425 |
| 4 | 1120 | 5657 |

```
Step3: Feature matrix and Target vector

x=df[['AREA']].values#feature Matrix

y=df.PRICE.values#Target Matrix

x[:5] #slicing

y[:5]

Step4: Split the data into 80-20

#from packagename import function

from sklearn.model_selection import train_test_split

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=100)

#80 20 split,random_state to reproduce the same split everytime

print(x_train.shape)

print(x_test.shape)

print(x_train.shape)
```

```
print(x_test.shape)

(40, 1)
(10, 1)
(40, 1)
(10, 1)

#step5: Fit the line:Train the SLR Model

From sklearn.linear_model import Linear Regression

lr_model= Linear Regression()

lr_model.fit(x_train,y_train)

print(lr_model.intercept_) # (PRICE=(-4481.80028058845)+8.65903854)*AREA

print(lr_model.coef_)#y=c+mx

b0:-3103.34066448488
b1:[7.75979089]
lr_model=Linear Regression(fit_intercept= False)

lr_model.fit(x_train,y_train)

print(lr_model.intercept_) # (PRICE=(-4481.80028058845)+8.65903854)*AREA

print(lr_model.coef_)#y=c+mx

b0:0.0
b1:6.03609138

#step6: predict using the model

From sklearn.metrics import r2_score

y_train

lr_model.predict(x_train)

# step7: calculating R^2score using tain and test model

r2_score(y_train,lr_model.predict(x_train))

R^2_Train_Score:0.820250203127675

r2_score(y_test,lr_model.predict(x_test))

R^2_Test_Score:0.5059420550739799

lr_model.score(x_test,y_test) #2.second way of calculating R2 score

R^2_Test_Score:0.5059420550739799

step8:Visualizing the model

plt.scatter(x_train[:,0],y_train,c='red')
```

```
plt.scatter(x_test[:,0],y_test,c='blue')

plt.plot(x_train[:,0],lr_model.predict(x_train),c='y')
```




**Conclusion: Comparing the training and testing R^2 score values, the accuracy of the simple linear regression model with respect to this dataset is average.**

## Program 2

**Apply:**

**a)Multiple linear regression model for student dataset and predict writing skill of student based on the math skill and reading skill of the student using the Gradient descent method. Find out $R^2$ score for the predicted model.**

```
#importing Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
```

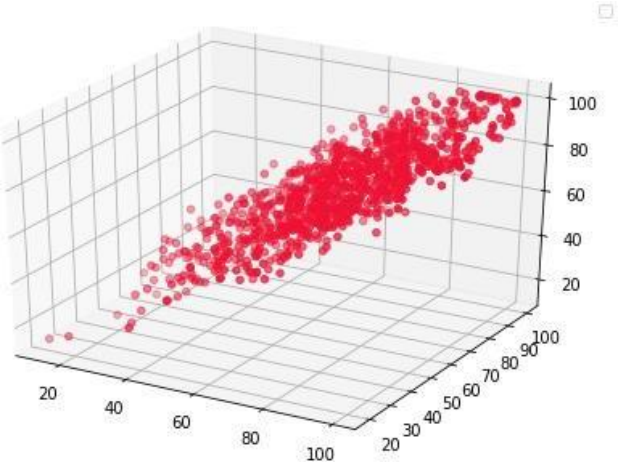
```
data = pd.read_csv('student.csv')
print(data.shape)
data.head()
```

(1000, 3)

| | Math | Reading | Writing |
|---|---|---|---|
| **0** | 48 | 68 | 63 |
| **1** | 62 | 81 | 72 |
| **2** | 79 | 80 | 78 |
| **3** | 76 | 83 | 79 |
| **4** | 59 | 64 | 62 |

```
math = data['Math'].values
read = data['Reading'].values
write = data['Writing'].values
```

```
# Ploting the scores as scatter plot
fig = plt.figure()
ax = Axes3D(fig)
ax.scatter(math, read, write, color='#ef1234')
plt.legend()
plt.show()
```




```
m = len(math)
x0 = np.ones(m)
X = np.array([x0, math, read]).T
```

```
# Initial Coefficients
B = np.array([0, 0,
0]) Y =
np.array(write)
alpha = 0.0001
defcost_function(X, Y,
   B): m = len(Y)
   J = np.sum((X.dot(B) - Y) ** 2)/(2 * m)
   return J

inital_cost = cost_function(X, Y,
B) print("Initial Cost")
print(inital_cost)

defgradient_descent(X, Y, B, alpha,
iterations): cost_history = [0] * iterations
   m = len(Y)

   for iteration in
      range(iterations): #
      Hypothesis Values
      h = X.dot(B)
      # Difference b/w Hypothesis and
```

```
        gradient = X.T.dot(loss) / m
        # Changing Values of B using
        Gradient B = B - alpha * gradient
        # New Cost Value
cost = cost_function(X, Y, B)
cost_history[iteration] = cost

    return B, cost_history

# 100000 Iterations
newB, cost_history = gradient_descent(X, Y, B, alpha, 100000)

# New Values of B
print("New
Coefficients")
print(newB)

# Final Cost of new
```

```
Initial Cost
2470.11
New Coefficients
[bo, b1,b2]:[-0.47889172  0.09137252  0.90144884]
Final Cost
10.475123473539167
```

```
# Model Evaluation - RMSE
defrmse(Y, Y_pred):
rmse = np.sqrt(sum((Y - Y_pred) ** 2) / len(Y))
    return rmse
```

```
# Model Evaluation - R2
Score def r2_score(Y,
Y_pred): mean_y =
np.mean(Y)
ss_tot = sum((Y - mean_y) ** 2)
ss_res = sum((Y - Y_pred) ** 2)
```

```
    return r2


Y_pred = X.dot(newB)


print("R2 Score")
print(r2_score(Y,
```

$R^2$ Score
0.9097223273061553

**Conclusion:**
**The accuracy of the multiple linear regression model is good depending on the $R^2$score value.**

**b.) Multiple linear regression model for housing_prices dataset and predict house price based on the area, floor and room size of the house using the library scikit learn. Find out the accuracy of the model using $R^2$score statistics for the predicted model.**

#importing libraries

```
import numpy as np
import pandas as pd
importmatplotlib.pyplot as plt
%matplotlib  inline
```

```
#Loading dataset
df=pd.read_csv("housing_prices.csv")
df.head()
```

| | AREA | FLOOR | ROOM | PRICE |
|---|---|---|---|---|
| **0** | 1000 | 7 | 2 | 5618 |
| **1** | 1030 | 7 | 1 | 5201 |
| **2** | 1060 | 1 | 1 | 4779 |
| **3** | 1090 | 6 | 1 | 5425 |
| **4** | 1120 | 0 | 2 | 5657 |

```
#setting Target and Feature Vectors
x=df.iloc[:,:3].values
y=df.iloc[:,3].values
```

```
#Splittiing the dataset
fromsklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=100)
```

```
# Fitting the model
from sklearn.linear_model import LinearRegression
mlr_model= LinearRegression(fit_intercept=True)
mlr_model.fit(x_train,y_train)
print(mlr_model.intercept_) # (PRICE=(-4481.80028058845)+8.65903854)*AREA
print(mlr_model.coef_)
```

```
b0:-3106.4127920034116
[b1,b2,b3]:[   4.68576316   71.78274093 1894.45529322]
```

```
# Finding R2 score
print(mlr_model.score(x_train,y_train))
print(mlr_model.score(x_test,y_test))
```

```
R2_Train_Score:0.9220702400776505
R2_Test_Score:0.8090037959414931
```

**Conclusion**:**The multiple linear regression model accuracy is good with respect to this dataset by comparing R2 training and testing score values.**

## Program 3

**Apply:**

a) **Decision tree on breast cancer dataset.**
**Find out**
   i) **No of benign and malignant cases in the testing phase.**
   ii) **Predict the accuracy of the both classifier.**

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

df = pd.read_csv('breast_cancer.csv')
df = df.iloc[:, :-1]
df.head()
x = df.iloc[:, 2:].values
print(x)
y = df.diagnosis.values
print(x[:2])
print(y[:5])
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2)
from sklearn.tree import DecisionTreeClassifier
dt_classifier = DecisionTreeClassifier()
dt_classifier.fit(x_train, y_train)
predictions = dt_classifier.predict(x_test)
prob_predictions = dt_classifier.predict_proba(x_test)
print(predictions)
print(prob_predictions)
from sklearn.metrics import accuracy_score, confusion_matrix ,classification_report
print("Training accuracy Score is : ", accuracy_score(y_train,
        dt_classifier.predict(x_train)))
print("Testing accuracy Score is : ", accuracy_score(y_test,
     dt_classifier.predict(x_test)))
print("Training Confusion Matrix is : \n", confusion_matrix(y_train,
  dt_classifier.predict(x_train)))
print("Testing Confusion Matrix is : \n", confusion_matrix(y_test,
   dt_classifier.predict(x_test)))
print(classification_report(y_test,dt_classifier.predict(x_test)))
```

**Output:**

```
Training accuracy Score is :  1.0
Testing accuracy Score is :  0.9385964912280702

Training Confusion Matrix is:
 [[286   0]
 [0 169]]

Testing Confusion Matrix is:
 [[71  0]
 [ 7 36]]
```

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| B | 0.97 | 0.92 | 0.94 | 73 |
| M | 0.87 | 0.95 | 0.91 | 41 |
| accuracy | | | 0.93 | 114 |
| macro avg | 0.92 | 0.93 | 0.93 | 114 |
| weighted avg | 0.93 | 0.93 | 0.93 | 114 |

**Conclusion:**

**Comparing Training and testing accuracy scores the accuracy of Decision Tree model is good. The Correctly classified tuples for training set is (286+169) and the misclassified tuples are zero.The correctly classified for training set is (71+36)and misclassified tuples are(7+0).**

**3b. Apply Naïve Bayesian classifier on breast cancer dataset.**

**Find out**

**i) No of benign and malignant cases in the testing phase.**

**ii) Predict the accuracy of the classifier**

```
# coding: utf-8
# ## Implementation of Naïve Bayes Algorithm
# ### Step 1 : Load required packages

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sklearn as sk

#### Step 2 : Load the csv/excel file into pandas dataframe and clean the data

df = pd.read_csv("breast_cancer.csv")
df = df.iloc[:, :-1]
df.shape()
df.head()

# ### Step 3 : Create the Feature Matrix and Target Vector and check the first 5 rows

x=df.iloc[:,2:].values
y=df.diagnosis.values

print(x[:2])
print(y[:5])


# ###Step 4 : Split the data into training set and test set from sklearn.model_selection

import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2,random_state=500)

x_train.shape #(455,30)

 x_test.shape#(114, 30)
y_train.shape

y_test.shape

(y_train == 'M').sum()

(y_train=='B').sum()
# Baseline model, accuracy, confusion_matrix, classification_report
```

#### Step 5: Instantiate a Guassian Naive Bayes model and train the model

```
278/len(y_train)
# Baseline model of
accuracy =(more number of occurrences)/total data elements

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

baseline_pred=["B"] *len(y_train) # baseline will have beningn for everything
Baseline model of accuracy :0.610989010989011

accuracy_score(y_train,baseline_pred) # takes actual and predicted as 2 arguments
confusion_matrix(y_train,baseline_pred)# takes actual and predicted as 2 arguments


from sklearn.naive_bayes import GaussianNB
nb_model=GaussianNB()
nb_model.fit(x_train,y_train)
print(x_train)
nb_model.score(x_train,y_train)
nb_model.score(x_test,y_test)

#confusion_matrix for training data

confusion_matrix(y_train,nb_model.predict(x_train))

Training Confusion Matrix:
  array([[269,   9],
         [ 22, 155]],
    dtype=int64)

#confusion_matrix for test data
confusion_matrix(y_test,nb_model.predict(x_test))

Testing Confusion Matrix:
array([[78,  1],
       [ 2, 33]],
dtype=int64)

print(classification_report(y_train,nb_model.predict(x_train)))


   precision    recall  f1-score   support

B       0.92      0.97      0.95       278
M       0.95      0.88      0.91       177

   avg / total       0.93      0.93      0.93       455



print(classification_report(y_test,nb_model.predict(x_test)))

   precision    recall  f1-score   support
```

| | | | | |
|---|---|---|---|---|
| B | 0.97 | 0.99 | 0.98 | 79 |
| M | 0.97 | 0.94 | 0.96 | 35 |
| avg / total | 0.97 | 0.97 | 0.97 | 114 |

**Conclusion: The naïve bayes model is good with respect to breast cancer dataset by comparing the precision recall and F1 score values of training and testing dataset (classification report)**

## Program 4:

**Apply:**

**a)Partitioning k-means clustering technique on ch1ex1 dataset with different K (number of clusters) as input and record the output.**

**Step 1 and 2: Import the libraries and Load the dataset.**

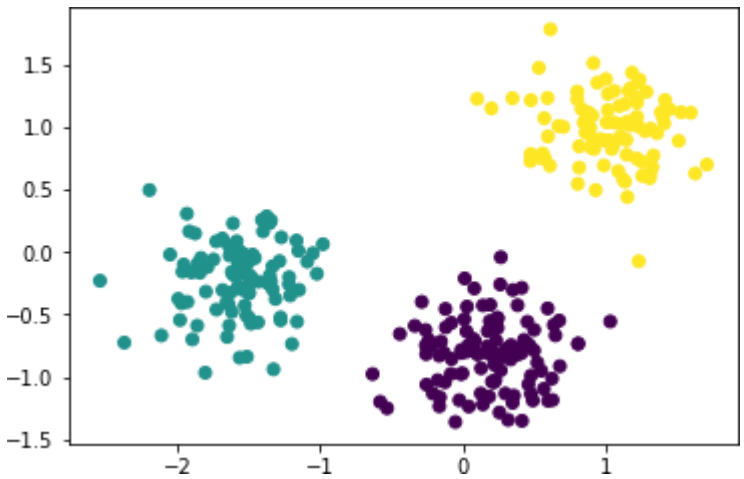
```
import pandas as pd
df = pd.read_csv('ch1ex1.csv')
points = df.values
from sklearn.cluster import KMeans
model = KMeans(n_clusters=3)
model.fit(points)
labels = model.predict(points)
importmatplotlib.pyplot as plt
```

**Step 2:** Assign column 0 of points to xs, and column 1 of points to ys

```
xs = points[:,0]
ys = points[:,1]
```

**Step 3:** Make a scatter plot of xs and ys, specifying the c=labels keyword arguments to color the points by their cluster label. You'll see that KMeans has done a good job of identifying the clusters!

```
plt.scatter(xs, ys, c=labels)
plt.show()
```




#**This is great**, but let's go one step further, and add the cluster centres (the "centroids") to the scatter plot.

**Step 3:** Obtain the coordinates of the centroids using the .cluster_centers_ attribute of model. Assign them to centroids.

