DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)

T.E/SEM VI/CBCGS/AIML
Academic Year: 2022-23

NAME	**CHIKANKAR PRATHAMESH SHIVAJI**
BRANCH	**CSE-(AI&ML)**
ROLL NO.	**11**
SUBJECT	**MACHINE LEARNING LAB**
COURSE CODE	**CSL604**
PRACTICAL NO.	**1**
DOP	**20/01/2023**
DOS	

STEPS TO INSTALL ANACONDA ON UBUNTU:

Step1 – Download and Install Anaconda Script

- Once you're logged into your VPS, refresh the APT command to synchronize all repositories via the command line:
  sudo apt-get update
- Download the newest Anaconda installer with the wget command. If you don't have it, install it first:
  apt-get install wget
- Download the Anaconda installer:
  wget https:// repo.anaconda.com/archive/Anaconda3-2022.05-Linux-x86_64.sh

Step 2 – Choose Installation Directory

After agreeing to the license terms, the following prompt will ask you to input the directory where to install the Anaconda on the Ubuntu system. The default location is the user's HOME directory on Ubuntu.

It is recommended to have Anaconda installed in this location. Therefore, press Enter to confirm the default location.

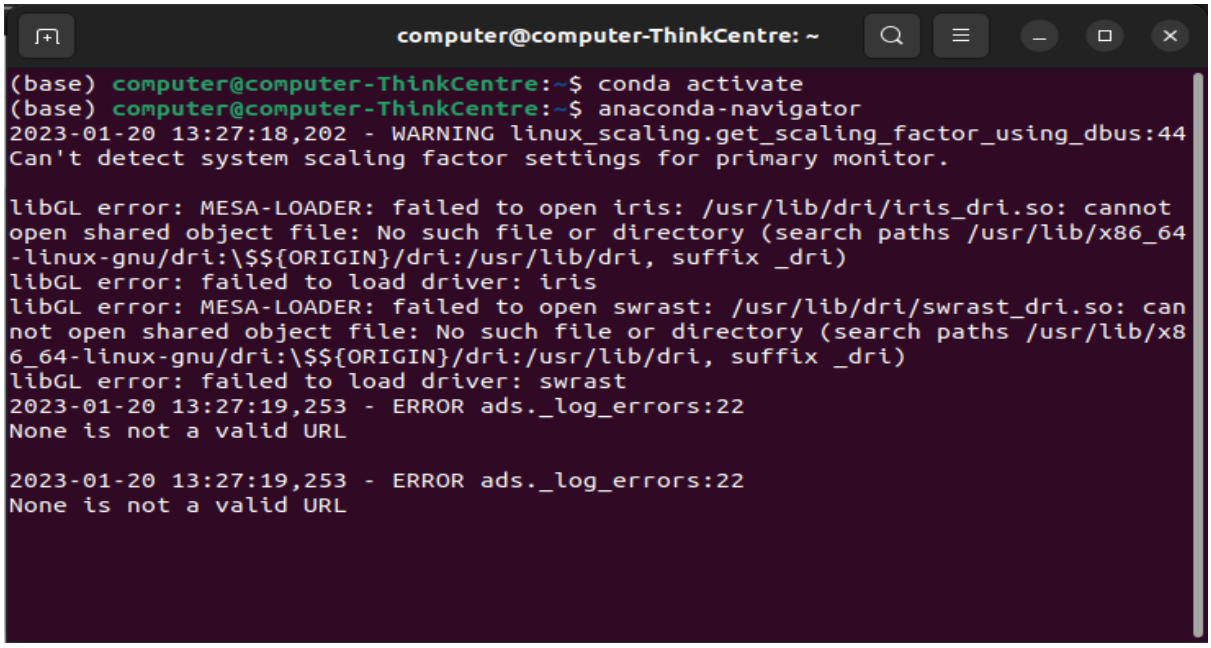
In the next prompt, you will see that the installation process has started. Wait a few minutes until the installer successfully completes the installation process. Type "yes" once more and press Enter.

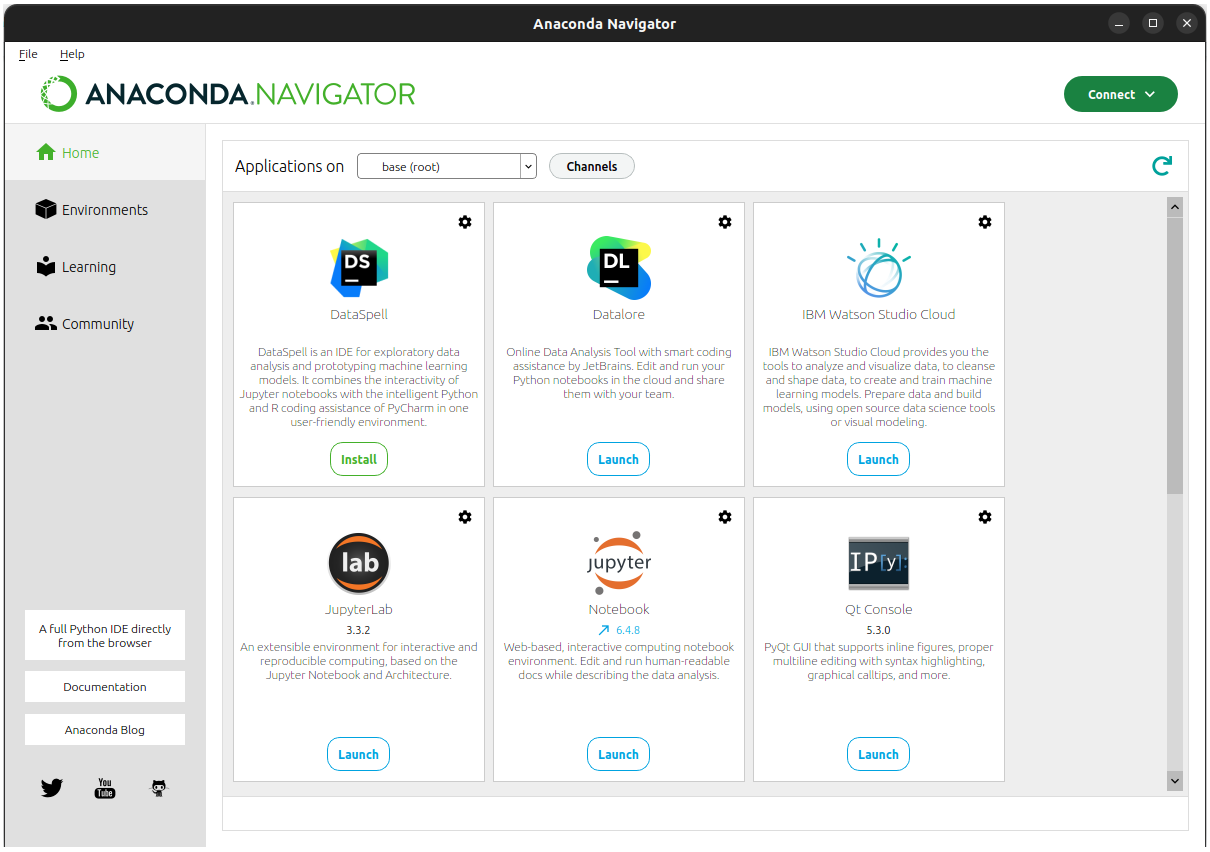
Congratulations, you have successfully installed Anaconda!

Step 3 – Activating Anaconda

conda activate

anaconda-navigator





FEATURES OF ANACONDA:

- **Compiled with Latest Python release:** Anaconda 5.3 is compiled with Python 3.7, taking advantage of Python's speed and feature improvements.

- **Better Reliability:** The reliability of Anaconda has been improved in the latest release by capturing and storing the package metadata for installed packages.

- **Enhanced CPU Performance:** The Intel Math Kernel Library 2019 for Deep Neural Networks (MKL 2019) has been introduced in Anaconda 5.3 distribution. Users deploying Tensor flow can make use of MKL 2019 for Deep Neural Networks. These Python binary packages are provided to achieve high CPU performance.

- **New packages are added:** There are over 230 packages which has been updated and added in the new release.

PROGRAM USING JUPYTER NOTEBOOK:

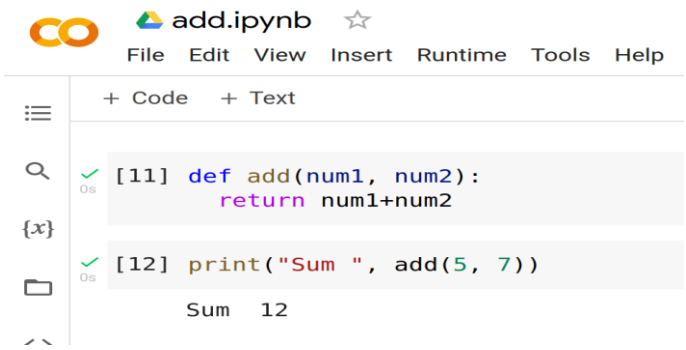
```
In [1]: num_1=int(input("enter first number: "))
        num_2=int(input("enter second number: "))
        print("your sum is : ",num_1+num_2)

        enter first number: 12
        enter second number: 34
        your sum is :  46
```

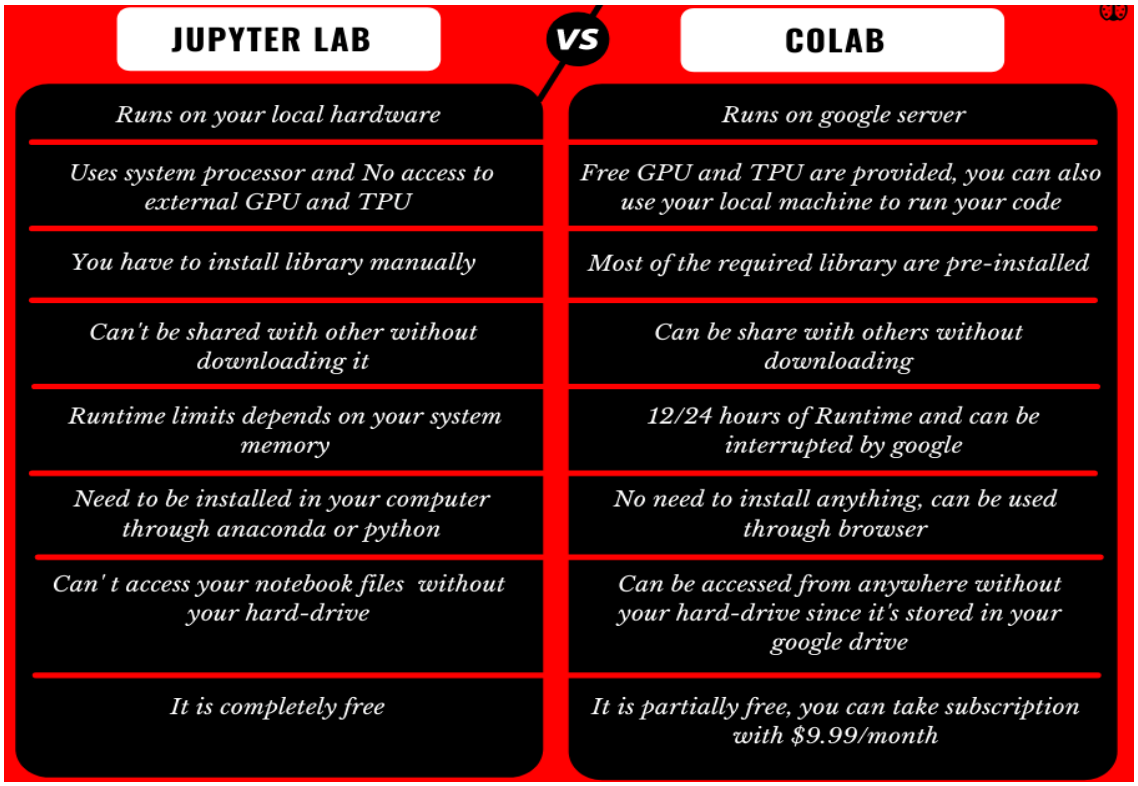
GOOGLE COLAB:

Colab notebooks allow you to combine executable code and rich text in a single document, along with images, HTML, LaTeX and more.