```
centroids = model.cluster_centers_
```

**Step 4:** Assign column 0 of centroids to centroids_x, and column 1 of centroids to centroids_y.

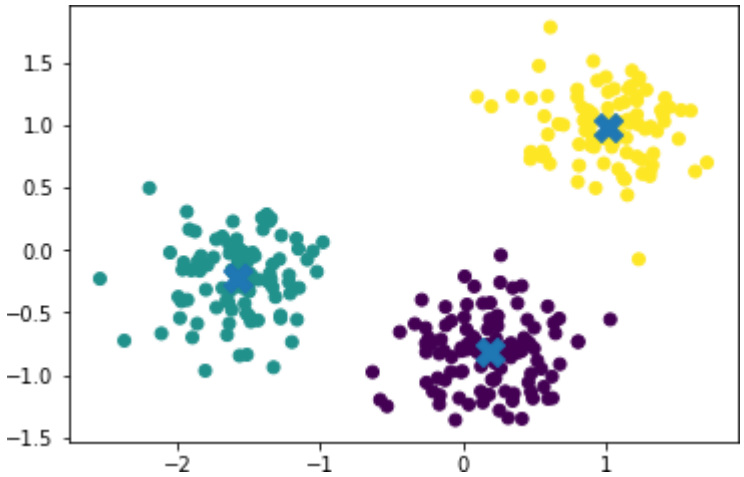
```
centroids_x = centroids[:,0]
centroids_y = centroids[:,1]
```

**Step 5: In a single cell, create two scatter plots (this will show the two on top of one another). Call `plt.show()` just once, at the end.**

**Firstly, the make the scatter plot you made above. Secondly, make a scatter plot of `centroids_x` and `centroids_y`, using `'X'` (a cross) as a marker by specifying the `marker` parameter. Set the size of the markers to be `200` using `s=200`.**

```
plt.scatter(xs, ys, c=labels)
plt.scatter(centroids_x, centroids_y, marker='X', s=200)
plt.show()
```

**Output:**




The centroids are important because they are what enables KMeans to assign new, previously unseen points to the existing clusters.

**Conclusion: The k-means clustering technique is applied to ch1ex1 dataset to form clusters depending on the number of clusters as input. Then the centroid of the clustering is shown using the cross mark.**

## 4b) Hierarchical Clustering Algorithm on seeds_less_rows dataset for extracting cluster labels of different varieties of seeds

### #Extracting the cluster labels in heirarchial clustering

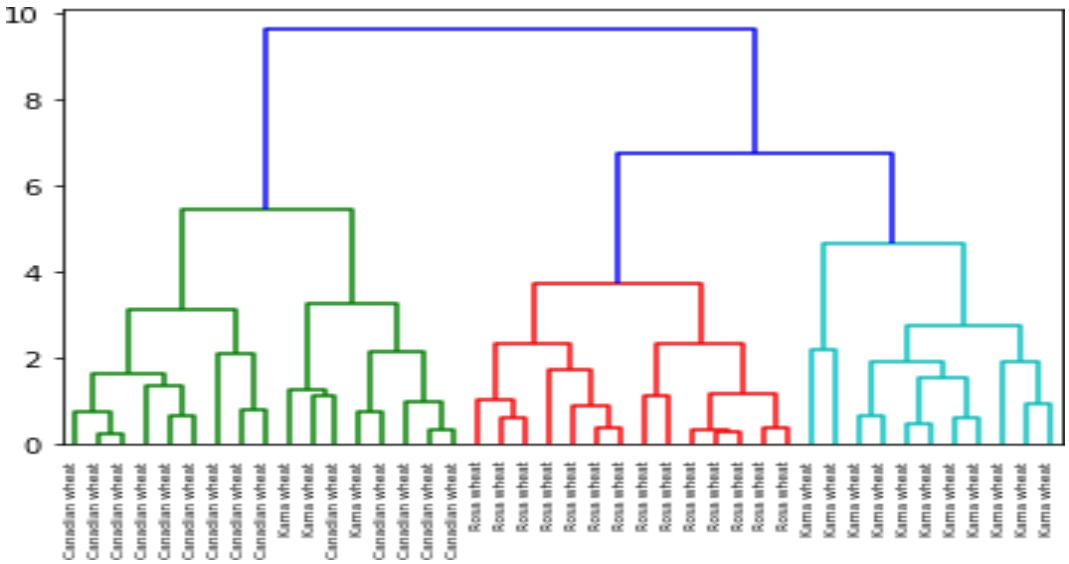
#we use the fcluster() function to extract the cluster labels for intermediate clustering, and #compare the labels with the grain varieties using a cross-tabulation.

**Step 1 and 2:** importing libraries and load the dataset:

```
import pandas as pd

seeds_df = pd.read_csv('seeds-less-rows.csv')

# remove the grain species from the DataFrame, save for later

varieties = list(seeds_df.pop('grain_variety'))

# extract the measurements as a NumPy array

samples = seeds_df.values
```

**Step 3:** Run the hierarchical clustering of the grain samples

```
fromscipy.cluster.hierarchy import linkage, dendrogram

importmatplotlib.pyplot as plt

mergings = linkage(samples, method='complete')

dendrogram(mergings,labels=varieties,leaf_rotation=90,leaf_font_size=6)

plt.show()
```




**Step 4:** Import fcluster from scipy.cluster.hierarchy

```
In[11]:  from scipy.cluster.hierarchy import fcluster
```

**Step 5:** Obtain a flat clustering by using the fcluster() function on mergings. Specify a maximum height of 6 and the keyword argument criterion='distance'. Assign the result to labels.

In[12]: labels = fcluster(mergings, 6, criterion='distance')

**Step 6:** Create a DataFramedf with two columns named 'labels' and 'varieties', using labels and varieties, respectively, for the column values.

In[13]: df = pd.DataFrame({'labels': labels, 'varieties': varieties})

**Step 7:** Create a cross-tabulation ct between df['labels'] and df['varieties'] to count the number of times each grain variety coincides with each cluster label.

In[14]: ct = pd.crosstab(df['labels'], df['varieties'])

**Step 8:** Display ct to see how your cluster labels correspond to the wheat varieties.

In[15]: ct

**Output:-**

Out[15]:

| varieties<br>labels | Canadian wheat | Kama wheat | Rosa wheat |
|---|---|---|---|
| 1 | 14 | 3 | 0 |
| 2 | 0 | 0 | 14 |
| 3 | 0 | 11 | 0 |

**Conclusion: Three varieties of labels extracted from 'seeds-less-rows' dataset by applying Hierarchical clustering technique as shown in the output table.**

## Program 5

### Demonstrate:

**a) Usage of Sigmoid activation function in artificial neural network**

```
import numpy as np
from functools import reduce
def perceptron(weight, bias, x):
model = np.add(np.dot(x, weight), bias)
print('model: {}'.format(model))
logit  =  1/(1+np.exp(-model))
print('Type: {}'.format(logit))
returnnp.round(logit)
def compute(logictype, weightdict, dataset):
weights = np.array([ weightdict[logictype][w] for w in weightdict[logictype].keys()])
output = np.array([ perceptron(weights, weightdict['bias'][logictype], val) for val in dataset])
    print(logictype)
    return logictype, output
def main():
    logic = {
        'logic_and' : {
            'w0': -0.1,
            'w1': 0.2,
            'w2': 0.2
        },
        'logic_nand': {
            'w0': 0.6,
            'w1': -0.8,
            'w2': -0.8
        },
        'bias': {
            'logic_and': -0.2,
            'logic_nand': 0.3,
```

```
        }
    }
dataset = np.array([
        [1,0,0],
        [1,0,1],
        [1,1,0],
        [1,1,1]   ])
logic_and = compute('logic_and', logic, dataset)
logic_nand = compute('logic_nand', logic, dataset)
def template(dataset, name, data):
   # act = name[6:]
print("Logic Function: {}".format(name[6:].upper()))
        print("X0\tX1\tX2\tY")
to Print = ["{1}\t{2}\t{3}\t{0}".format(output, *datas) for datas, output in zip(dataset, data)]
for i in to Print:
 print(i)
 gates = [logic_and, logic_nand]
for i in gates:
template(dataset, *i)
if __name___== '__main__':
main()
```

**output:**

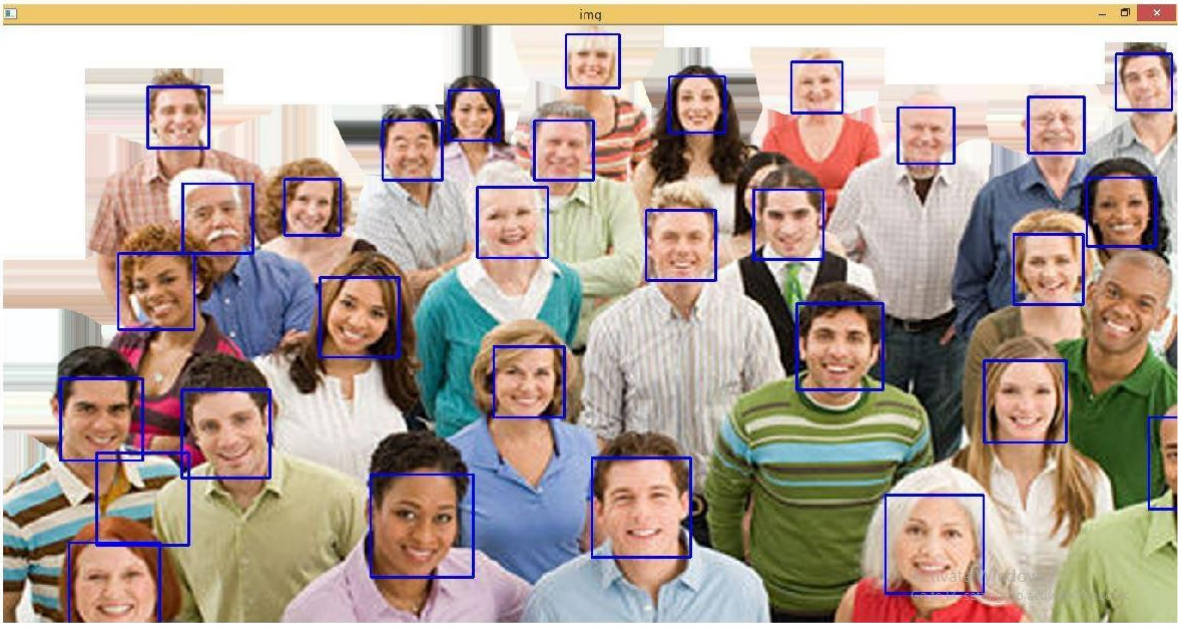
```
model: -0.30000000000000004
Type: 0.425557483188341
model: -0.1
Type: 0.47502081252106
model: -0.1
Type: 0.47502081252106
model: 0.10000000000000003
Type: 0.52497918747894
logic_and
```

```
model: 0.8999999999999999

Type: 0.7109495026250039

model: 0.09999999999999992

Type: 0.5249791874789399

model: 0.09999999999999992

Type: 0.5249791874789399

model: -0.7

Type: 0.3318122278318339

logic_nand

Logic Function: AND
X0      X1      X2      Y
1       0       0       0.0
1       0       1       0.0
1       1       0       0.0
1       1       1       1.0
Logic Function: NAND
X0      X1      X2      Y
1       0       0       1.0
1       0       1       1.0
1       1       0       1.0
1       1       1       0.0
```

**Conclusion: Sigmoid or logistic function used to display the working of AND and NAND logic functions.**

## 5b)Identification of face using opencv library

```
#using opencv
        #install -c menpoopencv
        import numpy as np
        import cv2
        face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
        img = cv2.imread('people.jpg')
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        faces = face_cascade.detectMultiScale(gray, 1.1, 5)
        for (x,y,w,h) in faces:
        cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
        roi_gray = gray[y:y+h, x:x+w]
        roi_color = img[y:y+h, x:x+w]
        cv2.imshow('img',img)
        cv2.waitKey(0)
        cv2.destroyAllWindows()
```




**Conclusion: Using open cv library of Neural Networks, faces are detected.**

## Program 6

### Using Keras and Tensor flow framework

**i) Load the Pima_indians_diabetes dataset**
**ii) Design a two-layer neural network with one hidden layer and one output layer**
**a. Use Relu activation function for the hidden layer**
**b. Use sigmoid activation function for the output layer**
**iii) Train the designed network for Pima_indians_diabetes**
**iv)Evaluate the network**
**v) Generate Predictions for 10 samples**

Seven key steps in using Keras to create a neural network or deep learning model, step-by-step including:

1) Importing necessary Libraries 2) How to load data. 3) How to define a neural network in Keras. 4) How to compile a Keras model using the efficient numerical backend. 5) How to train a model on data. 6) How to evaluate a model on data. 7) How to make predictions with the model.

```
# first neural network with keras tutorial
from numpy import loadtxt
import numpy as np
import pandas as pd
from keras import models
from keras.models import Sequential
from keras.layers import Dense
from keras import layers
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
import matplotlib.pyplot as plt
dataframe=pd.read_csv('pima-indians-diabetes.csv',delimiter=',')
dataframe.head()
```

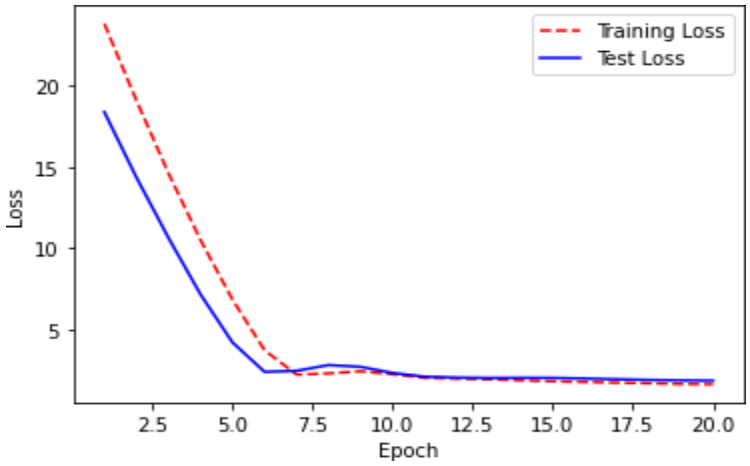
| | 6 | 148 | 72 | 35 | 0 | 33.6 | 0.627 | 50 | 1 |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 85 | 66 | 29 | 0 | 26.6 | 0.351 | 31 | 0 |
| 1 | 8 | 183 | 64 | 0 | 0 | 23.3 | 0.672 | 32 | 1 |
| 2 | 1 | 89 | 66 | 23 | 94 | 28.1 | 0.167 | 21 | 0 |
| 3 | 0 | 137 | 40 | 35 | 168 | 43.1 | 2.288 | 33 | 1 |
| 4 | 5 | 116 | 74 | 0 | 0 | 25.6 | 0.201 | 30 | 0 |

```
# split into input (X) and output (y) variables
X=dataframe.iloc[:,:8]
y=dataframe.iloc[:,8]

dataframe.shape
(767, 9)

features_train,features_test,target_train,target_test=train_test_split(X,y,
test_size=0.33,random_state=0)

# define the keras model
network=models.Sequential()
network.add(Dense(units=8,activation="relu",input_shape=(features_train.sha
pe[1],)))
```

```
network.add(Dense(units=8,activation="relu"))
```

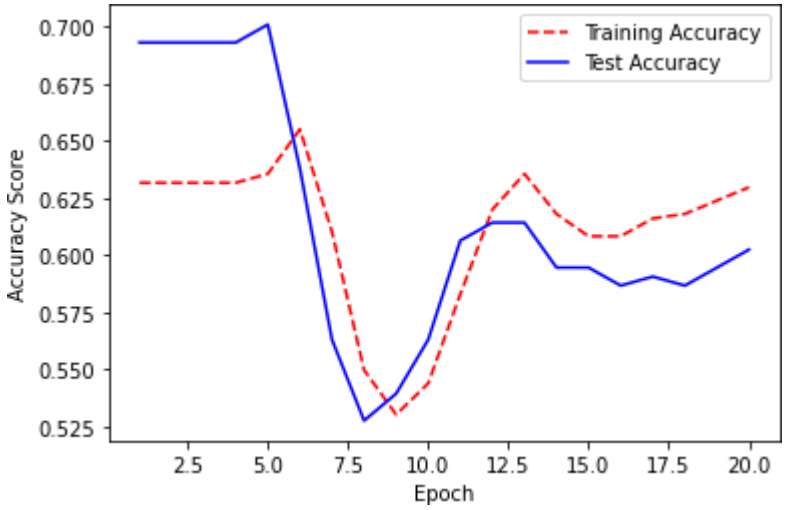
```
#network.add(Dense(units=16,activation="relu"))
network.add(Dense(units=1,activation="sigmoid"))
# compile the keras model
network.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accur
acy'])
#network.compile(loss='mse', optimizer='RMSprop', metrics=['accuracy'])
# fit the keras model on the dataset
#network.fit(features_train,features_test, epochs=10,
batch_size=100,verbose=2)
history=network.fit(features_train,target_train,epochs=20,verbose=1,batch_s
ize=100,validation_data=(features_test,target_test))
Train on 513 samples, validate on 254 samples
Epoch 1/20
513/513 [==============================] - 0s 327us/step - loss: 23.8525 -
accuracy: 0.6316 - val_loss: 18.4057 - val_accuracy: 0.6929
Epoch 2/20
513/513 [==============================] - 0s 29us/step - loss: 19.1240 -
accuracy: 0.6316 - val_loss: 14.3790 - val_accuracy: 0.6929
Epoch 3/20
513/513 [==============================] - 0s 39us/step - loss: 14.6355 -
accuracy: 0.6316 - val_loss: 10.6533 - val_accuracy: 0.6929
Epoch 4/20
513/513 [==============================] - 0s 47us/step - loss: 10.5196 -
accuracy: 0.6316 - val_loss: 7.1659 - val_accuracy: 0.6929
Epoch 5/20
513/513 [==============================] - 0s 45us/step - loss: 6.8415 -
accuracy: 0.6355 - val_loss: 4.1935 - val_accuracy: 0.7008
Epoch 6/20
513/513 [==============================] - 0s 43us/step - loss: 3.7177 -
accuracy: 0.6550 - val_loss: 2.3824 - val_accuracy: 0.6378
Epoch 7/20
513/513 [==============================] - 0s 33us/step - loss: 2.2131 -
accuracy: 0.6101 - val_loss: 2.4434 - val_accuracy: 0.5630
Epoch 8/20
513/513 [==============================] - 0s 37us/step - loss: 2.2830 -
accuracy: 0.5497 - val_loss: 2.8009 - val_accuracy: 0.5276
Epoch 9/20
513/513 [==============================] - 0s 37us/step - loss: 2.4204 -
accuracy: 0.5302 - val_loss: 2.6900 - val_accuracy: 0.5394
Epoch 10/20
513/513 [==============================] - 0s 39us/step - loss: 2.2307 -
accuracy: 0.5439 - val_loss: 2.3109 - val_accuracy: 0.5630
Epoch 11/20
513/513 [==============================] - 0s 49us/step - loss: 2.0121 -
accuracy: 0.5828 - val_loss: 2.0812 - val_accuracy: 0.6063
Epoch 12/20
513/513 [==============================] - 0s 45us/step - loss: 1.9620 -
accuracy: 0.6199 - val_loss: 2.0272 - val_accuracy: 0.6142
Epoch 13/20
513/513 [==============================] - 0s 37us/step - loss: 1.9209 -
accuracy: 0.6355 - val_loss: 2.0020 - val_accuracy: 0.6142
Epoch 14/20
513/513 [==============================] - 0s 49us/step - loss: 1.8549 -
accuracy: 0.6179 - val_loss: 2.0124 - val_accuracy: 0.5945
Epoch 15/20
513/513 [==============================] - 0s 55us/step - loss: 1.7957 -
accuracy: 0.6082 - val_loss: 2.0066 - val_accuracy: 0.5945
Epoch 16/20
513/513 [==============================] - 0s 45us/step - loss: 1.7566 -
```

```
accuracy: 0.6082 - val_loss: 1.9706 - val_accuracy: 0.5866
Epoch 17/20
513/513 [==============================] - 0s 51us/step - loss: 1.7174 -
accuracy: 0.6160 - val_loss: 1.9221 - val_accuracy: 0.5906
Epoch 18/20
513/513 [==============================] - 0s 39us/step - loss: 1.6742 -
accuracy: 0.6179 - val_loss: 1.8809 - val_accuracy: 0.5866
Epoch 19/20
513/513 [==============================] - 0s 47us/step - loss:  1.6343 -
accuracy: 0.6238 - val_loss: 1.8540 - val_accuracy: 0.5945
Epoch 20/20
513/513 [==============================] - 0s 49us/step - loss:  1.6173 -
accuracy: 0.6296 - val_loss: 1.8372 - val_accuracy: 0.6024
training_loss=history.history["loss"]
test_loss=history.history["val_loss"]
epoch_count=range(1,len(training_loss)+1)
plt.plot(epoch_count,training_loss,"r--")
plt.plot(epoch_count,test_loss,"b-")
plt.legend(["Training Loss","Test Loss"])
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.show()
```




```
_,accuracy=network.evaluate(features_train,target_train)
print('Accuracy: %.2f'%(accuracy*100))
513/513 [==============================] - 0s 215us/step
Accuracy: 63.16
# predict using the keras model
predicted_target=network.predict(features_test)
_,accuracy=network.evaluate(features_test,target_test)
print('Accuracy: %.2f'%(accuracy*100))
254/254 [==============================] - 0s 35us/step
Accuracy: 60.24
#Y=target_train
foriinrange(10):
print(predicted_target[i])
[0.44970706]
[0.4993118]
[0.9906837]
[0.44786653]
[0.02075692]
[0.03176354]
[0.999443]
[0.5751261]
[0.04377431]
[0.8482277]
training_accuracy=history.history["accuracy"]
test_accuracy=history.history["val_accuracy"]
plt.plot(epoch_count,training_accuracy,"r--")
```

```
plt.plot(epoch_count,test_accuracy,"b-")
plt.legend(["Training Accuracy","Test Accuracy"])
plt.xlabel("Epoch")
plt.ylabel("Accuracy Score")
plt.show()
```




**Conclusion :Using Keras and Tensor flow framework loaded the Pima_indians_diabetes dataset and designed a two-layer neural network with one hidden layer and one output layer and generated predictions for 10 samples.**

## Program 7:

### Using Keras and tensor flow network

**i) Load the mnist image dataset**
**ii) Design a two-layer neural network with one hidden layer and one output layer**
**a. Use CNN with Leaky Relu activation function for the hidden layer**
**b. Use sigmoid activation function for the output layer**
**iii) Train the designed network for mnist dataset**
**iv) Visualize the results of**
**a) Training vs validation accuracy**
**b) Training vs Validation loss**

```
import numpy as np
from keras.datasets import mnist
from keras.utils import to_categorical
import matplotlib.pyplot as plt
%matplotlib inline
```

```
Using TensorFlow backend.
```

```
import keras
from keras.models import Sequential,Input,Model
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras.layers.normalization import BatchNormalization
from keras.layers.advanced_activationsimport LeakyReLU
```

```
#from keras.datasets import mnist
(train_X,train_Y), (test_X,test_Y) = mnist.load_data()
print('Training data shape : ', train_X.shape, train_Y.shape)

print('Testing data shape : ', test_X.shape, test_Y.shape)
```

```
Training data shape :  (60000, 28, 28) (60000,)
Testing data shape :  (10000, 28, 28) (10000,)
```

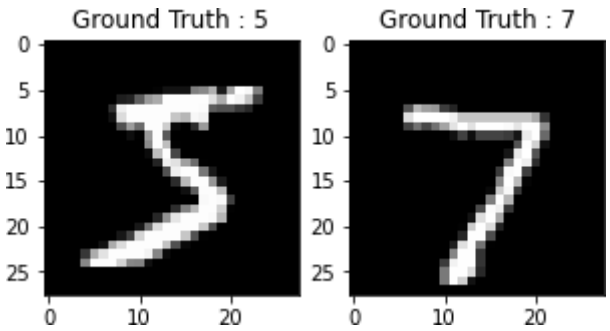
```
# Find the unique numbers from the train labels
classes = np.unique(train_Y)
```

```
nClasses =len(classes)
print('Total number of outputs : ', nClasses)
print('Output classes : ', classes)
```

```
Total number of outputs :  10
Output classes :  [0 1 2 3 4 5 6 7 8 9]
```

```
plt.figure(figsize=[5,5])

# Display the first image in training data
plt.subplot(121)
plt.imshow(train_X[0,:,:], cmap='gray')
plt.title("Ground Truth : {}".format(train_Y[0]))

# Display the first image in testing data
plt.subplot(122)
plt.imshow(test_X[0,:,:], cmap='gray')
plt.title("Ground Truth : {}".format(test_Y[0]))
```

```
Text(0.5, 1.0, 'Ground Truth : 7')
```




```
train_X = train_X.reshape(-1, 28,28, 1)
test_X = test_X.reshape(-1, 28,28, 1)
train_X.shape, test_X.shape
```

```
((60000, 28, 28, 1), (10000, 28, 28, 1))
```

```
train_X = train_X.astype('float32')
test_X = test_X.astype('float32')
train_X = train_X /255
test_X = test_X /255

# Change the labels from categorical to one-hot encoding
train_Y_one_hot = to_categorical(train_Y)
test_Y_one_hot = to_categorical(test_Y)

# Display the change for category label using one-hot encoding
print('Original label:', train_Y[0])
print('After conversion to one-hot:', train_Y_one_hot[0])
```

```
Original label: 5
After conversion to one-hot: [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
```

```
From sklearn.model_selection import train_test_split
train_X,valid_X,train_label,valid_label = train_test_split(train_X,
train_Y_one_hot, test_size=0.2, random_state=13)
train_X.shape,valid_X.shape,train_label.shape,valid_label.shape
```

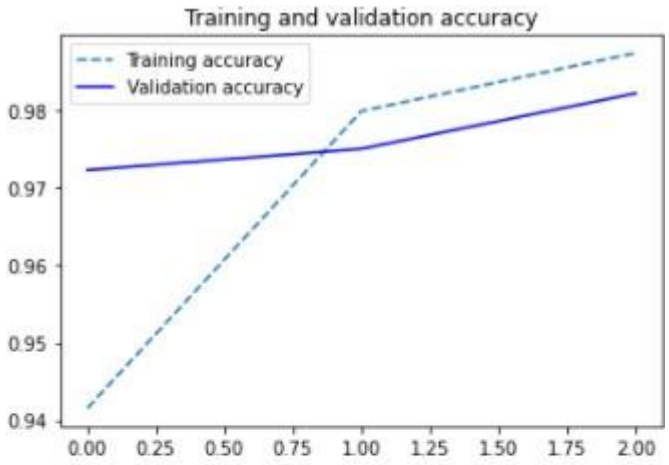
```
((48000, 28, 28, 1), (12000, 28, 28, 1), (48000, 10), (12000, 10))

batch_size =64
epochs =3
num_classes =10

m_model = Sequential()
m_model.add(Conv2D(32, kernel_size=(3,
3),activation='linear',input_shape=(28,28,1),padding='same'))
m_model.add(LeakyReLU(alpha=0.1))
m_model.add(MaxPooling2D((2, 2),padding='same'))
#fashion_model.add(Conv2D(64, (3, 3), activation='linear',padding='same'))
#fashion_model.add(LeakyReLU(alpha=0.1))
#fashion_model.add(MaxPooling2D(pool_size=(2, 2),padding='same'))
#fashion_model.add(Conv2D(128, (3, 3), activation='linear',padding='same'))
#fashion_model.add(LeakyReLU(alpha=0.1))
#fashion_model.add(MaxPooling2D(pool_size=(2, 2),padding='same'))
m_model.add(Flatten())
m_model.add(Dense(128, activation='linear'))
m_model.add(LeakyReLU(alpha=0.1))
m_model.add(Dense(num_classes, activation='softmax'))

m_model.compile(loss=keras.losses.categorical_crossentropy,
optimizer=keras.optimizers.Adam(),metrics=['accuracy'])

m_model.summary()

Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 28, 28, 32)        320
_________________________________________________________________
leaky_re_lu_5 (LeakyReLU)    (None, 28, 28, 32)        0
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
flatten_3 (Flatten)          (None, 6272)              0
_________________________________________________________________
dense_5 (Dense)              (None, 128)               802944
_________________________________________________________________
leaky_re_lu_6 (LeakyReLU)    (None, 128)               0
_________________________________________________________________
dense_6 (Dense)              (None, 10)                1290
=================================================================
Total params: 804,554
Trainable params: 804,554
Non-trainable params: 0
_________________________________________________________________

m_train = m_model.fit(train_X, train_label,
batch_size=batch_size,epochs=epochs,verbose=1,validation_data=(valid_X,
valid_label))

Train on 48000 samples, validate on 12000 samples
Epoch 1/3
48000/48000 [==============================] - 45s 928us/step - loss:
0.1946 - accuracy: 0.9427 - val_loss: 0.0938 - val_accuracy: 0.9713
Epoch 2/3
48000/48000 [==============================] - 46s 948us/step - loss:
0.0630 - accuracy: 0.9811 - val_loss: 0.0733 - val_accuracy: 0.9762
```

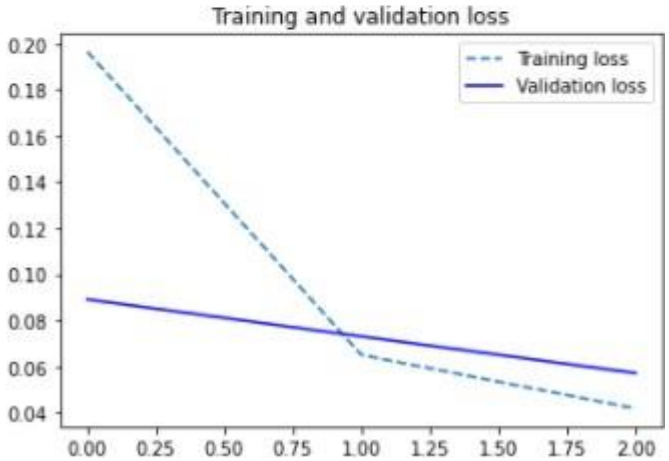
```
Epoch 3/3
48000/48000 [==============================] - 43s 897us/step - loss:
0.0433 - accuracy: 0.9871 - val_loss: 0.0570 - val_accuracy: 0.9819
```