FEATURES OF COLAB:

- Uploading files
- Tutorials in Colab.
- Uploading Images in your notebook
- Using Tab to complete the code
- Directly Saving to GitHub.
- Saving your project as a PDF.
- Importing the libraries without installing them.





DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)

T.E/SEM VI/CBCGS/AIML
Academic Year: 2022-23

NAME	**PRATHAMESH CHIKANKAR**
BRANCH	**CSE-(AI&ML)**
ROLL NO.	**11**
SUBJECT	**MACHINE LEARNING LAB**
COURSE CODE	**CSL604**
PRACTICAL NO.	**02**
DOP	**03/02/2023**
DOS	

NUMPY:

NumPy is a very popular python library for large multi-dimensional array and matrix processing, with the help of a large collection of high-level mathematical functions. It is very useful for fundamental scientific computations in Machine Learning. It is particularly useful for linear algebra, Fourier transform, and random number capabilities. High-end libraries like TensorFlow use NumPy internally for manipulation of Tensors.

PROGRAM:

```
# operations
import numpy as np
# Creating two arrays of rank 2
x = np.array([[1, 2], [3, 4]])
y = np.array([[5, 6], [7, 8]])
# Creating two arrays of rank 1
v = np.array([9, 10])
w = np.array([11, 12])
# Inner product of vectors
print(np.dot(v, w), "\n")
# Matrix and Vector product
print(np.dot(x, v), "\n")
# Matrix and matrix product
print(np.dot(x, y))
```

OUTPUT:

In [1]:
```
# Python program using NumPy

import numpy as np

# Creating two arrays of rank 2
x = np.array([[1, 2], [3, 4]])
y = np.array([[5, 6], [7, 8]])

# Creating two arrays of rank 1
v = np.array([9, 10])
w = np.array([11, 12])

# Inner product of vectors
print(np.dot(v, w), "\n")

# Matrix and Vector product
print(np.dot(x, v), "\n")

# Matrix and matrix product
print(np.dot(x, y))
```

```
219

[29 67]

[[19 22]
 [43 50]]
```

In [ ]:

SCIPY:

SciPy is a very popular library among Machine Learning enthusiasts as it contains different modules for optimization, linear algebra, integration and statistics. There is a difference between the SciPy library and the SciPy stack. The SciPy is one of the core packages that make up the SciPy stack. SciPy is also very useful for image manipulation.

PROGRAM:

```
from scipy import io
import numpy as np
arr = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9,])
#Export:
io.savemat('arr.mat', {"vec": arr})
#Import:
mydata = io.loadmat('arr.mat')
print(mydata)
```

OUTPUT:

```
from scipy import io
import numpy as np

arr = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9,])

#Export:
io.savemat('arr.mat', {"vec": arr})

#Import:
mydata = io.loadmat('arr.mat')

print(mydata)
```

```
{
  '__header__': b'MATLAB 5.0 MAT-file Platform: nt, Created on: Tue Sep 22 13:12:32 2020',
  '__version__': '1.0',
  '__globals__': [],
  'vec': array([[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]])
}
```

TENSORFLOW:

TensorFlow is a very popular open-source library for high performance numerical computation developed by the Google Brain team in Google. As the name suggests, Tensorflow is a framework that involves defining and running computations involving tensors. It can train and run deep neural networks that can be used to develop several AI applications. TensorFlow is widely used in the field of deep learning research and application.

PROGRAM:

```
<!DOCTYPE html>
<html>
<script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs"></script>
<body>
<h1>TensorFlow JavaScript</h1>
<h3>Get the data behind a tensor:</h3>
<div id="demo"></div>
<script>
const myArr = [[1, 2], [3, 4]]; const tensorA = tf.tensor(myArr);
tensorA.data().then(data => display(data));
// Result: 1,2,3,4 function display(data) {
        document.getElementById("demo").innerHTML = data;
}
</script>
</body>
</html>
```

OUTPUT:

```
<!DOCTYPE html>
<html>
<script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs"></script>
<body>

<h1>TensorFlow JavaScript</h1>
<h3>Get the data behind a tensor:</h3>

<div id="demo"></div>

<script>
const myArr = [[1, 2], [3, 4]];
const tensorA = tf.tensor(myArr);
tensorA.data().then(data => display(data));

// Result: 1,2,3,4
function display(data) {
  document.getElementById("demo").innerHTML = data;
}
</script>
</body>
</html>
```

TensorFlow JavaScript

Get the data behind a tensor:

1,2,3,4

PANDAS:

Pandas is a popular Python library for data analysis. It is not directly related to Machine Learning. As we know that the dataset must be prepared before training. In this case, Pandas comes handy as it was developed specifically for data extraction and preparation. It provides high-level data structures and a wide variety of tools for data analysis. It provides many inbuilt methods for grouping, combining and filtering data.

PROGRAM:

```
# arranging a given set of data
# into a table
# importing pandas as pd
import pandas as pd
data = {"country": ["Brazil", "Russia", "India", "China", "South Africa"],
        "capital": ["Brasilia", "Moscow", "New Delhi", "Beijing", "Pretoria"],
        "area": [8.516, 17.10, 3.286, 9.597, 1.221],
        "population": [200.4, 143.5, 1252, 1357, 52.98] }
data_table = pd.DataFrame(data)
print(data_table)
```

OUTPUT:

In [4]:
```
# Python program using Pandas for
# arranging a given set of data
# into a table

# importing pandas as pd
import pandas as pd

data = {"country": ["Brazil", "Russia", "India", "China", "South Africa"],
"capital": ["Brasilia", "Moscow", "New Delhi", "Beijing", "Pretoria"],
"area": [8.516, 17.10, 3.286, 9.597, 1.221],
"population": [200.4, 143.5, 1252, 1357, 52.98] }

data_table = pd.DataFrame(data)
print(data_table)
```

```
        country    capital    area  population
0        Brazil   Brasilia   8.516      200.40
1        Russia     Moscow  17.100      143.50
2         India  New Delhi   3.286     1252.00
3         China    Beijing   9.597     1357.00
4  South Africa   Pretoria   1.221       52.98
```

In [ ]:

MATPLOTLIB:

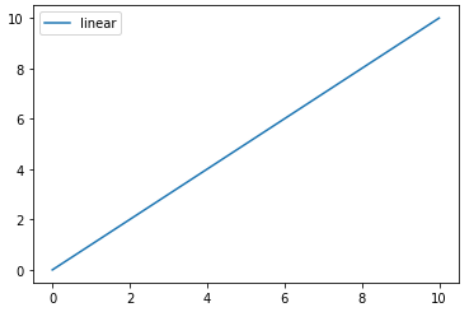
Matplotlib is a very popular Python library for data visualization. Like Pandas, it is not directly related to Machine Learning. It particularly comes in handy when a programmer wants to visualize the patterns in the data. It is a 2D plotting library used for creating 2D graphs and plots. A module named pyplot makes it easy for programmers for plotting as it provides features to control line styles, font properties, formatting axes, etc. It provides various kinds of graphs and plots for data visualization, viz., histogram, error charts, bar charts, etc.

PROGRAM:

```
# for forming a linear plot
import matplotlib.pyplot as plt
import numpy as np
# Prepare the data
x = np.linspace(0, 10, 100)
# Plot the data
plt.plot(x, x, label ='linear')
# Add a legend
plt.legend()
# Show the plot
plt.show()
```

OUTPUT:

In [5]:
```
# Python program using Matplotlib
# for forming a linear plot

import matplotlib.pyplot as plt
import numpy as np

# Prepare the data
x = np.linspace(0, 10, 100)

# Plot the data
plt.plot(x, x, label ='linear')

# Add a legend
plt.legend()

# Show the plot
plt.show()
```



In [ ]:

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)

T.E/SEM VI/CBCGS/AIML
Academic Year: 2022-23

NAME	**PRATHAMESH CHIKANKAR**
BRANCH	**CSE-(AI&ML)**
ROLL NO.	**11**
SUBJECT	**MACHINE LEARNING LAB**
COURSE CODE	**CSL604**
PRACTICAL NO.	**03**
DOP	**10/02/2023**
DOS	

Program(input)/Output :

- Import Libraries

```
In [1]: import pandas as pd
        import numpy as np
        import seaborn as sns
        import matplotlib.pyplot as plt

        %matplotlib inline
```

- Importing Data and Checking out

data_frame.head()

```
In [2]: HouseDF = pd.read_csv('USA_Housing.csv')
```

```
In [3]: HouseDF.head()
```

Out[3]:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price	Addres
0	79545.458574	5.682861	7.009188	4.09	23086.800503	1.059034e+06	208 Michael Ferry Ap 674\nLaurabury, N 3701.
1	79248.642455	6.002900	6.730821	3.09	40173.072174	1.505891e+06	188 Johnson View Suite 079\nLak Kathleen, CA.
2	61287.067179	5.865890	8.512727	5.13	36882.159400	1.058988e+06	9127 Elizabet Stravenue\nDanieltowr WI 06482.
3	63345.240046	7.188236	5.586729	3.26	34310.242831	1.260617e+06	USS Barnett\nFPO A 4482
4	59982.197226	5.040555	7.839388	4.23	26354.109472	6.309435e+05	USNS Raymond\nFP(AE 0938

data_frame.info()

```
In [4]: HouseDF.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 7 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   Avg. Area Income              5000 non-null   float64
 1   Avg. Area House Age           5000 non-null   float64
 2   Avg. Area Number of Rooms     5000 non-null   float64
 3   Avg. Area Number of Bedrooms  5000 non-null   float64
 4   Area Population               5000 non-null   float64
 5   Price                         5000 non-null   float64
 6   Address                       5000 non-null   object 
dtypes: float64(6), object(1)
memory usage: 273.6+ KB
```

data_frame.describe()

```
In [5]: HouseDF.describe()
```

Out[5]:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price
count	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5.000000e+03
mean	68583.108984	5.977222	6.987792	3.981330	36163.516039	1.232073e+06
std	10657.991214	0.991456	1.005833	1.234137	9925.650114	3.531176e+05
min	17796.631190	2.644304	3.236194	2.000000	172.610686	1.593866e+04
25%	61480.562388	5.322283	6.299250	3.140000	29403.928702	9.975771e+05
50%	68804.286404	5.970429	7.002902	4.050000	36199.406689	1.232669e+06
75%	75783.338666	6.650808	7.665871	4.490000	42861.290769	1.471210e+06
max	107701.748378	9.519088	10.759588	6.500000	69621.713378	2.469066e+06