```
test_eval = m_model.evaluate(test_X, test_Y_one_hot, verbose=0)
print('Test loss:', test_eval[0])
print('Test accuracy:', test_eval[1])
```

```
Test loss: 0.052222021067142486
Test accuracy: 0.9824000000953674
```

```
accuracy = m_train.history['accuracy']
val_accuracy = m_train.history['val_accuracy']
loss = m_train.history['loss']
val_loss = m_train.history['val_loss']
epochs =range(len(accuracy))
plt.plot(epochs, accuracy, '--', label='Training accuracy')
plt.plot(epochs, val_accuracy, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, '--', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```




```
epochs=1
```

```
# ADDING DROPOUT
m_model = Sequential()
m_model.add(Conv2D(32, kernel_size=(3,
3),activation='linear',padding='same',input_shape=(28,28,1)))
m_model.add(LeakyReLU(alpha=0.1))
m_model.add(MaxPooling2D((2, 2),padding='same'))
m_model.add(Dropout(0.25))
#fashion_model.add(Conv2D(64, (3, 3), activation='linear',padding='same'))
#fashion_model.add(LeakyReLU(alpha=0.1))
#fashion_model.add(MaxPooling2D(pool_size=(2, 2),padding='same'))
#fashion_model.add(Dropout(0.25))
#fashion_model.add(Conv2D(128, (3, 3), activation='linear',padding='same'))
#fashion_model.add(LeakyReLU(alpha=0.1))
#fashion_model.add(MaxPooling2D(pool_size=(2, 2),padding='same'))
#fashion_model.add(Dropout(0.4))
m_model.add(Flatten())
m_model.add(Dense(128, activation='linear'))
m_model.add(LeakyReLU(alpha=0.1))
m_model.add(Dropout(0.3))
m_model.add(Dense(num_classes, activation='softmax'))
```

```
m_model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)            (None, 28, 28, 32)        320
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 28, 28, 32)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 14, 14, 32)        0
_________________________________________________________________
flatten_2 (Flatten)          (None, 6272)              0
_________________________________________________________________
dense_3 (Dense)              (None, 128)               802944
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 128)               0
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_4 (Dense)              (None, 10)                1290
=================================================================
Total params: 804,554
Trainable params: 804,554
Non-trainable params: 0
_________________________________________________________________
```

```
m_model.compile(loss=keras.losses.categorical_crossentropy,
optimizer=keras.optimizers.Adam(),metrics=['accuracy'])
```

```
m_train_dropout = m_model.fit(train_X, train_label,
batch_size=batch_size,epochs=epochs,verbose=1,validation_data=(valid_X,
valid_label))
```

```
Train on 48000 samples, validate on 12000 samples
Epoch 1/1
48000/48000 [==============================] - 49s 1ms/step - loss: 0.2479
- accuracy: 0.9265 - val_loss: 0.1026 - val_accuracy: 0.9700
```

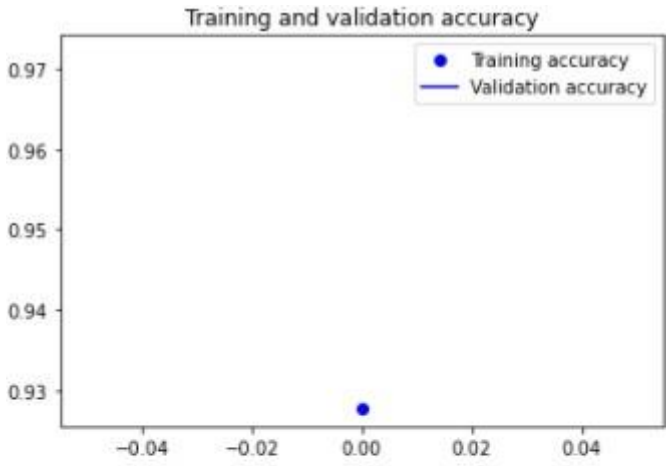
```
m_model.save("fashion_model_dropout.h5py")
test_eval = m_model.evaluate(test_X, test_Y_one_hot, verbose=1)
```

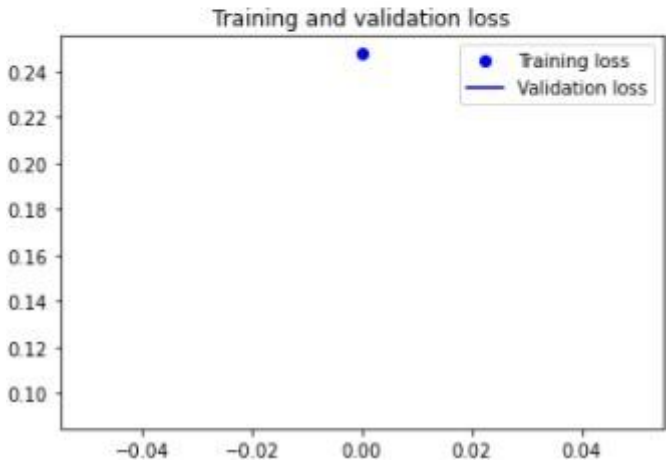
```
10000/10000 [==============================] - 3s 263us/step
```

```
print('Test loss:', test_eval[0])
print('Test accuracy:', test_eval[1])
```

```
Test loss: 0.08918832793608308
Test accuracy: 0.9713000059127808
```

```
accuracy = m_train_dropout.history['accuracy']
val_accuracy = m_train_dropout.history['val_accuracy']
loss = m_train_dropout.history['loss']
val_loss = m_train_dropout.history['val_loss']
epochs =range(len(accuracy))
plt.plot(epochs, accuracy, 'bo', label='Training accuracy')
plt.plot(epochs, val_accuracy, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```



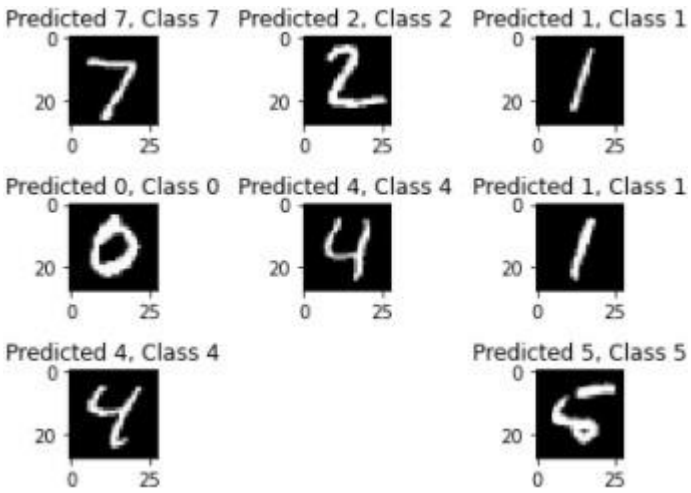

```
predicted_classes = m_model.predict(test_X)
```

```
predicted_classes = np.argmax(np.round(predicted_classes),axis=1)
predicted_classes.shape, test_Y.shape
```

```
((10000,), (10000,))
```

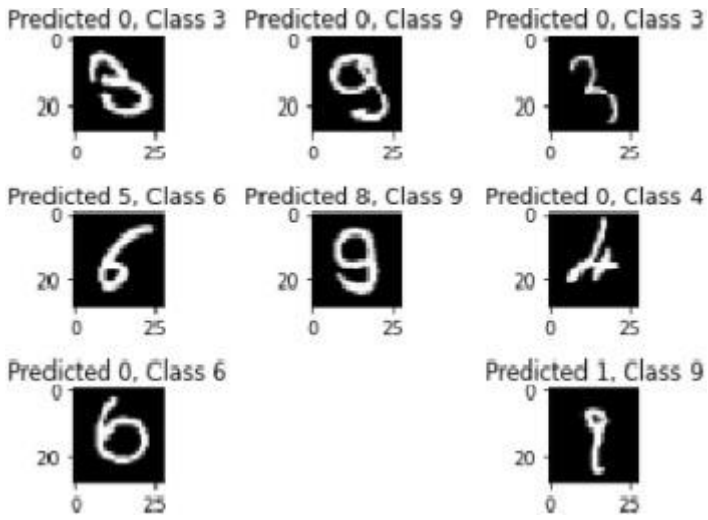
```
correct = np.where(predicted_classes==test_Y)[0]
print ("Found %d correct labels"%len(correct))
for i, correct inenumerate(correct[:9]):
    plt.subplot(3,3,i+1)
    plt.imshow(test_X[correct].reshape(28,28), cmap='gray',
interpolation='none')
    plt.title("Predicted {}, Class {}".format(predicted_classes[correct],
test_Y[correct]))
    plt.tight_layout()
```

```
Found 9680 correct labels
```




```
incorrect = np.where(predicted_classes!=test_Y)[0]
print ("Found %d incorrect labels"%len(incorrect))
for i, incorrect in enumerate(incorrect[:9]):
    plt.subplot(3,3,i+1)
    plt.imshow(test_X[incorrect].reshape(28,28), cmap='gray',
interpolation='none')
    plt.title("Predicted {}, Class {}".format(predicted_classes[incorrect],
test_Y[incorrect]))
    plt.tight_layout()
```

```
Found 320 incorrect labels
```




```
from sklearn.metrics import classification_report
target_names = ["Class {}".format(i) for i inrange(num_classes)]
print(classification_report(test_Y, predicted_classes,
target_names=target_names))
```

```
              precision    recall  f1-score   support

     Class 0       0.90      0.99      0.94       980
     Class 1       0.98      0.99      0.99      1135
     Class 2       0.99      0.94      0.96      1032
     Class 3       0.97      0.99      0.98      1010
     Class 4       0.98      0.98      0.98       982
     Class 5       1.00      0.93      0.96       892
     Class 6       0.97      0.98      0.98       958
     Class 7       0.95      0.98      0.97      1028
     Class 8       0.97      0.95      0.96       974
     Class 9       0.99      0.94      0.96      1009

    accuracy                           0.97     10000
   macro avg       0.97      0.97      0.97     10000
weighted avg       0.97      0.97      0.97     10000
```

**Conclusion: Using Keras and tensor flow network loaded the mnist image dataset and designed a two-layer neural network with one hidden layer and one output layer using CNN with Leaky Relu activation function for the hidden layer.**

## Program 8:

### Using Keras and tensor flow network

**i) Load the imdb text dataset**
**ii) Design a two-layer neural network with one hidden layer and one output layer**
**a. Use simple RNN in the hidden layer**
**b. Use sigmoid activation function for the output layer**
**iii) Train the designed network for imdb dataset**
**iv) Visualize the results of**
**a) Training vs validation accuracy**
**b) Training vs Validation loss**

```
from keras.models import Sequential
from keras.layers import Embedding, SimpleRNN
from keras.datasets import imdb
from keras.preprocessing import sequence
from keras.layers import Dense
max_features =10000
maxlen =500
batch_size =32

print('Loading data...')
(input_train, y_train), (input_test, y_test) = imdb.load_data(
num_words=max_features)
#(input_train, y_train), (input_test, y_test) = imdb.load_data()
print(len(input_train), 'train sequences')
print(len(input_test), 'test sequences')
print('Pad sequences (samples x time)')
input_train = sequence.pad_sequences(input_train, maxlen=maxlen)
input_test = sequence.pad_sequences(input_test, maxlen=maxlen)
```

```
print('input_train shape:', input_train.shape)
print('input_test shape:', input_test.shape)
```

```
25000 train sequences
25000 test sequences
Pad sequences (samples x time)
input_train shape: (25000, 500)
input_test shape: (25000, 500)
```

```
model = Sequential()
model.add(Embedding(max_features, 32)) #max_feature=10,000 so, 320,000
model.add(SimpleRNN(32))               #(32+32+1)*32=2080
model.add(Dense(1, activation='sigmoid'))#(32+1)*1=33
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, None, 32)          320000
_________________________________________________________________
simple_rnn_2 (SimpleRNN)     (None, 32)                2080
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 33
=================================================================
Total params: 322,113
Trainable params: 322,113
Non-trainable params: 0
_________________________________________________________________
```

```
model.compile(optimizer='rmsprop',
loss='binary_crossentropy',metrics=['acc'])
history = model.fit(input_train, y_train,epochs=10, batch_size=128,
validation_split=0.2)
```

```
Train on 20000 samples, validate on 5000 samples
Epoch 1/10
20000/20000 [==============================] - 33s 2ms/step - loss: 0.5955
- acc: 0.6679 - val_loss: 0.5106 - val_acc: 0.7566
Epoch 2/10
20000/20000 [==============================] - 36s 2ms/step - loss: 0.3544
- acc: 0.8530 - val_loss: 0.4272 - val_acc: 0.8158
Epoch 3/10
20000/20000 [==============================] - 37s 2ms/step - loss: 0.2823
- acc: 0.8870 - val_loss: 0.3698 - val_acc: 0.8652
Epoch 4/10
20000/20000 [==============================] - 41s 2ms/step - loss: 0.2192
- acc: 0.9174 - val_loss: 0.4816 - val_acc: 0.7870
Epoch 5/10
20000/20000 [==============================] - 36s 2ms/step - loss: 0.1675
- acc: 0.9376 - val_loss: 0.4021 - val_acc: 0.8440
Epoch 6/10
20000/20000 [==============================] - 32s 2ms/step - loss: 0.1261
- acc: 0.9570 - val_loss: 0.4502 - val_acc: 0.8312
Epoch 7/10
20000/20000 [==============================] - 32s 2ms/step - loss: 0.0758
- acc: 0.9740 - val_loss: 0.4815 - val_acc: 0.8328
Epoch 8/10
```

```
20000/20000 [==============================] - 35s 2ms/step - loss: 0.0552
- acc: 0.9829 - val_loss: 0.5122 - val_acc: 0.8474
Epoch 9/10
20000/20000 [==============================] - 33s 2ms/step - loss: 0.0313
- acc: 0.9908 - val_loss: 0.5852 - val_acc: 0.8282
Epoch 10/10
20000/20000 [==============================] - 32s 2ms/step - loss: 0.0239
- acc: 0.9933 - val_loss: 0.6137 - val_acc: 0.8376
```

```
predicted_classes = model.predict(input_test)
```

```
import numpy as np
predicted_classes = np.argmax(np.round(predicted_classes),axis=1)
```

```
predicted_classes.shape, y_test.shape
```

```
((25000,), (25000,))
```

```
correct = np.where(predicted_classes==y_test)[0]
print ("Found %d correct labels"%len(correct))
```