data_frame.columns

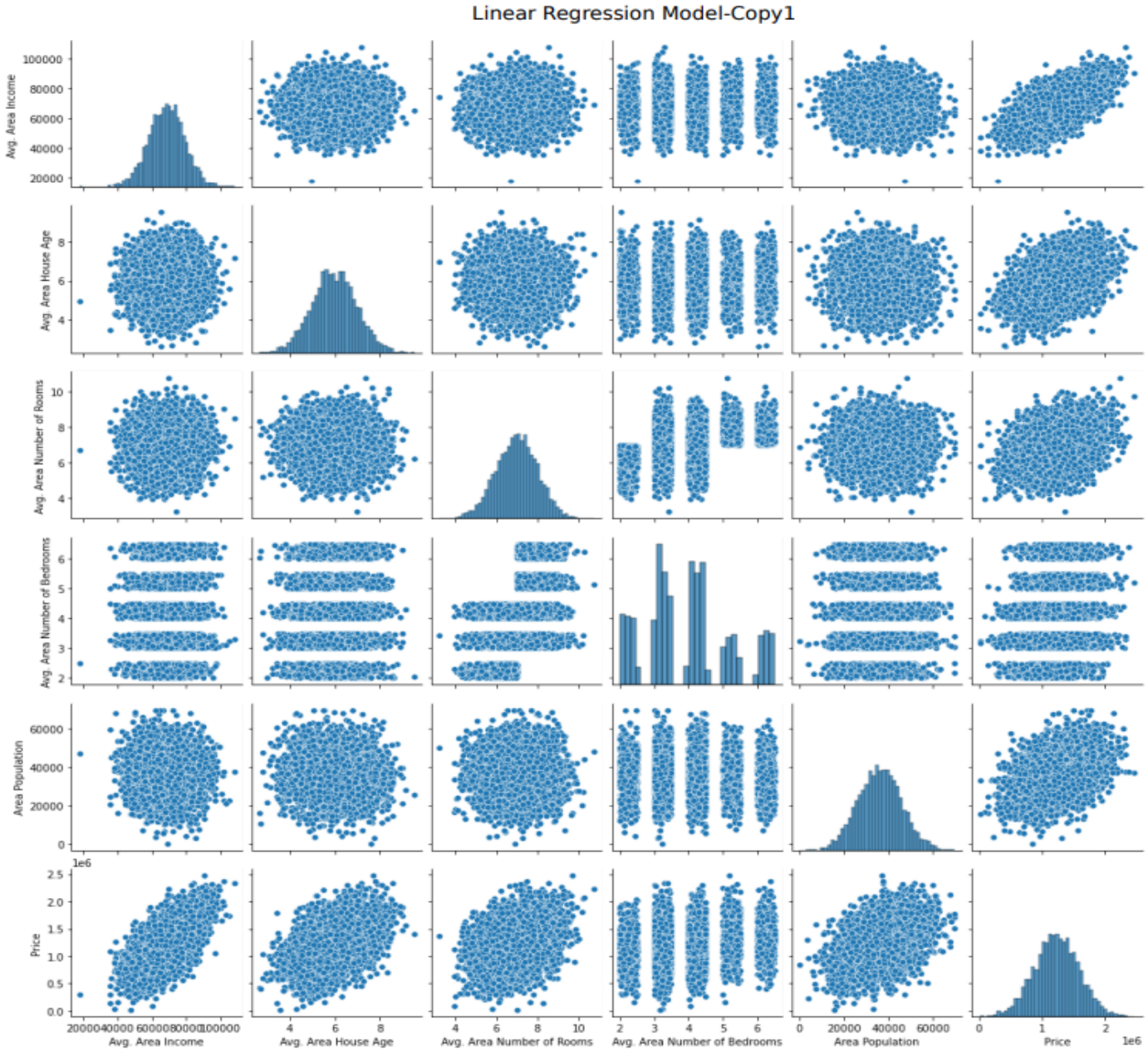
```
In [6]: HouseDF.columns
```

```
Out[6]: Index(['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms',
       'Avg. Area Number of Bedrooms', 'Area Population', 'Price', 'Address'],
      dtype='object')
```

- Exploratory Data Analysis for House Price Prediction

sns.pairplot(data_frame)

```
In [7]: sns.pairplot(HouseDF)
Out[7]: <seaborn.axisgrid.PairGrid at 0x7fb14ff35e80>
```

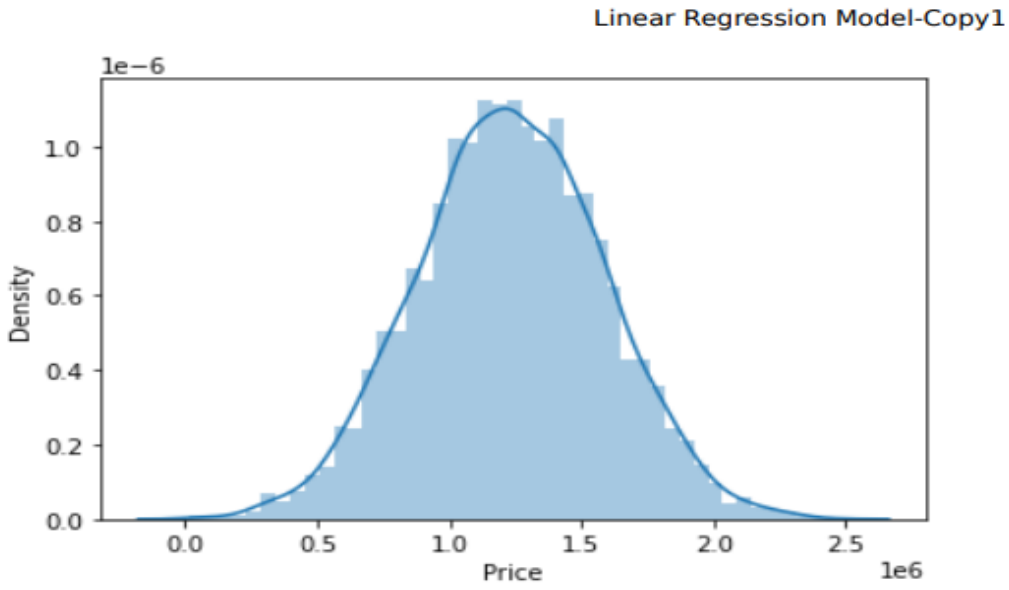


sns.distplot(data_frame[ ])

```
In [8]: sns.distplot(HouseDF['Price'])
```

```
/home/computer/anaconda3/lib/python3.9/site-packages/seaborn/distributions.
py:2619: FutureWarning: `distplot` is a deprecated function and will be rem
oved in a future version. Please adapt your code to use either `displot` (a
figure-level function with similar flexibility) or `histplot` (an axes-leve
l function for histograms).
  warnings.warn(msg, FutureWarning)
```

```
Out[8]: <AxesSubplot:xlabel='Price', ylabel='Density'>
```



sns.heatmap(data_frame.corr(),annot=True)

```
In [9]: sns.heatmap(HouseDF.corr(), annot=True)
```

```
Out[9]: <AxesSubplot:>
```

- Training a Linear Regression Model

X and y List

```
In [10]: X = HouseDF[['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number o
                      'Avg. Area Number of Bedrooms', 'Area Population']]

         y = HouseDF['Price']
```

Split Data into Train, Test

```
In [11]: from sklearn.model_selection import train_test_split
```

```
In [12]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, ra
```

- Creating and Training the LinearRegression Model

```
In [13]: from sklearn.linear_model import LinearRegression
```

```
In [14]: lm = LinearRegression()
```

```
In [15]: lm.fit(X_train,y_train)
```

```
Out[15]: LinearRegression()
```

- LinearRegression Model Evaluation

```
In [16]: print(lm.intercept_)
```

```
-2640159.79685191
```

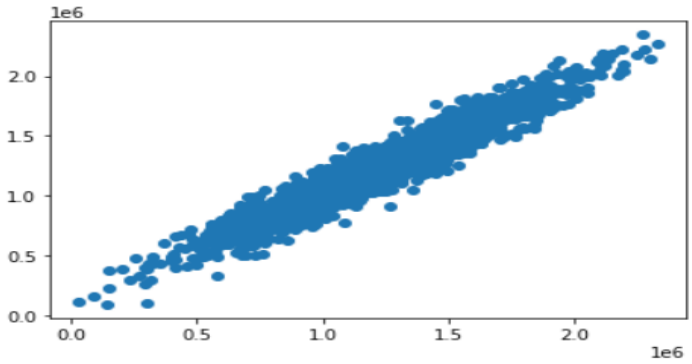
```
In [17]: coeff_df = pd.DataFrame(lm.coef_,X.columns,columns=['Coefficient'])
         coeff_df
```

Out[17]:

	Coefficient
Avg. Area Income	21.528276
Avg. Area House Age	164883.282027
Avg. Area Number of Rooms	122368.678027
Avg. Area Number of Bedrooms	2233.801864
Area Population	15.150420

- Predictions from our Linear Regression Model

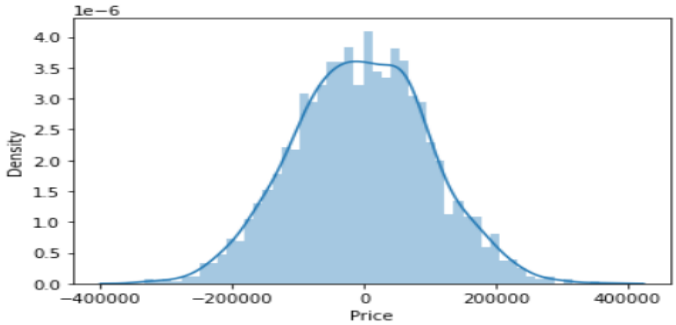
```
In [18]: predictions = lm.predict(X_test)

In [19]: plt.scatter(y_test,predictions)

Out[19]: <matplotlib.collections.PathCollection at 0x7fb14c0c6d00>
```



In the above scatter plot, we see data is in line shape, which means our model has done good predictions.

sns.distplot((y_test-predictions),bins=50)

```
In [20]: sns.distplot((y_test-predictions),bins=50);

/home/computer/anaconda3/lib/python3.9/site-packages/seaborn/distributions.
py:2619: FutureWarning: `distplot` is a deprecated function and will be rem
oved in a future version. Please adapt your code to use either `displot` (a
figure-level function with similar flexibility) or `histplot` (an axes-leve
l function for histograms).
  warnings.warn(msg, FutureWarning)
```



In the above histogram plot, we see data is in bell shape (Normally Distributed), which means our model has done good predictions.

- Regression Evaluation Metrics

```
In [21]: from sklearn import metrics

In [22]: print('MAE:', metrics.mean_absolute_error(y_test, predictions))
         print('MSE:', metrics.mean_squared_error(y_test, predictions))
         print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predictions)))

         MAE: 82288.22251914954
         MSE: 10460958907.209501
         RMSE: 102278.82922291153

In [ ]:
```

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)

T.E/SEM VI/CBCGS/AIML
Academic Year: 2022-23

NAME	**PRATHAMESH CHIKANKAR**
BRANCH	**CSE-(AI&ML)**
ROLL NO.	**11**
SUBJECT	**MACHINE LEARNING LAB**
COURSE CODE	**CSL604**
PRACTICAL NO.	**04**
DOP	
DOS	

Program(input)/Output :

- Import Libraries

In [2]:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from seaborn import load_dataset
%matplotlib inline
plt.style.use('ggplot')
```

- Importing Data and Checking out

data_frame.head()

In [3]:

```
data = load_dataset("titanic")
data
```

Out[3]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_m
0	0	3	male	22.0	1	0	7.2500	S	Third	man	T
1	1	1	female	38.0	1	0	71.2833	C	First	woman	Fa
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	Fa
3	1	1	female	35.0	1	0	53.1000	S	First	woman	Fa
4	0	3	male	35.0	0	0	8.0500	S	Third	man	T
...	...	...	...	...	...	...	...	...	...	...	
886	0	2	male	27.0	0	0	13.0000	S	Second	man	T
887	1	1	female	19.0	0	0	30.0000	S	First	woman	Fa
888	0	3	female	NaN	1	2	23.4500	S	Third	woman	Fa
889	1	1	male	26.0	0	0	30.0000	C	First	man	T
890	0	3	male	32.0	0	0	7.7500	Q	Third	man	T

891 rows × 15 columns

data_frame.info()

In [4]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype   
---  ------       --------------  -----   
 0   survived     891 non-null    int64   
 1   pclass       891 non-null    int64   
 2   sex          891 non-null    object  
 3   age          714 non-null    float64 
 4   sibsp        891 non-null    int64   
 5   parch        891 non-null    int64   
 6   fare         891 non-null    float64 
 7   embarked     889 non-null    object  
 8   class        891 non-null    category
 9   who          891 non-null    object  
 10  adult_male   891 non-null    bool    
 11  deck         203 non-null    category
 12  embark_town  889 non-null    object  
 13  alive        891 non-null    object  
 14  alone        891 non-null    bool    
dtypes: bool(2), category(2), float64(2), int64(4), object(5)
memory usage: 80.7+ KB
```

data_frame.columns
data_frame.describe()

In [5]:

```
columns = ['alive', 'alone', 'embark_town', 'who', 'adult_male', 'deck']
data_2 = data.drop(columns, axis=1)
```

In [6]:

```
data_2.describe(include='all').T
```

Out[6]:

	count	unique	top	freq	mean	std	min	25%	50%	75%	m
survived	891.0	NaN	NaN	NaN	0.383838	0.486592	0.0	0.0	0.0	1.0	
pclass	891.0	NaN	NaN	NaN	2.308642	0.836071	1.0	2.0	3.0	3.0	
sex	891	2	male	577	NaN	NaN	NaN	NaN	NaN	NaN	N
age	714.0	NaN	NaN	NaN	29.699118	14.526497	0.42	20.125	28.0	38.0	8
sibsp	891.0	NaN	NaN	NaN	0.523008	1.102743	0.0	0.0	0.0	1.0	
parch	891.0	NaN	NaN	NaN	0.381594	0.806057	0.0	0.0	0.0	0.0	
fare	891.0	NaN	NaN	NaN	32.204208	49.693429	0.0	7.9104	14.4542	31.0	512.3
embarked	889	3	S	644	NaN	NaN	NaN	NaN	NaN	NaN	N
class	891	3	Third	491	NaN	NaN	NaN	NaN	NaN	NaN	N

- max and min values :

In [7]:

```
print(f"Max value of age column : {data_2['age'].max()}")
print(f"Min value of age column : {data_2['age'].min()}")
```

```
Max value of age column : 80.0
Min value of age column : 0.42
```

- data_set_2 :

In [8]:

```
bins = [0, 5, 17, 25, 50, 80]
labels = ['Infant', 'Kid', 'Young', 'Adult', 'Old']
data_2['age'] = pd.cut(data_2['age'], bins = bins, labels=labels)
```

In [9]:

```
pd.DataFrame(data_2['age'].value_counts())
```

Out[9]:

	age
Adult	349
Young	188
Kid	69
Old	64
Infant	44

data_frame.mode()[ ]
data_frame.fillna()
data_frame[ ].unique()

In [12]:

```
data_2['age'].mode()[0]
```

Out[12]:

```
'Adult'
```

In [14]:

```
data_4 = data_2.fillna({'age' : data_2['age'].mode()[0]})
```

In [15]:

```
data_2['embarked'].unique()
```

Out[15]:

```
array(['S', 'C', 'Q', nan], dtype=object)
```

In [16]:

```
print(f"How many 'S' on embarked column : {data_2[data_2['embarked'] == 'S'].shape[
print(f"How many 'C' on embarked column : {data_2[data_2['embarked'] == 'C'].shape[
print(f"How many 'Q' on embarked column : {data_2[data_2['embarked'] == 'Q'].shape[
```

```
How many 'S' on embarked column : 644
How many 'C' on embarked column : 168
How many 'Q' on embarked column : 77
```

- filling data_set_3 in data_set_2

In [17]:

```
data_3 = data_2.fillna({'embarked' : 'S'})
data_4[['pclass', 'survived']].groupby(['pclass']).sum().sort_values(by='survived')
```

Out[17]:

pclass	survived
2	87
3	119
1	136

In [18]:

```
data_4[['sex', 'survived']].groupby(['sex']).sum().sort_values(by='survived')
```

Out[18]:

sex	survived
male	109
female	233

In [29]:

```
bins = [-1, 7.9104, 14.4542, 31, 512.330]
labels = ['low','medium-low','medium','high']
data_4['fare']=pd.cut(data_3['fare'],bins = bins,labels = labels)
```

sns.distplot(data_frame[ ])

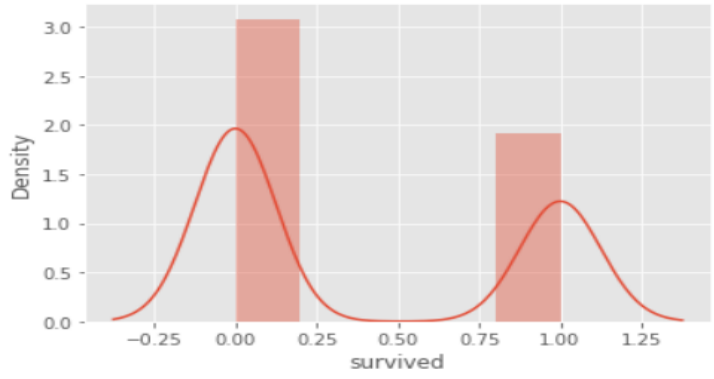
In [27]:

```
data_5 = data_4.drop('class', axis=1)
sns.distplot(data_5['survived'])
```

```
/home/computer/anaconda3/lib/python3.9/site-packages/seaborn/distribut
ions.py:2619: FutureWarning: `distplot` is a deprecated function and w
ill be removed in a future version. Please adapt your code to use eith
er `displot` (a figure-level function with similar flexibility) or `hi
stplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[27]:

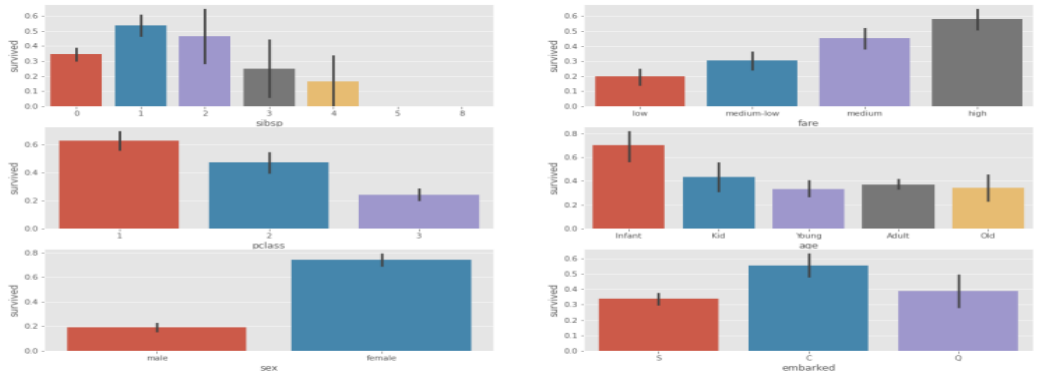
```
<AxesSubplot:xlabel='survived', ylabel='Density'>
```



plt.figure()
plt.subplot()
plt.barplot(x=‘ ’,y=‘ ’,data=data_set)

In [24]:

```
plt.figure(figsize=(20, 10))
plt.subplot(321)
sns.barplot(x = 'sibsp', y = 'survived', data = data_5)
plt.subplot(322)
sns.barplot(x = 'fare', y = 'survived', data = data_5)
plt.subplot(323)
sns.barplot(x = 'pclass', y = 'survived', data = data_5)
plt.subplot(324)
sns.barplot(x = 'age', y = 'survived', data = data_5)
plt.subplot(325)
sns.barplot(x = 'sex', y = 'survived', data = data_5)
plt.subplot(326)
sns.barplot(x = 'embarked', y = 'survived', data = data_5);
```



- creating data_set for dummies

data_frame.shape

In [30]:

```
dummies = ['fare', 'age', 'embarked', 'sex']
dummy_data = pd.get_dummies(data_5[dummies])
```

In [31]:

```
dummy_data.shape
```

Out[31]:

```
(891, 14)
```

- creating another data_set

In [32]:

```
data_6 = pd.concat([data_5, dummy_data], axis = 1)
data_6.drop(dummies, axis=1, inplace=True)
```

- importing libraries

In [33]:

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix
```

data_frame.drop()

In [34]:

```
X = data_6.drop('survived', axis = 1)
y = data_6['survived']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.33, random_
```

- logistic regression

data_frame.fit()

data_frame.predict()

In [35]:

```
log_reg = LogisticRegression()
log_reg.fit(X_train, y_train)
y_pred = log_reg.predict(X_test)
y_pred
```

Out[35]:

```
array([0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0,
1,
       0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
0,
       1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1,
0,
       1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1,
0,
       1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0,
1,
       0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1,
0,
       1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1,
0,
       1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0,
1,
       0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
1,
       0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0,
1,
       0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0,
0,
       0, 0, 0, 0, 0, 0, 1, 1, 1])
```

- calculating accuracy and confusion matrix

In [36]:

```
accuracy_score(y_pred, y_test)
```

Out[36]:

```
0.8067796610169492
```

In [37]:

```
confusion_matrix(y_pred, y_test)
```

Out[37]:

```
array([[158,  31],
       [ 26,  80]])
```

In [ ]:

```
# 31 + 26 = 57 wrong prediction
```

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)

T.E/SEM VI/CBCGS/AIML
Academic Year: 2022-23

NAME	**PRATHAMESH CHIKANKAR**
BRANCH	**CSE-(AI&ML)**
ROLL NO.	**11**
SUBJECT	**MACHINE LEARNING LAB**
COURSE CODE	**CSL604**
PRACTICAL NO.	**05**
DOP	**24/03/2023**
DOS	

Program(input)/Output :

- **Data Preprocessing Step**
  1. Importing libraries & datasets
  2. Extracting Independent and dependent variable
  3. Splitting the dataset into training and test set
  4. Feature Scaling

```
In [5]: #Data Pre-processing Step
        # importing libraries
        import numpy as nm
        import matplotlib.pyplot as mtp
        import pandas as pd

        #importing datasets
        data_set= pd.read_csv('/home/computer/Downloads/suv_data.csv')

        #Extracting Independent and dependent Variable
        x= data_set.iloc[:, [2,3]].values
        y= data_set.iloc[:, 4].values

        # Splitting the dataset into training 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
        #feature Scaling
        from sklearn.preprocessing import StandardScaler
        st_x= StandardScaler()
        x_train= st_x.fit_transform(x_train)
        x_test= st_x.transform(x_test)
```