```
Found 12500 correct labels
```

```
incorrect = np.where(predicted_classes!=y_test)[0]
print ("Found %d incorrect labels"%len(incorrect))
```

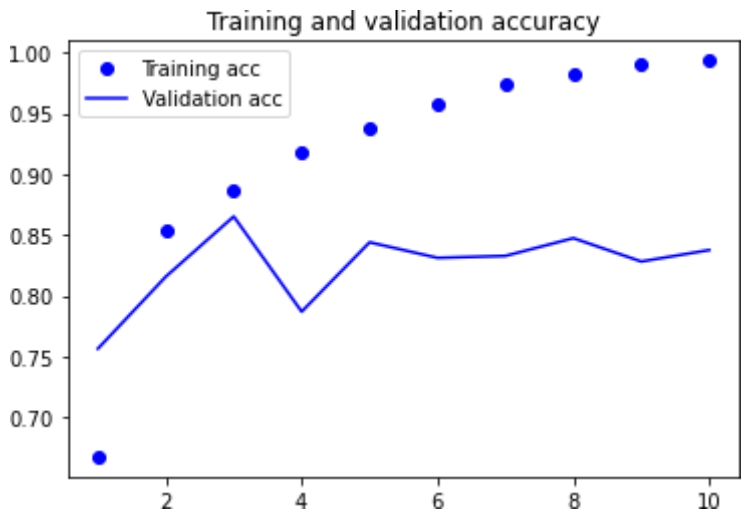
```
Found 12500 incorrect labels
```

```
from sklearn.metrics import classification_report
num_classes=2
target_names = ["Class {}".format(i) for i inrange(num_classes)]
print(classification_report(y_test, predicted_classes,
target_names=target_names))
```

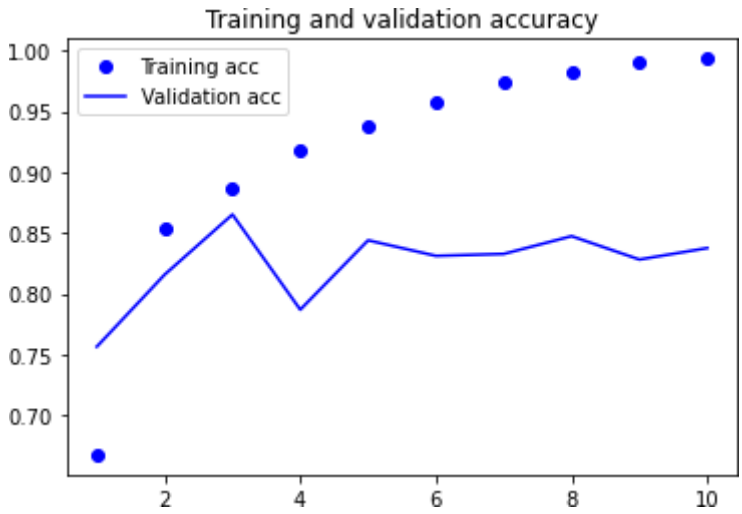
```
              precision    recall  f1-score   support

     Class 0       0.50      1.00      0.67     12500
     Class 1       0.00      0.00      0.00     12500

    accuracy                           0.50     25000
   macro avg       0.25      0.50      0.33     25000
weighted avg       0.25      0.50      0.33     25000
_warn_prf(average, modifier, msg_start, len(result))
```

```
import matplotlib.pyplot as plt
acc = history.history['acc']
val_acc = history.history['val_acc']
epochs =range(1, len(acc) +1)
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x22133e2fd08>
```




```
plt.figure()
loss =
history.history['loss
'] val_loss =
history.history['val_
loss'] epochs
=range(1, len(acc)
+1)
plt.plot(epochs, loss, 'bo', label='Training
loss') plt.plot(epochs, val_loss, 'b',
label='Validation loss') plt.title('Training and
validation loss')
pl
t.
le
ge
nd
()
pl
t.
sh
ow
()
.
```




**Conclusion: Using Keras and tensor flow network loaded the imdb text dataset and designed a two-layer neural network with one hidden layer and one output layer using simple RNN in the hidden layer.**

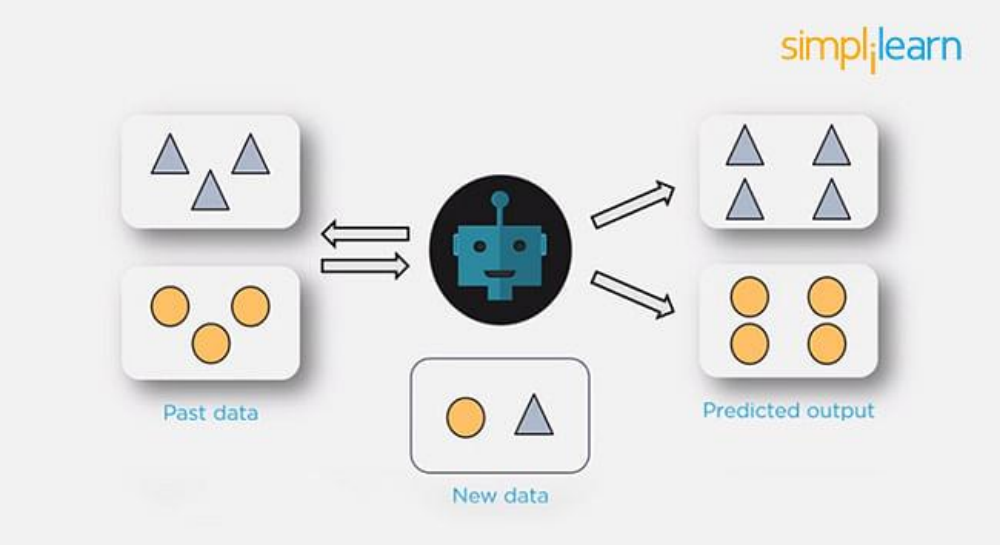
# Viva Questions

1. What Are the Different Types of Machine Learning?

There are three types of machine learning:

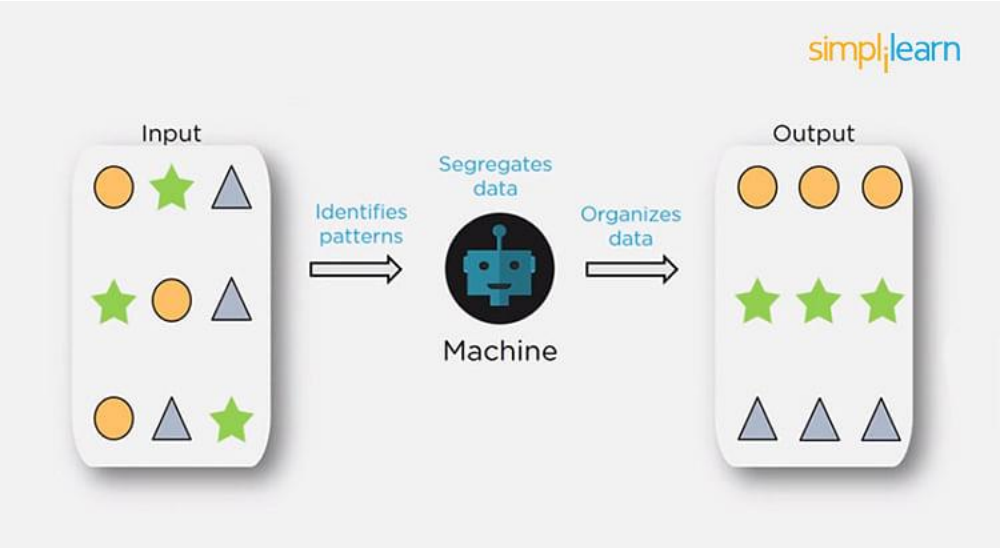
Supervised Learning

In supervised machine learning, a model makes predictions or decisions based on past or labeled data. Labeled data refers to sets of data that are given tags or labels, and thus made more meaningful.



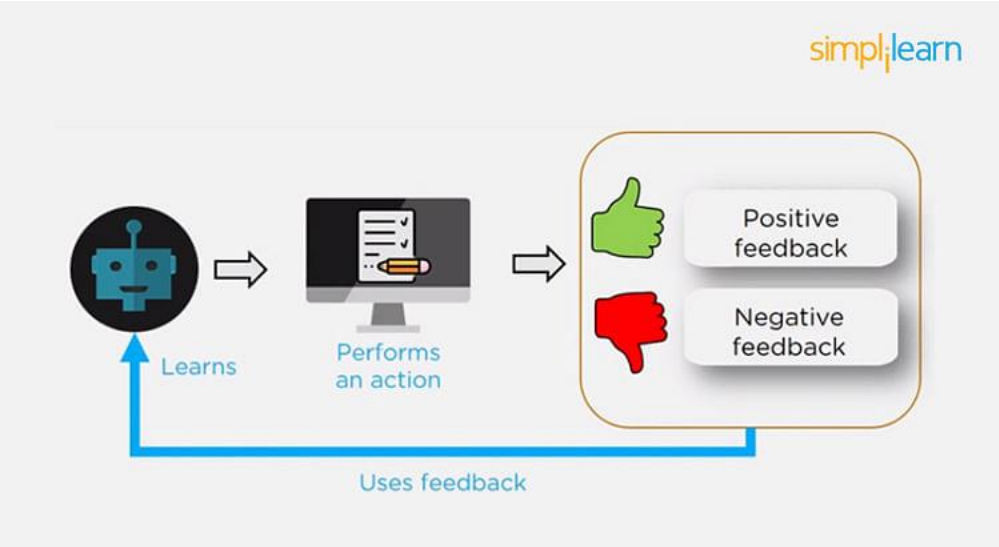

Unsupervised Learning

In unsupervised learning, we don't have labeled data. A model can identify patterns, anomalies, and relationships in the input data.




Reinforcement Learning

Using reinforcement learning, the model can learn based on the rewards it received for its previous action.




Consider an environment where an agent is working. The agent is given a target to achieve. Every time the agent takes some action toward the target, it is given positive feedback. And, if the action taken is going away from the goal, the agent is given negative feedback.

2. What is Overfitting, and How Can You Avoid It?

The Overfitting is a situation that occurs when a model learns the training set too well, taking up random fluctuations in the training data as concepts. These impact the model's ability to generalize and don't apply to new data.

When a model is given the training data, it shows 100 percent accuracy—technically a slight loss. But, when we use the test data, there may be an error and low efficiency. This condition is known as overfitting. There are multiple ways of avoiding overfitting, such as:

- Regularization. It involves a cost term for the features involved with the objective function
- Making a simple model. With lesser variables and parameters, the variance can be reduced
- Cross-validation methods like k-folds can also be used
- If some model parameters are likely to cause overfitting, techniques for regularization like LASSO can be used that penalize these parameters

3. What is 'training Set' and 'test Set' in a Machine Learning Model? How Much Data Will You Allocate for Your Training, Validation, and Test Sets?

There is a three-step process followed to create a model:

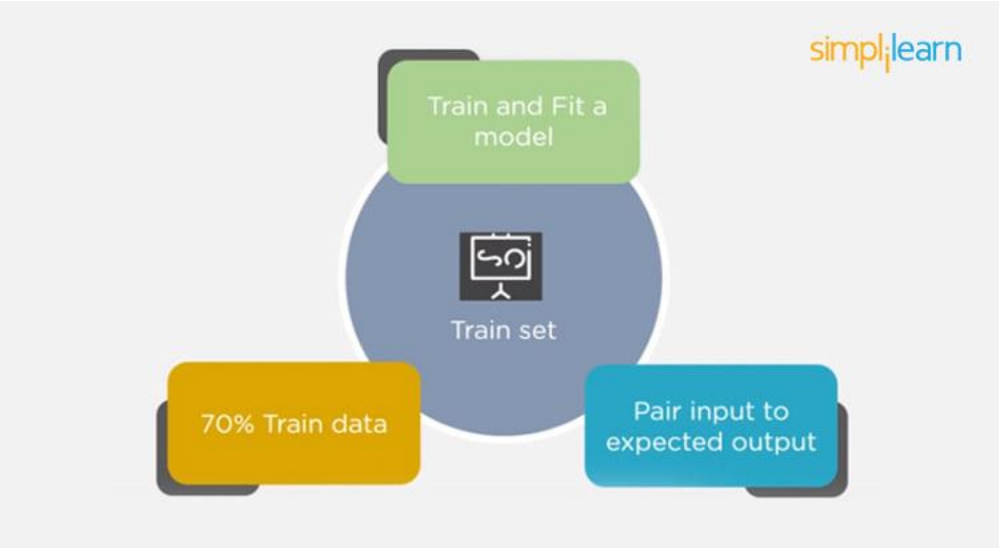
1. Train the model
2. Test the model
3. Deploy the model

| Training Set | Test Set |
|---|---|

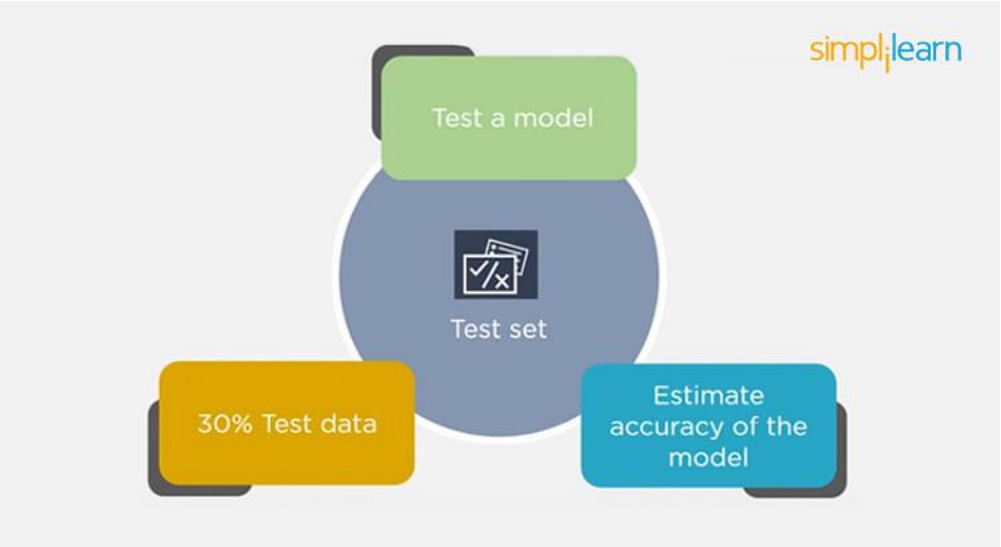
| | |
|---|---|
| • The training set is examples given to the model to analyze and learn<br>• 70% of the total data is typically taken as the training dataset<br>• This is labeled data used to train the model | • The test set is used to test the accuracy of the hypothesis generated by the model<br>• Remaining 30% is taken as testing dataset<br>• We test without labeled data and then verify results with labels |

Consider a case where you have labeled data for 1,000 records. One way to train the model is to expose all 1,000 records during the training process. Then you take a small set of the same data to test the model, which would give good results in this case.

But, this is not an accurate way of testing. So, we set aside a portion of that data called the 'test set' before starting the training process. The remaining data is called the 'training set' that we use for training the model. The training set passes through the model multiple times until the accuracy is high, and errors are minimized.




Now, we pass the test data to check if the model can accurately predict the values and determine if training is effective. If you get errors, you either need to change your model or retrain it with more data.



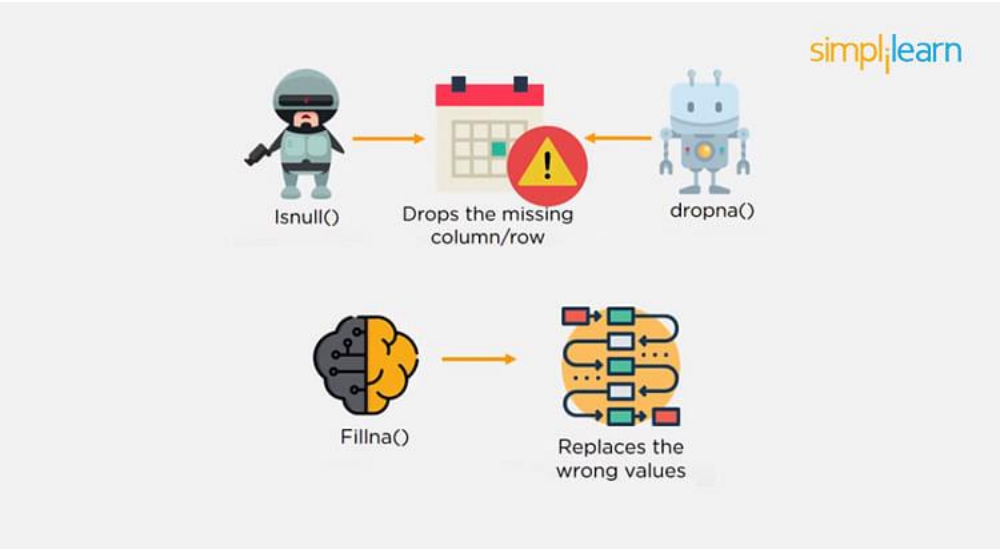

Regarding the question of how to split the data into a training set and test set, there is no fixed rule, and the ratio can vary based on individual preferences.

4. How Do You Handle Missing or Corrupted Data in a Dataset?

One of the easiest ways to handle missing or corrupted data is to drop those rows or columns or replace them entirely with some other value.

There are two useful methods in Pandas:

- IsNull() and dropna() will help to find the columns/rows with missing data and drop them
- Fillna() will replace the wrong values with a placeholder value




5. How Can You Choose a Classifier Based on a Training Set Data Size?

When the training set is small, a model that has a right bias and low variance seems to work better because they are less likely to overfit.

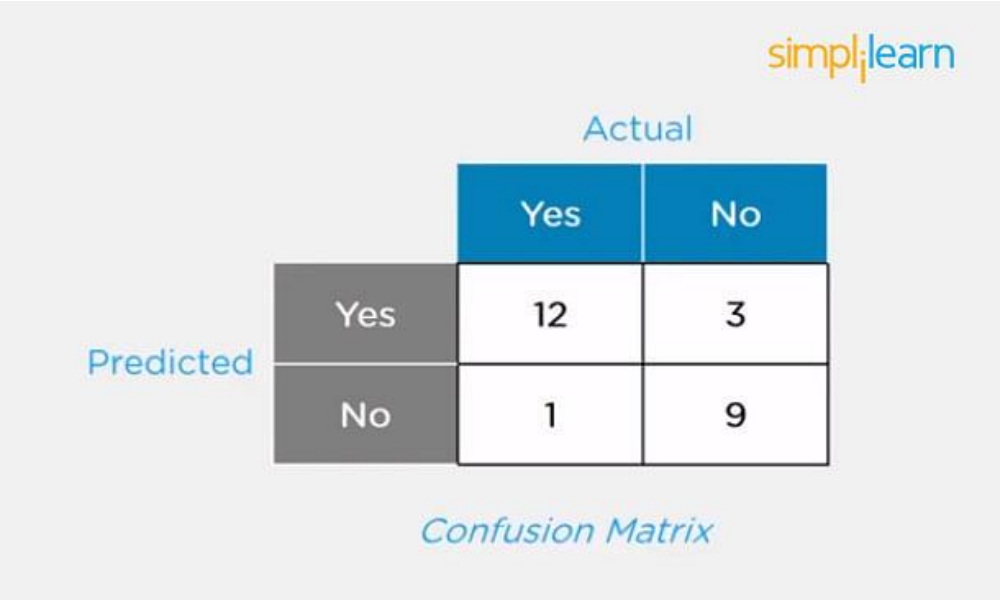
For example, Naive Bayes works best when the training set is large. Models with low bias and high variance tend to perform better as they work fine with complex relationships.

6. Explain the Confusion Matrix with Respect to Machine Learning Algorithms.

A confusion matrix (or error matrix) is a specific table that is used to measure the performance of an algorithm. It is mostly used in supervised learning; in unsupervised learning, it's called the matching matrix. The confusion matrix has two parameters:

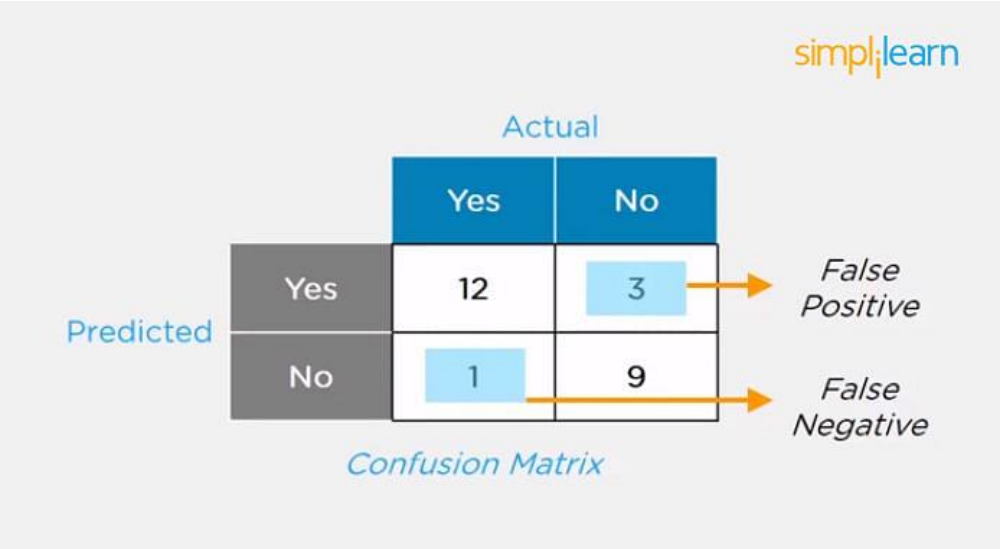
- Actual
- Predicted

It also has identical sets of features in both of these dimensions.
Consider a confusion matrix (binary matrix) shown below:




Here,
For actual values:
Total Yes = 12+1 = 13
Total No = 3+9 = 12
Similarly, for predicted values:
Total Yes = 12+3 = 15
Total No = 1+9 = 10
For a model to be accurate, the values across the diagonals should be high. The total sum of all the values in the matrix equals the total observations in the test data set.
For the above matrix, total observations = 12+3+1+9 = 25
Now, accuracy = sum of the values across the diagonal/total dataset
= (12+9) / 25
= 21 / 25
= 84%

7. What Is a False Positive and False Negative and How Are They Significant?
False positives are those cases that wrongly get classified as True but are False.
False negatives are those cases that wrongly get classified as False but are True.
In the term 'False Positive,' the word 'Positive' refers to the 'Yes' row of the predicted value in the confusion matrix. The complete term indicates that the system has predicted it as a positive, but the actual value is negative.




So, looking at the confusion matrix, we get:
False-positive = 3
True positive = 12
Similarly, in the term 'False Negative,' the word 'Negative' refers to the 'No' row of the predicted value in the confusion matrix. And the complete term indicates that the system has predicted it as negative, but the actual value is positive.
So, looking at the confusion matrix, we get:
False Negative = 1
True Negative = 9

8. What Are the Three Stages of Building a Model in Machine Learning?
The three stages of building a machine learning model are:

- Model Building

Choose a suitable algorithm for the model and train it according to the requirement

- Model Testing

Check the accuracy of the model through the test data

- Applying the Model

Make the required changes after testing and use the final model for real-time projects
Here, it's important to remember that once in a while, the model needs to be checked to make sure it's working correctly. It should be modified to make sure that it is up-to-date.

9. What is Deep Learning?
The Deep learning is a subset of machine learning that involves systems that think and learn like humans using artificial neural networks. The term 'deep' comes from the fact that you can have several layers of neural networks.
One of the primary differences between machine learning and deep learning is that feature engineering is done manually in machine learning. In the case of deep learning, the model consisting of neural networks will automatically determine which features to use (and which not to use).

10. What Are the Differences Between Machine Learning and Deep Learning?

| Machine Learning | Deep Learning |
|---|---|
| • Enables machines to take decisions on their own, based on past data<br>• It needs only a small amount of data for training<br>• Works well on the low-end system, so you don't need large machines<br>• Most features need to be identified in advance and manually coded<br>• The problem is divided into two parts and solved individually and then combined | • Enables machines to take decisions with the help of artificial neural networks<br>• It needs a large amount of training data<br>• Needs high-end machines because it requires a lot of computing power<br>• The machine learns the features from the data it is provided<br>• The problem is solved in an end-to-end manner |

11. What Are the Applications of Supervised Machine Learning in Modern Businesses?