The scaled output for the test set will be:

```
In [6]: data_set
```

Out[6]:

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
...	...	...	...	...	...
395	15691863	Female	46	41000	1
396	15706071	Male	51	23000	1
397	15654296	Female	50	20000	1
398	15755018	Male	36	33000	0
399	15594041	Female	49	36000	1

400 rows × 5 columns

- **Fitting the SVM classifier to the training set**
  1. Support vector classifier

```
In [7]: from sklearn.svm import SVC # "Support vector classifier"
        classifier = SVC(kernel='linear', random_state=0)
        classifier.fit(x_train, y_train)
```

```
Out[7]: SVC(kernel='linear', random_state=0)
```

Below is the output for the prediction of the test set

```
In [9]: y_pred

Out[9]: array([0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
        0, 1,
               0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
        0, 0,
               1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
        0, 1,
               0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1,
        0, 1,
               0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1])
```

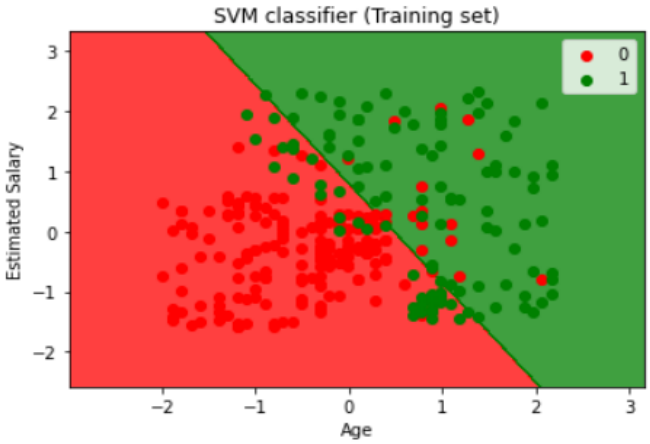
- **Creating the confusion matrix:**

```
In [10]: #Creating the Confusion matrix
         from sklearn.metrics import confusion_matrix
         cm= confusion_matrix(y_test, y_pred)

In [11]: cm

Out[11]: array([[66,  2],
                [ 8, 24]])
```

- **Visualizing the training set result:**

```
In [32]:
from matplotlib.colors import ListedColormap
x_set, y_set = x_train, y_train
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() +
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1
alpha = 0.75, cmap = ListedColormap(('red', 'green')))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('red', 'green'))(i), label = j)
mtp.title('SVM classifier (Training set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should
be avoided as value-mapping will have precedence in case its length matches
with *x* & *y*.  Please use the *color* keyword-argument or provide a 2D ar
ray with a single row if you intend to specify the same RGB or RGBA value f
or all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should
be avoided as value-mapping will have precedence in case its length matches
with *x* & *y*.  Please use the *color* keyword-argument or provide a 2D ar
ray with a single row if you intend to specify the same RGB or RGBA value f
or all points.
```

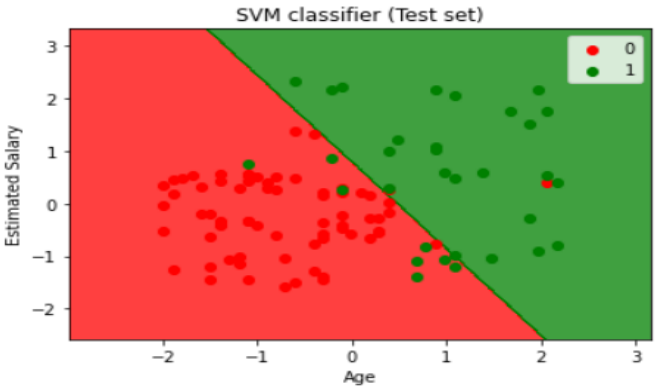


- **Visualizing the test set result:**

In [33]:

```
from matplotlib.colors import ListedColormap
x_set, y_set = x_test, y_test
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() +
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1
alpha = 0.75, cmap = ListedColormap(('red','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('red', 'green'))(i), label = j)
mtp.title('SVM classifier (Test set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should
be avoided as value-mapping will have precedence in case its length matches
with *x* & *y*.  Please use the *color* keyword-argument or provide a 2D ar
ray with a single row if you intend to specify the same RGB or RGBA value f
or all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should
be avoided as value-mapping will have precedence in case its length matches
with *x* & *y*.  Please use the *color* keyword-argument or provide a 2D ar
ray with a single row if you intend to specify the same RGB or RGBA value f
or all points.
```



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)

T.E/SEM VI/CBCGS/AIML
Academic Year: 2022-23

NAME	**PRATHAMESH CHIKANKAR**
BRANCH	**CSE-(AI&ML)**
ROLL NO.	**11**
SUBJECT	**MACHINE LEARNING LAB**
COURSE CODE	**CSL604**
PRACTICAL NO.	**06**
DOP	
DOS	

Program - Output :

- **Defining Variables -**

```
def hebbian_learning(samples):
        print(f'{"INPUT":^8} {"TARGET":^16}{"WEIGHT CHANGES":^15}{"WEIGHTS":^25}')
         w1, w2, b = 0, 0, 0
        print(' ' * 45, f'({w1:2}, {w2:2}, {b:2})')
        for x1, x2, y in samples:
                w1 = w1 + x1 * y
                w2 = w2 + x2 * y
                b = b + y
                print(f'({x1:2}, {x2:2}) {y:2} ({x1y:2}, {x2y:2}, {y:2}) ({w1:2}, {w2:2}, {b:2})')
```

In [24]:

```
def hebbian_learning(samples):
    print(f'{"INPUT":^8} {"TARGET":^16}{"WEIGHT CHANGES":^15}{"WEIGHTS":^25}')
    w1, w2, b = 0, 0, 0
    print(' ' * 45, f'({w1:2}, {w2:2}, {b:2})')
    for x1, x2, y in samples:
        w1 = w1 + x1 * y
        w2 = w2 + x2 * y
        b = b + y
        print(f'({x1:2}, {x2:2}) {y:2} ({x1:2}, {x2:2}, {y:2}) ({w1:2}, {w2:2}, {
```

- **Defining inputs -**
  1. **AND logic gate**

In [ ]:

```
AND_samples = {
    'binary_input_binary_output': [
        [1, 1, 1],
        [1, 0, 0],
        [0, 1, 0],
        [0, 0, 0]
    ],
    'binary_input_bipolar_output': [
        [1, 1, 1],
        [1, 0, -1],
        [0, 1, -1],
        [0, 0, -1]
    ],
    'bipolar_input_bipolar_output': [
        [ 1, 1, 1],
        [ 1, -1, -1],
        [-1, 1, -1],
        [-1, -1, -1]
    ]
}
```

2. **OR logic gate**
3. **XOR logic gate**

```
OR_samples = {
    'binary_input_binary_output': [
        [1, 1, 1],
        [1, 0, 1],
        [0, 1, 1],
        [0, 0, 0]
    ],
    'binary_input_bipolar_output': [
        [1, 1, 1],
        [1, 0, 1],
        [0, 1, 1],
        [0, 0, -1]
    ],
    'bipolar_input_bipolar_output': [
        [ 1, 1, 1],
        [ 1, -1, 1],
        [-1, 1, 1],
        [-1, -1, -1]
    ]
}
XOR_samples = {
    'binary_input_binary_output': [
        [1, 1, 0],
        [1, 0, 1],
        [0, 1, 1],
        [0, 0, 0]
    ],
    'binary_input_bipolar_output': [
        [1, 1, -1],
        [1, 0, 1],
        [0, 1, 1],
        [0, 0, -1]
    ],
    'bipolar_input_bipolar_output': [
        [ 1, 1, -1],
        [ 1, -1, 1],
        [-1, 1, 1],
        [-1, -1, -1]
    ]
}
```

- **Testing Code with Output -**
  1. **For AND logic gate**

```
In [26]:
```

```
print('-'*20 ,'HEBBIAN LEARNING', '-'*20)
print('AND with Binary Input and Binary Output')
hebbian_learning(AND_samples['binary_input_binary_output'])
print('AND with Binary Input and Bipolar Output')
hebbian_learning(AND_samples['binary_input_bipolar_output'])
print('AND with Bipolar Input and Bipolar Output')
hebbian_learning(AND_samples['bipolar_input_bipolar_output'])
```

```
-------------------- HEBBIAN LEARNING --------------------
AND with Binary Input and Binary Output
 INPUT        TARGET     WEIGHT CHANGES          WEIGHTS
                                              ( 0,  0,  0)

( 1,  1)  1 ( 1,  1,  1) ( 1,  1,  1)
( 1,  0)  0 ( 1,  0,  0) ( 1,  1,  1)
( 0,  1)  0 ( 0,  1,  0) ( 1,  1,  1)
( 0,  0)  0 ( 0,  0,  0) ( 1,  1,  1)
AND with Binary Input and Bipolar Output
 INPUT        TARGET     WEIGHT CHANGES          WEIGHTS
                                              ( 0,  0,  0)

( 1,  1)  1 ( 1,  1,  1) ( 1,  1,  1)
( 1,  0) -1 ( 1,  0, -1) ( 0,  1,  0)
( 0,  1) -1 ( 0,  1, -1) ( 0,  0, -1)
( 0,  0) -1 ( 0,  0, -1) ( 0,  0, -2)
AND with Bipolar Input and Bipolar Output
 INPUT        TARGET     WEIGHT CHANGES          WEIGHTS
                                              ( 0,  0,  0)

( 1,  1)  1 ( 1,  1,  1) ( 1,  1,  1)
( 1, -1) -1 ( 1, -1, -1) ( 0,  2,  0)
(-1,  1) -1 (-1,  1, -1) ( 1,  1, -1)
(-1, -1) -1 (-1, -1, -1) ( 2,  2, -2)
```

2. Using OR logic gate

In [27]:

```
print('-'*20, 'HEBBIAN LEARNING', '-'*20)
print('OR with binary input and binary output')
hebbian_learning(OR_samples['binary_input_binary_output'])
print('OR with binary input and bipolar output')
hebbian_learning(OR_samples['binary_input_bipolar_output'])
print('OR with bipolar input and bipolar output')
hebbian_learning(OR_samples['bipolar_input_bipolar_output'])
```

```
-------------------- HEBBIAN LEARNING --------------------
OR with binary input and binary output
 INPUT         TARGET     WEIGHT CHANGES          WEIGHTS
                                                ( 0,  0,  0)

( 1,  1)  1 ( 1,  1,  1) ( 1,  1,  1)
( 1,  0)  1 ( 1,  0,  1) ( 2,  1,  2)
( 0,  1)  1 ( 0,  1,  1) ( 2,  2,  3)
( 0,  0)  0 ( 0,  0,  0) ( 2,  2,  3)
OR with binary input and bipolar output
 INPUT         TARGET     WEIGHT CHANGES          WEIGHTS
                                                ( 0,  0,  0)

( 1,  1)  1 ( 1,  1,  1) ( 1,  1,  1)
( 1,  0)  1 ( 1,  0,  1) ( 2,  1,  2)
( 0,  1)  1 ( 0,  1,  1) ( 2,  2,  3)
( 0,  0) -1 ( 0,  0, -1) ( 2,  2,  2)
OR with bipolar input and bipolar output
 INPUT         TARGET     WEIGHT CHANGES          WEIGHTS
                                                ( 0,  0,  0)

( 1,  1)  1 ( 1,  1,  1) ( 1,  1,  1)
( 1, -1)  1 ( 1, -1,  1) ( 2,  0,  2)
(-1,  1)  1 (-1,  1,  1) ( 1,  1,  3)
(-1, -1) -1 (-1, -1, -1) ( 2,  2,  2)
```