Applications of supervised machine learning include:

- Email Spam Detection

Here we train the model using historical data that consists of emails categorized as spam or not spam. This labeled information is fed as input to the model.

- Healthcare Diagnosis

By providing images regarding a disease, a model can be trained to detect if a person is suffering from the disease or not.

- Sentiment Analysis

This refers to the process of using algorithms to mine documents and determine whether they’re positive, neutral, or negative in sentiment.

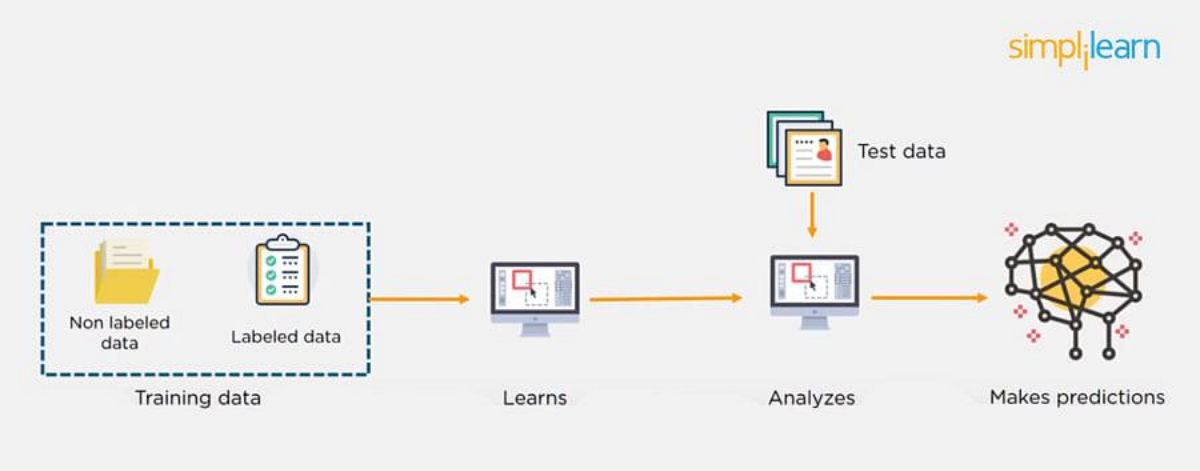
- Fraud Detection

By training the model to identify suspicious patterns, we can detect instances of possible fraud.

12. What is Semi-supervised Machine Learning?

Supervised learning uses data that is completely labeled, whereas unsupervised learning uses no training data.

In the case of semi-supervised learning, the training data contains a small amount of labeled data and a large amount of unlabeled data.

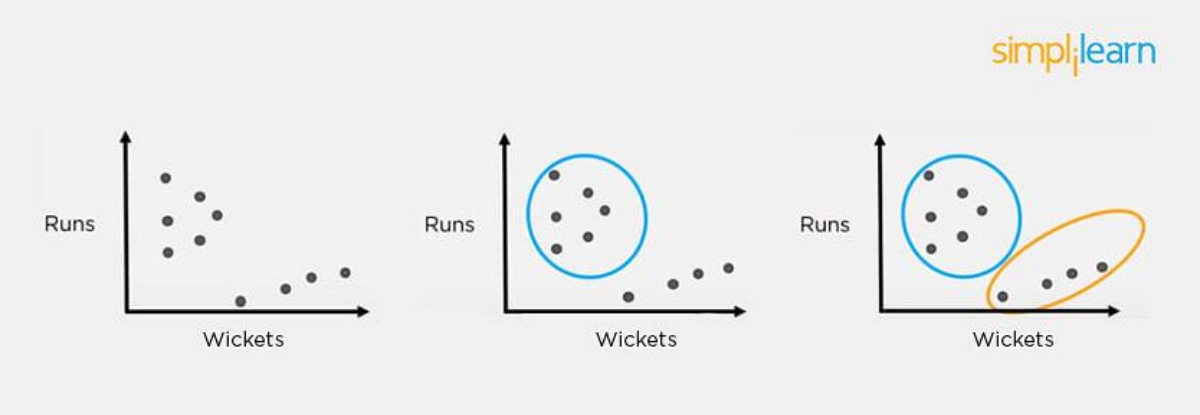



13. What Are Unsupervised Machine Learning Techniques?

There are two techniques used in unsupervised learning: clustering and association.

Clustering

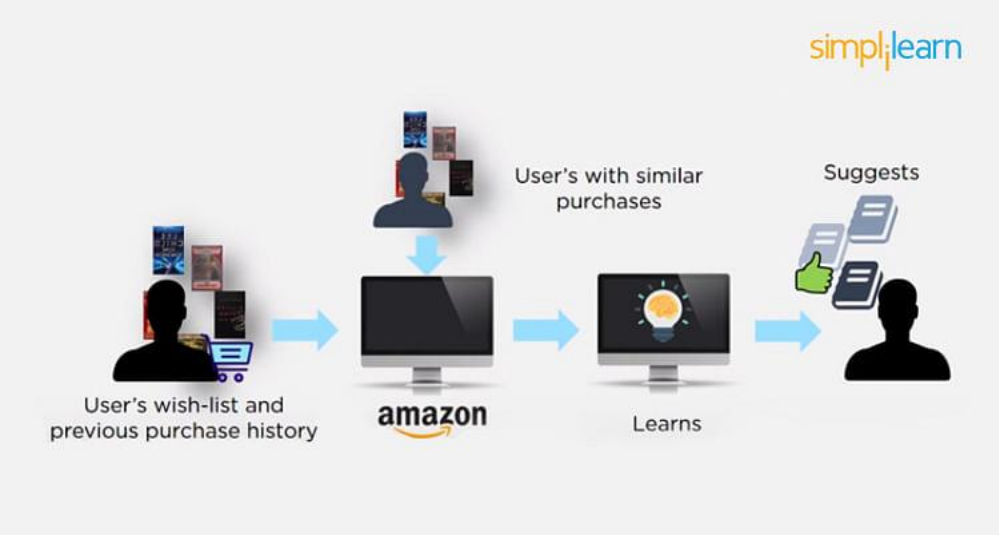
Clustering problems involve data to be divided into subsets. These subsets, also called clusters, contain data that are similar to each other. Different clusters reveal different details about the objects, unlike classification or regression.




Association

In an association problem, we identify patterns of associations between different variables or items.

For example, an e-commerce website can suggest other items for you to buy, based on the prior purchases that you have made, spending habits, items in your wishlist, other customers' purchase habits, and so on.




14. What is the Difference Between Supervised and Unsupervised Machine Learning?

- Supervised learning - This model learns from the labeled data and makes a future prediction as output
- Unsupervised learning - This model uses unlabeled input data and allows the algorithm to act on that information without guidance.

15. What is the Difference Between Inductive Machine Learning and Deductive Machine Learning?

| Inductive Learning | Deductive Learning |
|---|---|
| • It observes instances based on defined principles to draw a conclusion<br>• Example: Explaining to a child to keep away from the fire by showing a video where fire causes damage | • It concludes experiences<br>• Example: Allow the child to play with fire. If he or she gets burned, they will learn that it is dangerous and will refrain from making the same mistake again |

16. Compare K-means and KNN Algorithms.

| K-means | KNN |
|---|---|
| • K-Means is unsupervised<br>• K-Means is a clustering algorithm<br>• The points in each cluster are similar to each other, and each cluster is different from its neighboring clusters | • KNN is supervised in nature<br>• KNN is a classification algorithm<br>• It classifies an unlabeled observation based on its K (can be any number) surrounding neighbors |

17. What Is 'naive' in the Naive Bayes Classifier?

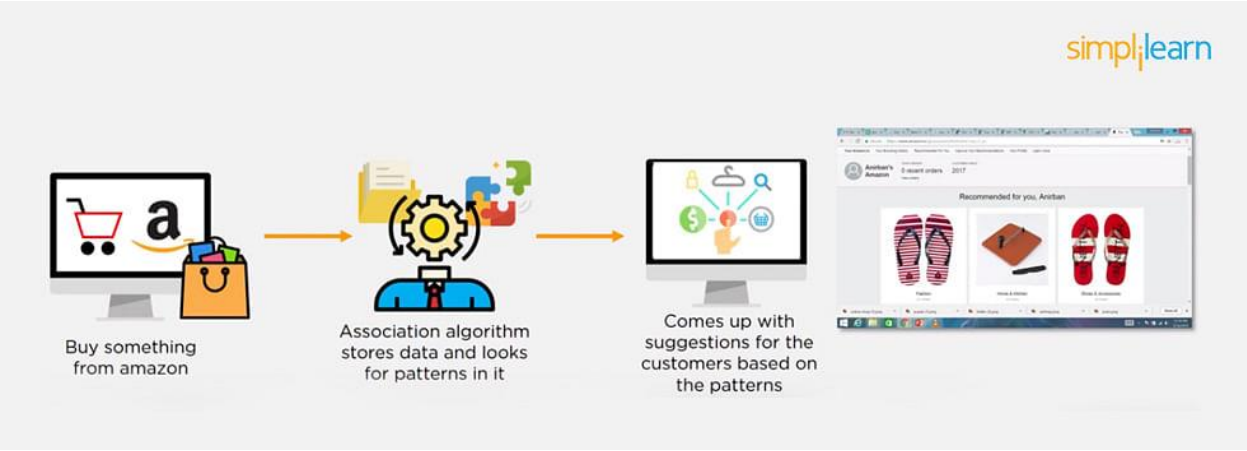
The classifier is called 'naive' because it makes assumptions that may or may not turn out to be correct. The algorithm assumes that the presence of one feature of a class is not related to the presence of any other feature (absolute independence of features), given the class variable.
For instance, a fruit may be considered to be a cherry if it is red in color and round in shape, regardless of other features. This assumption may or may not be right (as an apple also matches the description).
Learn AI and Grab the Best Opportunities Out There
With Purdue University's AI ProgramJoin The Program

18. Explain How a System Can Play a Game of Chess Using Reinforcement Learning.
Reinforcement learning has an environment and an agent. The agent performs some actions to achieve a specific goal. Every time the agent performs a task that is taking it towards the goal, it is rewarded. And, every time it takes a step that goes against that goal or in the reverse direction, it is penalized.
Earlier, chess programs had to determine the best moves after much research on numerous factors. Building a machine designed to play such games would require many rules to be specified.
With reinforced learning, we don't have to deal with this problem as the learning agent learns by playing the game. It will make a move (decision), check if it's the right move (feedback), and keep the outcomes in memory for the next step it takes (learning). There is a reward for every correct decision the system takes and punishment for the wrong one.

19. How Will You Know Which Machine Learning Algorithm to Choose for Your Classification Problem?
While there is no fixed rule to choose an algorithm for a classification problem, you can follow these guidelines:

- If accuracy is a concern, test different algorithms and cross-validate them
- If the training dataset is small, use models that have low variance and high bias
- If the training dataset is large, use models that have high variance and little bias

20. How is Amazon Able to Recommend Other Things to Buy? How Does the Recommendation Engine Work?
Once a user buys something from Amazon, Amazon stores that purchase data for future reference and finds products that are most likely also to be bought, it is possible because of the Association algorithm, which can identify patterns in a given dataset.



21. When Will You Use Classification over Regression?

Classification is used when your target is categorical, while regression is used when your target variable is continuous. Both classification and regression belong to the category of supervised machine learning algorithms.

Examples of classification problems include:

- Predicting yes or no
- Estimating gender
- Breed of an animal
- Type of color

Examples of regression problems include:

- Estimating sales and price of a product
- Predicting the score of a team
- Predicting the amount of rainfall

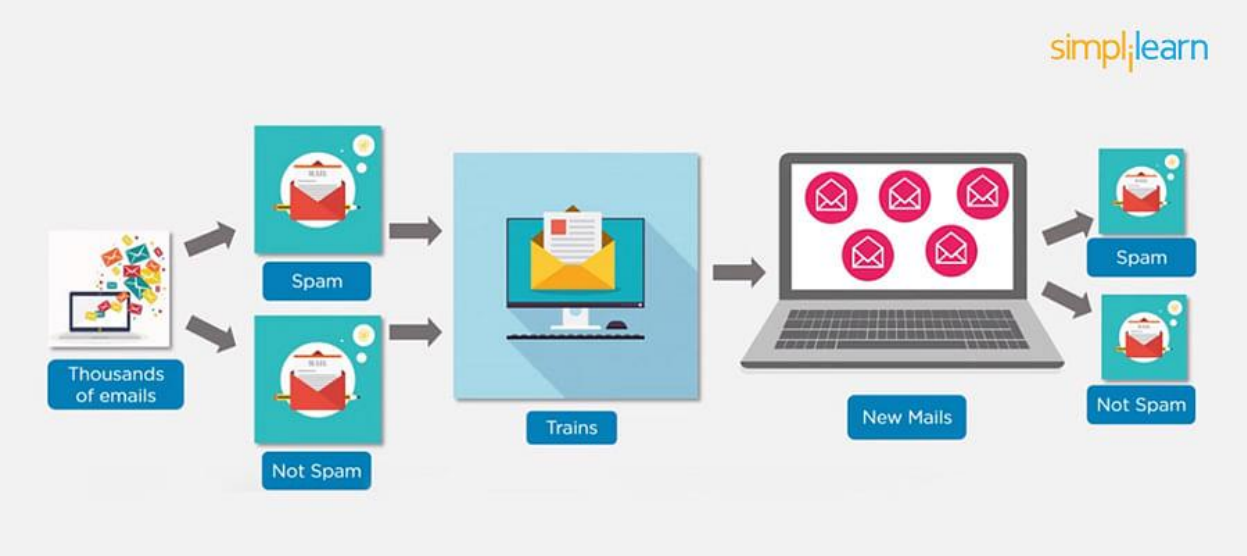
Grab the Highest Paying Machine Learning Jobs

With PCP in Generative AI and Machine LearningExplore Program

22. How Do You Design an Email Spam Filter?

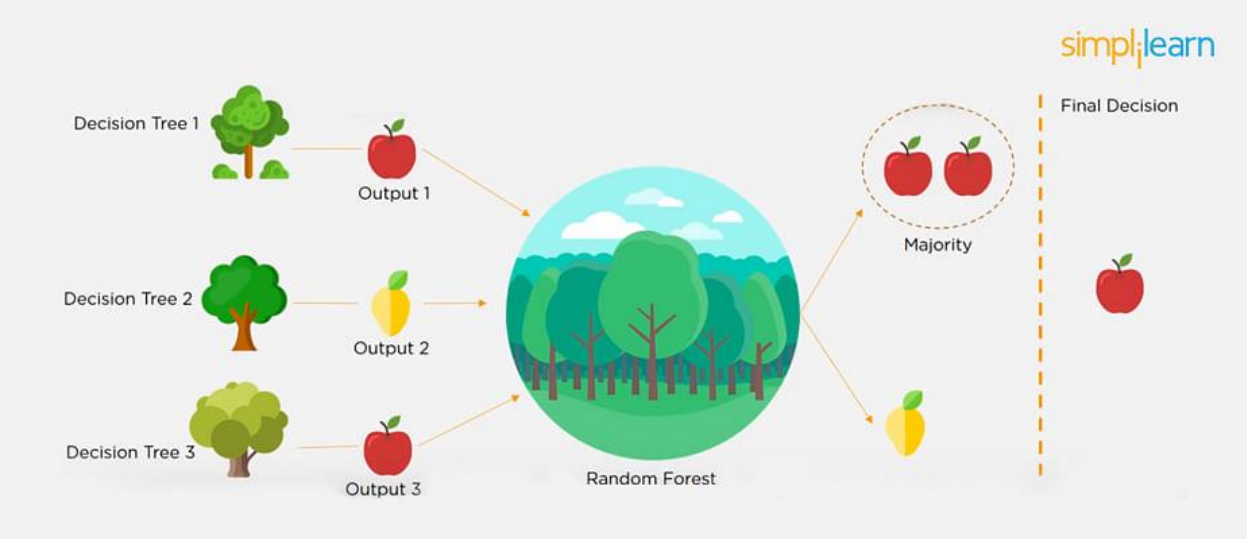
Building a spam filter involves the following process:

- The email spam filter will be fed with thousands of emails
- Each of these emails already has a label: 'spam' or 'not spam.'
- The supervised machine learning algorithm will then determine which type of emails are being marked as spam based on spam words like the lottery, free offer, no money, full refund, etc.
- The next time an email is about to hit your inbox, the spam filter will use statistical analysis and algorithms like Decision Trees and SVM to determine how likely the email is spam
- If the likelihood is high, it will label it as spam, and the email won't hit your inbox
- Based on the accuracy of each model, we will use the algorithm with the highest accuracy after testing all the models




23. What is a Random Forest?

A ‘random forest’ is a supervised machine learning algorithm that is generally used for classification problems. It operates by constructing multiple decision trees during the training phase. The random forest chooses the decision of the majority of the trees as the final decision.

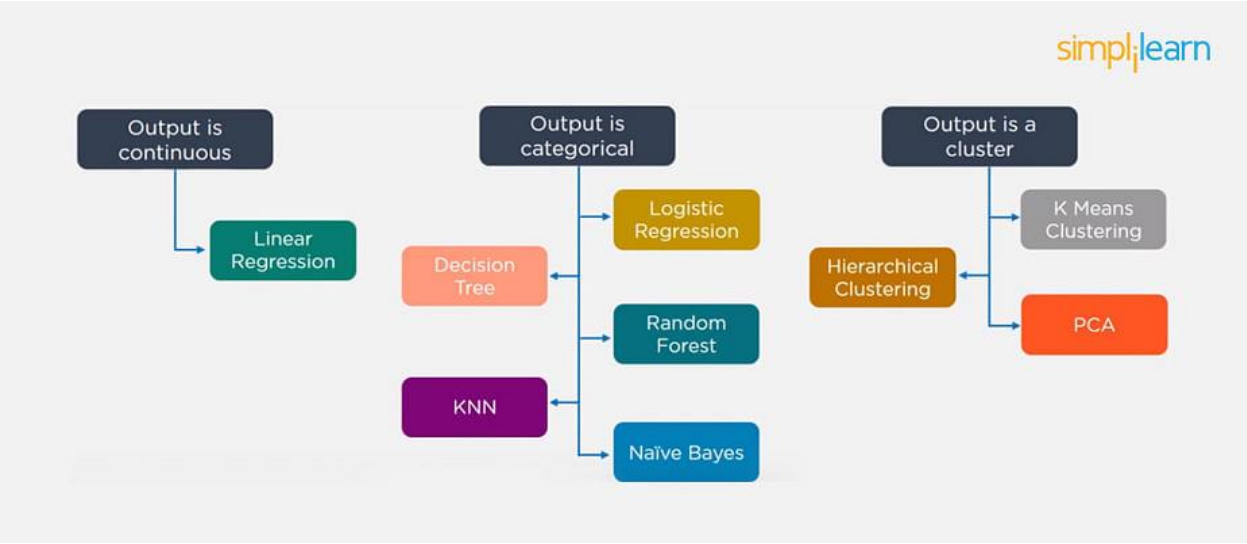


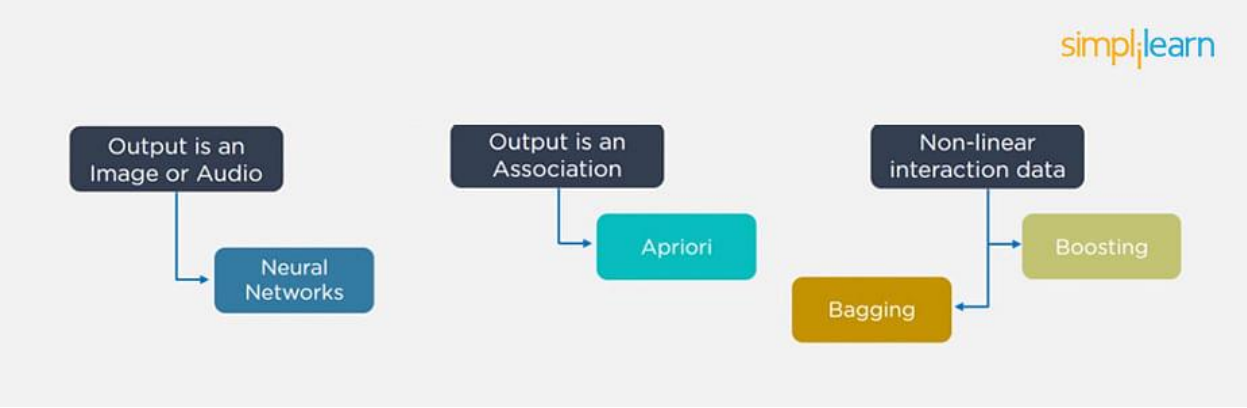

24. Considering a Long List of Machine Learning Algorithms, given a Data Set, How Do You Decide Which One to Use?

There is no master algorithm for all situations. Choosing an algorithm depends on the following questions:

- How much data do you have, and is it continuous or categorical?
- Is the problem related to classification, association, clustering, or regression?
- Predefined variables (labeled), unlabeled, or mix?
- What is the goal?

Based on the above questions, the following algorithms can be used:






25. What is Bias and Variance in a Machine Learning Model?

Bias

Bias in a machine learning model occurs when the predicted values are further from the actual values. Low bias indicates a model where the prediction values are very close to the actual ones.

Underfitting: High bias can cause an algorithm to miss the relevant relations between features and target outputs.

Variance

Variance refers to the amount the target model will change when trained with different training data. For a good model, the variance should be minimized.

Overfitting: High variance can cause an algorithm to model the random noise in the training data rather than the intended outputs.

26. What is the Trade-off Between Bias and Variance?

The bias-variance decomposition essentially decomposes the learning error from any algorithm by adding the bias, variance, and a bit of irreducible error due to noise in the underlying dataset.