3. For XOR logic gate

In [28]:

```
print('-'*20, 'HEBBIAN LEARNING', '-'*20)
print('XOR with binary input and binary output')
hebbian_learning(XOR_samples['binary_input_binary_output'])
print('XOR with binary input and bipolar output')
hebbian_learning(XOR_samples['binary_input_bipolar_output'])
print('XOR with bipolar input and bipolar output')
hebbian_learning(XOR_samples['bipolar_input_bipolar_output'])
```

```
-------------------- HEBBIAN LEARNING --------------------
XOR with binary input and binary output
 INPUT         TARGET     WEIGHT CHANGES          WEIGHTS
                                                ( 0,  0,  0)

( 1,  1)  0 ( 1,  1,  0) ( 0,  0,  0)
( 1,  0)  1 ( 1,  0,  1) ( 1,  0,  1)
( 0,  1)  1 ( 0,  1,  1) ( 1,  1,  2)
( 0,  0)  0 ( 0,  0,  0) ( 1,  1,  2)
XOR with binary input and bipolar output
 INPUT         TARGET     WEIGHT CHANGES          WEIGHTS
                                                ( 0,  0,  0)

( 1,  1) -1 ( 1,  1, -1) (-1, -1, -1)
( 1,  0)  1 ( 1,  0,  1) ( 0, -1,  0)
( 0,  1)  1 ( 0,  1,  1) ( 0,  0,  1)
( 0,  0) -1 ( 0,  0, -1) ( 0,  0,  0)
XOR with bipolar input and bipolar output
 INPUT         TARGET     WEIGHT CHANGES          WEIGHTS
                                                ( 0,  0,  0)

( 1,  1) -1 ( 1,  1, -1) (-1, -1, -1)
( 1, -1)  1 ( 1, -1,  1) ( 0, -2,  0)
(-1,  1)  1 (-1,  1,  1) (-1, -1,  1)
(-1, -1) -1 (-1, -1, -1) ( 0,  0,  0)
```

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)

T.E/SEM VI/CBCGS/AIML
Academic Year: 2022-23

NAME	**PRATHAMESH CHIKANKAR**
BRANCH	**CSE-(AI&ML)**
ROLL NO.	**11**
SUBJECT	**MACHINE LEARNING LAB**
COURSE CODE	**CSL604**
PRACTICAL NO.	**07**
DOP	
DOS	

Program - Output :

- **Importing Libraries -**
  1. **For plotting**
  2. **For matrix math**
  3. **For normalization + probability density function computation**
  4. **For data preprocessing**

In [1]:
```
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style("white")
%matplotlib inline

import numpy as np

from scipy import stats

import pandas as pd
from math import sqrt, log, exp, pi
from random import uniform
print("import done")
```

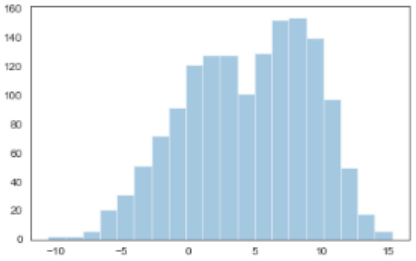
```
import done
```

- **Generating the data yourself, Select μ1, σ1 and μ2, σ2 to generate the data**

In [2]:
```
random_seed=36788765
np.random.seed(random_seed)

Mean1 = 2.0
Standard_dev1 = 4.0
Mean2 = 9.0
Standard_dev2 = 2.0


y1 = np.random.normal(Mean1, Standard_dev1, 1000)
y2 = np.random.normal(Mean2, Standard_dev2, 500)
data=np.append(y1,y2)


Min_graph = min(data)
Max_graph = max(data)
x = np.linspace(Min_graph, Max_graph, 2000)

print('Input Gaussian {:}: μ = {:.2}, σ = {:.2}'.format("1", Mean1, Standard_dev1))
print('Input Gaussian {:}: μ = {:.2}, σ = {:.2}'.format("2", Mean2, Standard_dev2))
sns.distplot(data, bins=20, kde=False);
```

```
Input Gaussian 1: μ = 2.0, σ = 4.0
Input Gaussian 2: μ = 9.0, σ = 2.0
```

```
/home/computer/anaconda3/lib/python3.9/site-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



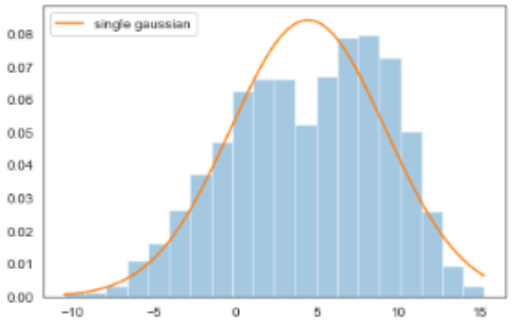
In [3]:
```
class Gaussian:
    "Model univariate Gaussian"
    def __init__(self, mu, sigma):

        self.mu = mu
        self.sigma = sigma

    #probability density function
    def pdf(self, datum):
        "Probability of a data point given the current parameters"
        u = (datum - self.mu) / abs(self.sigma)
        y = (1 / (sqrt(2 * pi) * abs(self.sigma))) * exp(-u * u / 2)
        return y

    def __repr__(self):
        return 'Gaussian({0:4.6}, {1:4.6})'.format(self.mu, self.sigma)
print("done")
```

```
done
```

- **Calculating the mean and standard deviation of the dataset shows it does not fit well (as single Gaussion will not fit the data well)**

In [4]:
```
best_single = Gaussian(np.mean(data), np.std(data))
print('Best single Gaussian: μ = {:.2}, σ = {:.2}'.format(best_single.mu, best_single.sigma))
#fit a single gaussian curve to the data
g_single = stats.norm(best_single.mu, best_single.sigma).pdf(x)
sns.distplot(data, bins=20, kde=False, norm_hist=True);
plt.plot(x, g_single, label='single gaussian');
plt.legend();
```

```
Best single Gaussian: μ = 4.4, σ = 4.8
```



- **EM with 2 Gaussian mixture model**

In [5]:
```
class GaussianMixture_self:
    "Model mixture of two univariate Gaussians and their EM estimation"

    def __init__(self, data, mu_min=min(data), mu_max=max(data), sigma_min=1, sigma_max=1, mix=.5):
        self.data = data
        #todo the Algorithm would be numerical enhanced by normalizing the data first, next do all the EM steps and

        #init with multiple gaussians
        self.one = Gaussian(uniform(mu_min, mu_max),
                            uniform(sigma_min, sigma_max))
        self.two = Gaussian(uniform(mu_min, mu_max),
                            uniform(sigma_min, sigma_max))

        #as well as how much to mix them
        self.mix = mix
```

```
    def Estep(self):
        "Perform an E(stimation)-step, assign each point to gaussian 1 or 2 with a percentage"
        # compute weights
        self.loglike = 0. # = log(p = 1)
        for datum in self.data:
            # unnormalized weights
            wp1 = self.one.pdf(datum) * self.mix
            wp2 = self.two.pdf(datum) * (1. - self.mix)
            # compute denominator
            den = wp1 + wp2
            # normalize
            wp1 /= den
            wp2 /= den      # wp1+wp2= 1, it either belongs to gaussian 1 or gaussion 2
            # add into loglike
            self.loglike += log(den) #freshening up self.loglike in the process
            # yield weight tuple
            yield (wp1, wp2)
```

```
    def Mstep(self, weights):
        "Perform an M(aximization)-step"
        # compute denominators
        (left, rigt) = zip(*weights)
        one_den = sum(left)
        two_den = sum(rigt)

        # compute new means
        self.one.mu = sum(w * d  for (w, d) in zip(left, data)) / one_den
        self.two.mu = sum(w * d  for (w, d) in zip(rigt, data)) / two_den

        # compute new sigmas
        self.one.sigma = sqrt(sum(w * ((d - self.one.mu) ** 2)
                                  for (w, d) in zip(left, data)) / one_den)
        self.two.sigma = sqrt(sum(w * ((d - self.two.mu) ** 2)
                                  for (w, d) in zip(rigt, data)) / two_den)
        # compute new mix
        self.mix = one_den / len(data)


    def iterate(self, N=1, verbose=False):
        "Perform N iterations, then compute log-likelihood"
        for i in range(1, N+1):
            self.Mstep(self.Estep()) #The heart of the algorith, perform E-stepand next M-step
            if verbose:
                print('{0:2} {1}'.format(i, self))
        self.Estep() # to freshen up self.loglike

    def pdf(self, x):
        return (self.mix)*self.one.pdf(x) + (1-self.mix)*self.two.pdf(x)

    def __repr__(self):
        return 'GaussianMixture({0}, {1}, mix={2.03})'.format(self.one,
                                                              self.two,
                                                              self.mix)

    def __str__(self):
        return 'Mixture: {0}, {1}, mix={2:.03})'.format(self.one,
                                                        self.two,
                                                        self.mix)
print("done")
```

```
done
```

- **See the algorithm in action**

In [6]:
```
n_iterations = 20
best_mix = None
best_loglike = float('-inf')
mix = GaussianMixture_self(data)
for _ in range(n_iterations):
    try:
        #train!
        mix.iterate(verbose=True)
        if mix.loglike > best_loglike:
            best_loglike = mix.loglike
            best_mix = mix

    except (ZeroDivisionError, ValueError, RuntimeWarning): # Catch division errors from bad starts, and just throw
        print("one less")
        pass
```

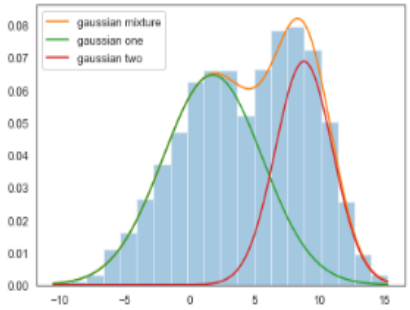
```
 1 Mixture: Gaussian(-8.75384, 1.35749), Gaussian(4.49768, 4.69147), mix=0.00361)
 1 Mixture: Gaussian(-8.4477, 1.685), Gaussian(4.47239, 4.72355), mix=0.00175)
 1 Mixture: Gaussian(-7.95802, 1.91244), Gaussian(4.46376, 4.73467), mix=0.00112)
 1 Mixture: Gaussian(-7.33053, 2.02968), Gaussian(4.45984, 4.7399), mix=0.000852)
 1 Mixture: Gaussian(-6.66907, 2.02428), Gaussian(4.45795, 4.74258), mix=0.000733)
 1 Mixture: Gaussian(-6.06975, 1.90364), Gaussian(4.45718, 4.74393), mix=0.000702)
 1 Mixture: Gaussian(-5.59631, 1.69731), Gaussian(4.45718, 4.74442), mix=0.000734)
 1 Mixture: Gaussian(-5.28418, 1.46631), Gaussian(4.45789, 4.7442), mix=0.000831)
 1 Mixture: Gaussian(-5.12307, 1.27697), Gaussian(4.45936, 4.74326), mix=0.000998)
 1 Mixture: Gaussian(-5.05818, 1.14023), Gaussian(4.46164, 4.74163), mix=0.00124)
 1 Mixture: Gaussian(-5.04384, 1.03771), Gaussian(4.4648, 4.73926), mix=0.00158)
 1 Mixture: Gaussian(-5.0565, 0.95484), Gaussian(4.46902, 4.73603), mix=0.00202)
 1 Mixture: Gaussian(-5.08416, 0.883156), Gaussian(4.47452, 4.73175), mix=0.00259)
 1 Mixture: Gaussian(-5.11979, 0.818027), Gaussian(4.48157, 4.7262), mix=0.00331)
 1 Mixture: Gaussian(-5.15861, 0.757302), Gaussian(4.49033, 4.71921), mix=0.0042)
 1 Mixture: Gaussian(-5.197, 0.700611), Gaussian(4.5008, 4.71078), mix=0.00526)
 1 Mixture: Gaussian(-5.23221, 0.648837), Gaussian(4.5126, 4.70116), mix=0.00645)
 1 Mixture: Gaussian(-5.26261, 0.603408), Gaussian(4.52492, 4.69103), mix=0.00768)
 1 Mixture: Gaussian(-5.28775, 0.565431), Gaussian(4.53658, 4.68135), mix=0.00883)
 1 Mixture: Gaussian(-5.30819, 0.535008), Gaussian(4.54648, 4.67304), mix=0.00981)
```

- **FInally finding best Mixture Gaussian model**

In [7]:
```
n_iterations = 300
n_random_restarts = 4
best_mix = None
best_loglike = float('-inf')
print('Computing best model with random restarts...\n')
for _ in range(n_random_restarts):
    mix = GaussianMixture_self(data)
    for _ in range(n_iterations):
        try:
            mix.iterate()
            if mix.loglike > best_loglike:
                best_loglike = mix.loglike
                best_mix = mix
        except (ZeroDivisionError, ValueError, RuntimeWarning): # Catch division errors from bad starts, and just t
            pass
#print('Best Gaussian Mixture : μ = {:.2}, σ = {:.2} with μ = {:.2}, σ = {:.2}'.format(best_mix.one.mu, best_mix.on

print('Input Gaussian {:}: μ = {:.2}, σ = {:.2}'.format("1", Mean1, Standard_dev1))
print('Input Gaussian {:}: μ = {:.2}, σ = {:.2}'.format("2", Mean2, Standard_dev2))
print('Gaussian {:}: μ = {:.2}, σ = {:.2}, weight = {:.2}'.format("1", best_mix.one.mu, best_mix.one.sigma, best_mi
print('Gaussian {:}: μ = {:.2}, σ = {:.2}, weight = {:.2}'.format("2", best_mix.two.mu, best_mix.two.sigma, (1-best
#Show mixture
sns.distplot(data, bins=20, kde=False, norm_hist=True);
g_both = [best_mix.pdf(e) for e in x]
plt.plot(x, g_both, label='gaussian mixture');
g_left = [best_mix.one.pdf(e) * best_mix.mix for e in x]
plt.plot(x, g_left, label='gaussian one');
g_right = [best_mix.two.pdf(e) * (1-best_mix.mix) for e in x]
plt.plot(x, g_right, label='gaussian two');
plt.legend();
```

```
Computing best model with random restarts...

Input Gaussian 1: μ = 2.0, σ = 4.0
Input Gaussian 2: μ = 9.0, σ = 2.0
Gaussian 1: μ = 1.8, σ = 3.8, weight = 0.62
Gaussian 2: μ = 8.8, σ = 2.2, weight = 0.38
```

```
/home/computer/anaconda3/lib/python3.9/site-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a
deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a fig
ure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)

T.E/SEM VI/CBCGS/AIML
Academic Year: 2022-23

NAME	**PRATHAMESH CHIKANKAR**
BRANCH	**CSE-(AI&ML)**
ROLL NO.	**11**
SUBJECT	**MACHINE LEARNING LAB**
COURSE CODE	**CSL604**
PRACTICAL NO.	**08**
DOP	
DOS	

Program - Output :

- **Importing Libraries -**

In [1]:

```
from tabulate import tabulate
```

- **For AND gate**

In [2]:

```
#inputs
x1 = [0, 0, 1, 1]
x2 = [0, 1, 0, 1]
w1 = [1, 1, 1, 1]
w2 = [1, 1, 1, 1]
t = 2
#output
print("x1    x2    w1   w2     t     0")
for i in range(len(x1)):
    if ( x1[i]*w1[i] + x2[i]*w2[i] ) >= t:
        print(x1[i],'   ',x2[i],'   ',w1[i],'   ',w2[i],'   ',t,'   ', 1)
    else:
        print(x1[i],'   ',x2[i],'   ',w1[i],'   ',w2[i],'   ',t,'   ', 0)
```

```
x1    x2    w1   w2     t     0
0     0     1     1     2     0
0     1     1     1     2     0
1     0     1     1     2     0
1     1     1     1     2     1
```

- **For OR gate**

In [3]:

```
#inputs
x1 = [0, 0, 1, 1]
x2 = [0, 1, 0, 1]
w1 = [1, 1, 1, 1]
w2 = [1, 1, 1, 1]
t = 1
#output
print("x1    x2    w1   w2     t     0")
for i in range(len(x1)):
    if ( x1[i]*w1[i] + x2[i]*w2[i] ) >= t:
        print(x1[i],'   ',x2[i],'   ',w1[i],'   ',w2[i],'   ',t,'   ', 1)
    else:
        print(x1[i],'   ',x2[i],'   ',w1[i],'   ',w2[i],'   ',t,'   ', 0)
```

```
x1    x2    w1   w2     t     0
0     0     1     1     1     0
0     1     1     1     1     1
1     0     1     1     1     1
1     1     1     1     1     1
```

- **For NOT gate**

In [4]:

```
#inputs
x = [0, 1]
w = [-1, -1]
t = 0
#output
print("x       w       t       O")
for i in range(len(x)):
    if ( x[i]*w[i]) >= t:
        print(x[i],'    ',w[i],'    ',t,'    ', 1)
    else:
        print(x[i],'    ',w[i],'    ',t,'    ', 0)
```

```
x       w       t       O
0       -1      0       1
1       -1      0       0
```

- **For NAND gate**

In [5]:

```
#inputs
x1 = [0, 0, 1, 1]
x2 = [0, 1, 0, 1]
w1 = [-1, -1, -1, -1]
w2 = [-1, -1, -1, -1]
t = -2
#output
print("x1    x2    w1     w2      t     O")
for i in range(len(x1)):
    if ( x1[i]*w1[i] + x2[i]*w2[i] ) > t:
        print(x1[i],'    ',x2[i],'    ',w1[i],'    ',w2[i],'    ',t,'    ', 1)
    else:
        print(x1[i],'    ',x2[i],'    ',w1[i],'    ',w2[i],'    ',t,'    ', 0)
```

```
x1    x2    w1     w2      t     O
0     0     -1     -1     -2     1
0     1     -1     -1     -2     1
1     0     -1     -1     -2     1
1     1     -1     -1     -2     0
```

- **For NOR gate**

In [6]:

```
#inputs
x1 = [0, 0, 1, 1]
x2 = [0, 1, 0, 1]
w1 = [1, 1, 1, 1]
w2 = [1, 1, 1, 1]
t = 0
#output
print("x1    x2    w1   w2     t     O")
for i in range(len(x1)):
    if ( x1[i]*w1[i] + x2[i]*w2[i] ) <= t:
        print(x1[i],'   ',x2[i],'   ',w1[i],'   ',w2[i],'   ',t,'   ', 1)
    else:
        print(x1[i],'   ',x2[i],'   ',w1[i],'   ',w2[i],'   ',t,'   ', 0)
```

```
x1    x2    w1   w2     t     O
0     0     1     1     0     1
0     1     1     1     0     0
1     0     1     1     0     0
1     1     1     1     0     0
```

- **For EXOR gate**

In [7]:

```
#inputs
x1 = [0, 0, 1, 1]
x2 = [0, 1, 0, 1]
w1 = [1, 1, 1, 1]
w2 = [1, 1, 1, 1]
w3 = [1, 1, 1, 1]
w4 = [-1, -1, -1, -1]
w5 = [-1, -1, -1, -1]
w6 = [1, 1, 1, 1]
t1 = [0.5,0.5,0.5,0.5]
t2 = [-1.5,-1.5,-1.5,-1.5]
t3 = [1.5,1.5,1.5,1.5]
def XOR (a, b):
    if a != b:
        return 1
    else:
        return 0
#output
print('x1    x2    w1    w2    w3     w4    w5     w6    t1        t2      t3     O
for i in range(len(x1)):
    print(x1[i],'   ',x2[i],'   ',w1[i],'   ',w2[i],'   ',w3[i],'   ',w4[i],'   ',w
```

```
x1    x2    w1    w2    w3     w4    w5     w6    t1        t2      t3
O
0     0     1     1     1     -1    -1     1     0.5     -1.5     1.5
0
0     1     1     1     1     -1    -1     1     0.5     -1.5     1.5
1
1     0     1     1     1     -1    -1     1     0.5     -1.5     1.5
1
1     1     1     1     1     -1    -1     1     0.5     -1.5     1.5
0
```

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)

T.E/SEM VI/CBCGS/AIML
Academic Year: 2022-23

NAME	**PRATHAMESH CHIKANKAR**
BRANCH	**CSE-(AI&ML)**
ROLL NO.	**11**
SUBJECT	**MACHINE LEARNING LAB**
COURSE CODE	**CSL604**
PRACTICAL NO.	**09**
DOP	
DOS	

Program - Output :

- **Importing Libraries -**

In [1]:

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

- **Importing data as Pandas DataFrame**

In [6]:

```
data=pd.read_csv('/home/computer/Documents/iris.csv')
data.columns=['Sepal_len_cm','Sepal_wid_cm','Petal_len_cm','Petal_wid_cm','Type']
data.head(10)
```

Out[6]:

	Sepal_len_cm	Sepal_wid_cm	Petal_len_cm	Petal_wid_cm	Type
0	4.9	3.0	1.4	0.2	0
1	4.7	3.2	1.3	0.2	0
2	4.6	3.1	1.5	0.2	0
3	5.0	3.6	1.4	0.2	0
4	5.4	3.9	1.7	0.4	0
5	4.6	3.4	1.4	0.3	0
6	5.0	3.4	1.5	0.2	0
7	4.4	2.9	1.4	0.2	0
8	4.9	3.1	1.5	0.1	0
9	5.4	3.7	1.5	0.2	0

- **Training (using Sigmoid function as the activation function)**

In [7]:

```
def activation_func(value):    #Tangent Hypotenuse
    #return (1/(1+np.exp(-value)))
    return ((np.exp(value)-np.exp(-value))/(np.exp(value)+np.exp(-value)))
```

In [8]:

```
def perceptron_train(in_data,labels,alpha):
    X=np.array(in_data)
    y=np.array(labels)
    weights=np.random.random(X.shape[1])
    original=weights
    bias=np.random.random_sample()
    for key in range(X.shape[0]):
        a=activation_func(np.matmul(np.transpose(weights),X[key]))
        yn=0
        if a>=0.7:
            yn=1
        elif a<=(-0.7):
            yn=-1
        weights=weights+alpha*(yn-y[key])*X[key]
        print('Iteration '+str(key)+': '+str(weights))
    print('Difference: '+str(weights-original))
    return weights
```