Necessarily, if you make the model more complex and add more variables, you'll lose bias but gain variance. To get the optimally-reduced amount of error, you'll have to trade off bias and variance. Neither high bias nor high variance is desired.

High bias and low variance algorithms train models that are consistent, but inaccurate on average.

High variance and low bias algorithms train models that are accurate but inconsistent.

27. Define Precision and Recall.

Precision

Precision is the ratio of several events you can correctly recall to the total number of events you recall (mix of correct and wrong recalls).

Precision = (True Positive) / (True Positive + False Positive)

Recall

A recall is the ratio of the number of events you can recall the number of total events.

Recall = (True Positive) / (True Positive + False Negative)

28. What is a Decision Tree Classification?

A decision tree builds classification (or regression) models as a tree structure, with datasets broken up into ever-smaller subsets while developing the decision tree, literally in a tree-like way with branches and nodes. Decision trees can handle both categorical and numerical data.

Become the Highest Paid AI Engineer!
With Our Trending AI Engineer Master ProgramKnow More

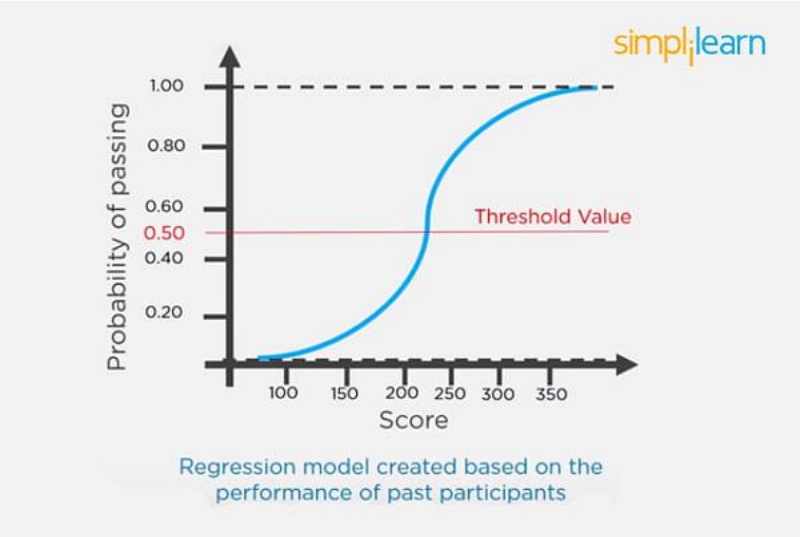
29. What is Pruning in Decision Trees, and How Is It Done?
Pruning is a technique in machine learning that reduces the size of decision trees. It reduces the complexity of the final classifier, and hence improves predictive accuracy by the reduction of overfitting.
Pruning can occur in:

- Top-down fashion. It will traverse nodes and trim subtrees starting at the root
- Bottom-up fashion. It will begin at the leaf nodes

There is a popular pruning algorithm called reduced error pruning, in which:

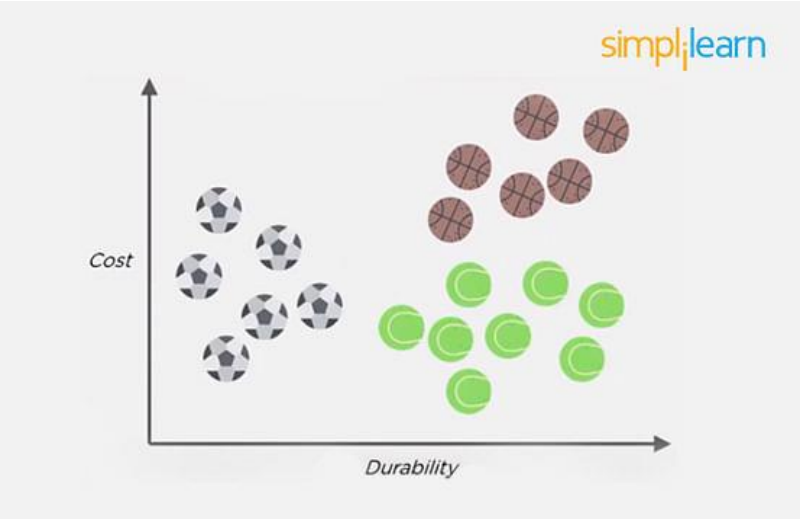
- Starting at the leaves, each node is replaced with its most popular class
- If the prediction accuracy is not affected, the change is kept
- There is an advantage of simplicity and speed

30. Briefly Explain Logistic Regression.
Logistic regression is a classification algorithm used to predict a binary outcome for a given set of independent variables.
The output of logistic regression is either a 0 or 1 with a threshold value of generally 0.5. Any value above 0.5 is considered as 1, and any point below 0.5 is considered as 0.

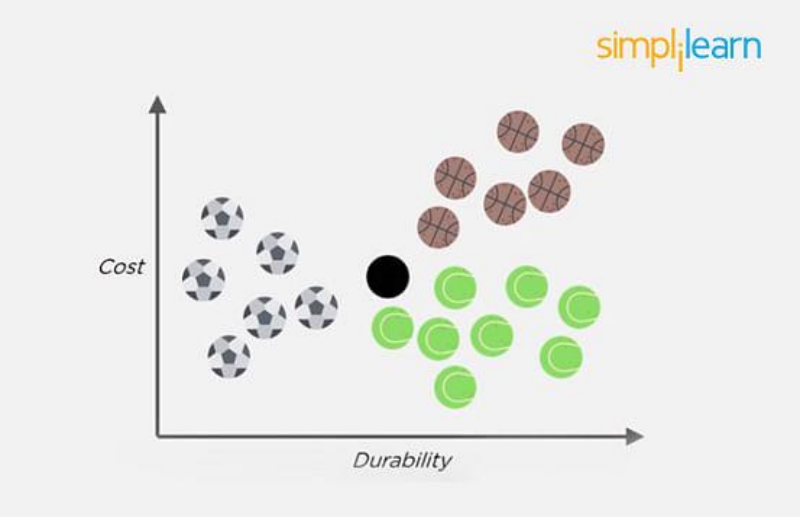



31. Explain the K Nearest Neighbor Algorithm.
K nearest neighbor algorithm is a classification algorithm that works in a way that a new data point is assigned to a neighboring group to which it is most similar.
In K nearest neighbors, K can be an integer greater than 1. So, for every new data point, we want to classify, we compute to which neighboring group it is closest.
Let us classify an object using the following example. Consider there are three clusters:

- Football
- Basketball
- Tennis ball




Let the new data point to be classified is a black ball. We use KNN to classify it. Assume K = 5 (initially). Next, we find the K (five) nearest data points, as shown.




Observe that all five selected points do not belong to the same cluster. There are three tennis balls and one each of basketball and football.
When multiple classes are involved, we prefer the majority. Here the majority is with the tennis ball, so the new data point is assigned to this cluster.

32. What is a Recommendation System?
Anyone who has used Spotify or shopped at Amazon will recognize a recommendation system: It's an information filtering system that predicts what a user might want to hear or see based on choice patterns provided by the user.

33. What is Kernel SVM?
Kernel SVM is the abbreviated version of the kernel support vector machine. Kernel methods are a class of algorithms for pattern analysis, and the most common one is the kernel SVM.

34. What Are Some Methods of Reducing Dimensionality?

You can reduce dimensionality by combining features with feature engineering, removing collinear features, or using algorithmic dimensionality reduction.
Now that you have gone through these machine learning interview questions, you must have got an idea of your strengths and weaknesses in this domain.

35. What is Principal Component Analysis?
Principal Component Analysis or PCA is a multivariate statistical technique that is used for analyzing quantitative data. The objective of PCA is to reduce higher dimensional data to lower dimensions, remove noise, and extract crucial information such as features and attributes from large amounts of data.

36. What do you understand by the F1 score?
The F1 score is a metric that combines both Precision and Recall. It is also the weighted average of precision and recall.
The F1 score can be calculated using the below formula:
F1 = 2 * (P * R) / (P + R)
The F1 score is one when both Precision and Recall scores are one.
Did You Know? 🔍
Machine Learning Engineers in the US earn an average annual salary of over $109,100, making it one of the most lucrative tech careers.

37. What do you understand by Type I vs Type II error?
Type I Error: Type I error occurs when the null hypothesis is true and we reject it.
Type II Error: Type II error occurs when the null hypothesis is false and we accept it.

| | | reality | |
|---|---|---|---|
| | | $H_0$ = True | $H_0$ = False |
| Conclusion | $H_0$ is not rejected | OK | Type II error |
| | $H_0$ is rejected | Type I error | OK |

38. Explain Correlation and Covariance?
Correlation: Correlation tells us how strongly two random variables are related to each other. It takes values between -1 to +1.
Formula to calculate Correlation:

$$\text{Correlation} = \frac{\text{Cov}(x, y)}{6x * 6y}$$

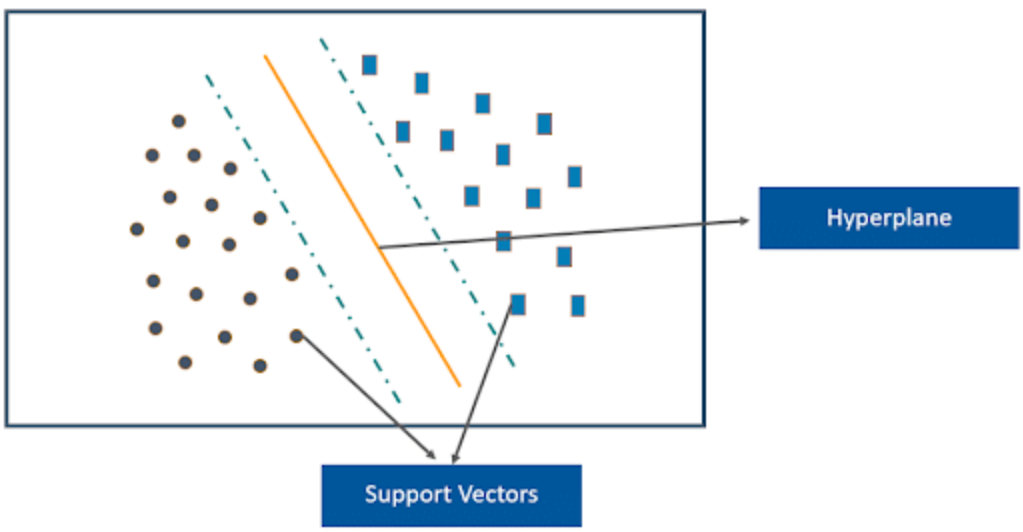
Covariance: Covariance tells us the direction of the linear relationship between two random variables. It can take any value between - ∞ and + ∞.

Formula to calculate Covariance:

$$\text{Cov}(x, y) = \frac{\sum (x_i - x') * (y_i - y')}{N}$$

39. What are Support Vectors in SVM?

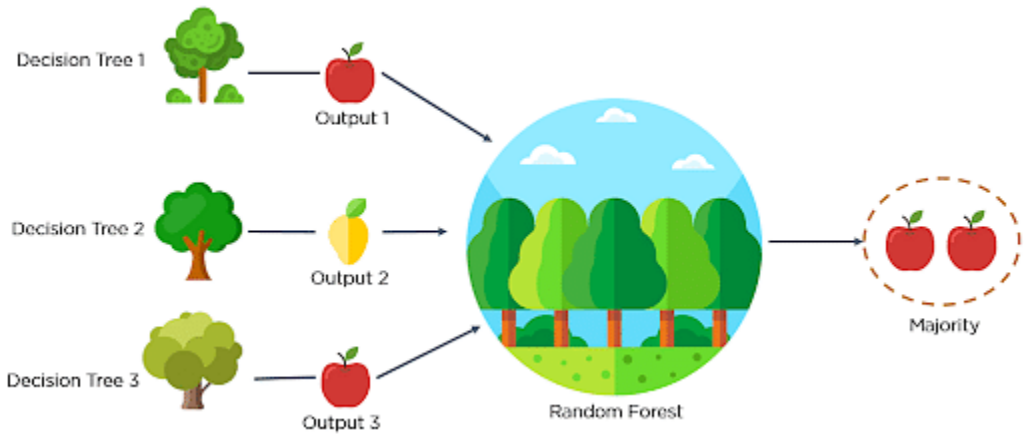
Support Vectors are data points that are nearest to the hyperplane. It influences the position and orientation of the hyperplane. Removing the support vectors will alter the position of the hyperplane. The support vectors help us build our support vector machine model.




40. What is Ensemble learning?

Ensemble learning is a combination of the results obtained from multiple machine learning models to increase the accuracy for improved decision-making.

Example: A Random Forest with 100 trees can provide much better results than using just one decision tree.




41. What is Cross-Validation?

Cross-Validation in Machine Learning is a statistical resampling technique that uses different parts of the dataset to train and test a machine learning algorithm on different iterations. The aim of cross-validation is to test the model's ability to predict a new set of data that was not used to train the model. Cross-validation avoids the overfitting of data.

K-Fold Cross Validation is the most popular resampling technique that divides the whole dataset into K sets of equal sizes.

Advance Your AI Engineering Career

With Microsoft's Latest AI ProgramSign Up Today

42. What are the different methods to split a tree in a decision tree algorithm?

Variance: Splitting the nodes of a decision tree using the variance is done when the target variable is continuous.

$$\text{Variance} = \frac{\sum (X-\bar{X})^2}{N}$$

Information Gain: Splitting the nodes of a decision tree using Information Gain is preferred when the target variable is categorical.

$$\text{IG} = 1 - \text{Entropy}$$

$$\text{Entropy} = -\sum p_i \log_2 p_i$$

Gini Impurity: Splitting the nodes of a decision tree using Gini Impurity is followed when the target variable is categorical.

$$I_G(n) = 1 - \sum_{i=1}^{n} (p_i)^2$$

43. How does the Support Vector Machine algorithm handle self-learning?

The SVM algorithm has a learning rate and expansion rate which takes care of self-learning. The learning rate compensates or penalizes the hyperplanes for making all the incorrect moves while the expansion rate handles finding the maximum separation area between different classes.

44. What are the assumptions you need to take before starting with linear regression?
There are primarily 5 assumptions for a Linear Regression model:

- Multivariate normality
- No auto-correlation
- Homoscedasticity
- Linear relationship
- No or little multicollinearity

45. What is the difference between Lasso and Ridge regression?
Lasso(also known as L1) and Ridge(also known as L2) regression are two popular regularization techniques that are used to avoid overfitting of data. These methods are used to penalize the coefficients to find the optimum solution and reduce complexity. The Lasso regression works by penalizing the sum of the absolute values of the coefficients. In Ridge or L2 regression, the penalty function is determined by the sum of the squares of the coefficients.