- **Testing and Scoring**

In [9]:

```
def perceptron_test(in_data,label_shape,weights):
    X=np.array(in_data)
    y=np.zeros(label_shape)
    for key in range(X.shape[1]):
        a=activation_func((weights*X[key]).sum())
        y[key]=0
        if a>=0.7:
            y[key]=1
        elif a<=(-0.7):
            y[key]=-1
    return y
```

In [10]:

```
def score(result,labels):
    difference=result-np.array(labels)
    correct_ctr=0
    for elem in range(difference.shape[0]):
        if difference[elem]==0:
            correct_ctr+=1
    score=correct_ctr*100/difference.size
    print('Score='+str(score))
```

- **Dividing dataframe**

In [11]:

```
# Dividing DataFrame "data" into "d_train" (60%) and "d_test" (40%)
divider = np.random.rand(len(data)) < 0.70
d_train=data[divider]
d_test=data[~divider]
```

In [12]:

```
# Dividing d_train into data and labels/targets
d_train_y=d_train['Type']
d_train_X=d_train.drop(['Type'],axis=1)

# Dividing d_train into data and labels/targets
d_test_y=d_test['Type']
d_test_X=d_test.drop(['Type'],axis=1)
```

- **Getting learning rate**

In [13]:

```
# Learning rate
alpha = 0.01

# Train
weights = perceptron_train(d_train_X, d_train_y, alpha)
```

```
Iteration 0: [0.99990004 0.07493098 0.36786327 0.41037301]
Iteration 1: [1.04690004 0.10693098 0.38086327 0.41237301]
Iteration 2: [1.09290004 0.13793098 0.39586327 0.41437301]
Iteration 3: [1.14290004 0.17393098 0.40986327 0.41637301]
Iteration 4: [1.19690004 0.21293098 0.42686327 0.42037301]
Iteration 5: [1.24690004 0.24693098 0.44186327 0.42237301]
Iteration 6: [1.30090004 0.28593098 0.45486327 0.42637301]
Iteration 7: [1.35190004 0.32093098 0.46886327 0.42937301]
Iteration 8: [1.40890004 0.35893098 0.48586327 0.43237301]
Iteration 9: [1.45990004 0.39693098 0.50086327 0.43537301]
Iteration 10: [1.50590004 0.43293098 0.51086327 0.43737301]
Iteration 11: [1.55690004 0.46593098 0.52786327 0.44237301]
Iteration 56: [5.97290004 2.67793098 3.38386327 1.30637301]
Iteration 57: [6.09890004 2.72393098 3.47186327 1.33237301]
Iteration 58: [6.21090004 2.78393098 3.55386327 1.35837301]
Iteration 59: [6.32090004 2.83393098 3.63386327 1.38437301]
Iteration 60: [6.43090004 2.88593098 3.72186327 1.40837301]
Iteration 61: [6.55290004 2.94593098 3.81386327 1.43637301]
Iteration 62: [6.66890004 2.99793098 3.89386327 1.46037301]
Iteration 63: [6.76890004 3.04393098 3.95086327 1.48037301]
Iteration 90: [7.21090004 3.26593098 4.27186327 1.57837301]
Iteration 91: [7.21090004 3.26593098 4.27186327 1.57837301]
Iteration 92: [7.21090004 3.26593098 4.27186327 1.57837301]
Iteration 93: [7.21090004 3.26593098 4.27186327 1.57837301]
Iteration 94: [7.21090004 3.26593098 4.27186327 1.57837301]
Iteration 95: [7.21090004 3.26593098 4.27186327 1.57837301]
Iteration 96: [7.21090004 3.26593098 4.27186327 1.57837301]
Iteration 97: [7.21090004 3.26593098 4.27186327 1.57837301]
Difference: [6.26  3.221 3.918 1.17 ]
```

- **Testing and Calculating score**

In [14]:

```
# Test
result_test=perceptron_test(d_test_X,d_test_y.shape,weights)
```

In [15]:

```
# Calculate score
score(result_test,d_test_y)
```

```
Score=35.294117647058826
```

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)

T.E/SEM VI/CBCGS/AIML
Academic Year: 2022-23

NAME	**PRATHAMESH CHIKANKAR**
BRANCH	**CSE-(AI&ML)**
ROLL NO.	**11**
SUBJECT	**MACHINE LEARNING LAB**
COURSE CODE	**CSL604**
PRACTICAL NO.	**10**
DOP	
DOS	

Program - Output :

- **Importing Libraries -**

In [1]:
```
# importing required libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

- **Importing or loading the dataset**

In [4]:
```
# importing or loading the dataset
dataset = pd.read_csv('/home/computer/Downloads/Wine.csv')

# distributing the dataset into two components X and Y
X = dataset.iloc[:, 0:13].values
y = dataset.iloc[:, 13].values
```

- **Splitting the dataset into the Training set and Test set**

In [5]:
```
# Splitting the X and Y into the
# Training set and Testing 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, ra
```

- **Feature Scaling**

In [6]:
```
# performing preprocessing part
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()

X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

- **Applying the PCA function into the training and testing set for analysis**

In [7]:
```
# Applying PCA function on training
# and testing set of X component
from sklearn.decomposition import PCA

pca = PCA(n_components = 2)

X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)

explained_variance = pca.explained_variance_ratio_
```

- **Fitting Logistic Regression To the training set**

In [8]:
```
# Fitting Logistic Regression To the training set
from sklearn.linear_model import LogisticRegression

classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train)
```

Out[8]:
```
LogisticRegression(random_state=0)
```

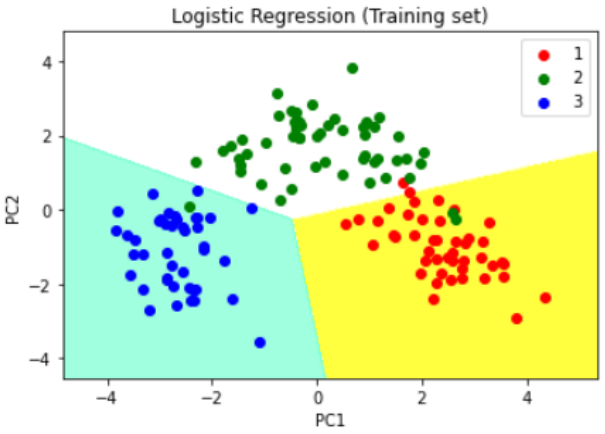
- **Predicting the test set result and making confusion matrix**

In [9]:
```
# Predicting the test set result using
# predict function under LogisticRegression
y_pred = classifier.predict(X_test)
```

In [10]:
```
# making confusion matrix between
# test set of Y and predicted value.
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred)
```

- **Predicting the training set result**

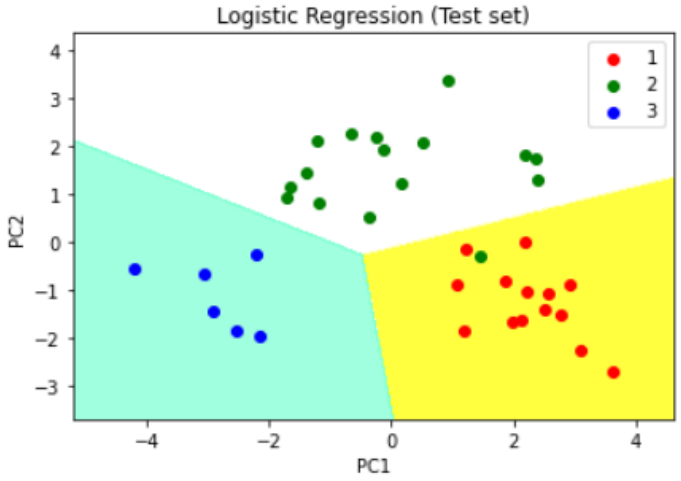
In [11]:
```
# Predicting the training set
# result through scatter plot
from matplotlib.colors import ListedColormap

X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1,
                               stop = X_set[:, 0].max() + 1, step =
                               np.arange(start = X_set[:, 1].min() -
                               stop = X_set[:, 1].max() + 1, step =

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(),
                     X2.ravel()]).T).reshape(X1.shape), alpha = 0.75,
                     cmap = ListedColormap(('yellow', 'white', 'aquamarine

plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y_set)):
        plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                                c = ListedColormap(('red', 'green', 'blue'))(

plt.title('Logistic Regression (Training set)')
plt.xlabel('PC1') # for Xlabel
plt.ylabel('PC2') # for Ylabel
plt.legend() # to show legend

# show scatter plot
plt.show()
```



- **Visualizing the Test set results**

In [12]:

```
# Visualising the Test set results through scatter plot
from matplotlib.colors import ListedColormap

X_set, y_set = X_test, y_test

X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1,
                               stop = X_set[:, 0].max() + 1, step =
                               np.arange(start = X_set[:, 1].min() -
                               stop = X_set[:, 1].max() + 1, step =

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(),
                     X2.ravel()]).T).reshape(X1.shape), alpha = 0.75,
                     cmap = ListedColormap(('yellow', 'white', 'aquamarine

plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y_set)):
        plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                                c = ListedColormap(('red', 'green', 'blue'))(

# title for scatter plot
plt.title('Logistic Regression (Test set)')
plt.xlabel('PC1') # for Xlabel
plt.ylabel('PC2') # for Ylabel
plt.legend()

# show scatter plot
plt.show()
```



Name :- Prathamesh S. Chikankar

Roll No :- AIML11 Branch :- PSE-I(AIAML)

Subject :- Machine Learning [ML] Topic :- Assignment No. 01

Year :- TE Date :- March 23

Sign :- Prathamesh

1. Define machine learning

⇨ A machine that is intellectually capable as much as humans, have always attracted writers and early computer scientists who were excited about artificial intelligence & machine learning

→ machine learning is defined by Tom mitchell as "A program learns from experience 'E' with respect to some class of tasks 'T' and performance measure 'P', if its performance on tasks in 'T' as measured by 'P' improves with 'E'. Here 'E' represents the past experienced data and 'T' represents the tasks such as predict, classification, etc.. Example of 'P', we might increase accuracy in predict.

⇨ Explain how supervised learning is different from unsupervised Learning

	Supervised learning	Unsupervised learning
1)	Supervised learning algorithms are trained using labelled data.	Unsupervised learning algorithms are trained using unlabelled data.
2)	Takes direct feedback to check if it is predicting correct output or not.	Unsupervised learning model does not take any feedback
3)	Supervised learning model predicts the output.	Unsupervised learning models finds the hidden patterns in data.
4)	Input data is provided to model along with output in supervised.	Only input data is provided to the model in unsupervised learning
5)	The goal of supervised learning is to train model so that is can predict output when it is given new data.	The goal of the unsupervised learning is to find hidden patterns and useful insights from the unknown dataset.
6)	Supervised learning needs supervision to train the model.	Unsupervised learning doesn't need any supervisor to train the model.
7)	Supervised learning model produces an accurate result.	Unsupervised learning model may give less accurate result as compared to supervised learning

2. Explain the steps of developing machine learning application.

1] Collection of Data: you could collect samples from website and extracting data. - from RSS feed or an API
- publically available data

2] preparation of the input data:
- Once you have the input data, you need to check whether it is in a usable format or not
- Some algorithm accepts features in a special format.

3] Analyse the input data: looking at the data you have passed in a text editor to check collection and preparation of input data steps are properly working and you don't have a bunch of empty values.
- plotting data in 1, 2 & 3 dimensions can also help.

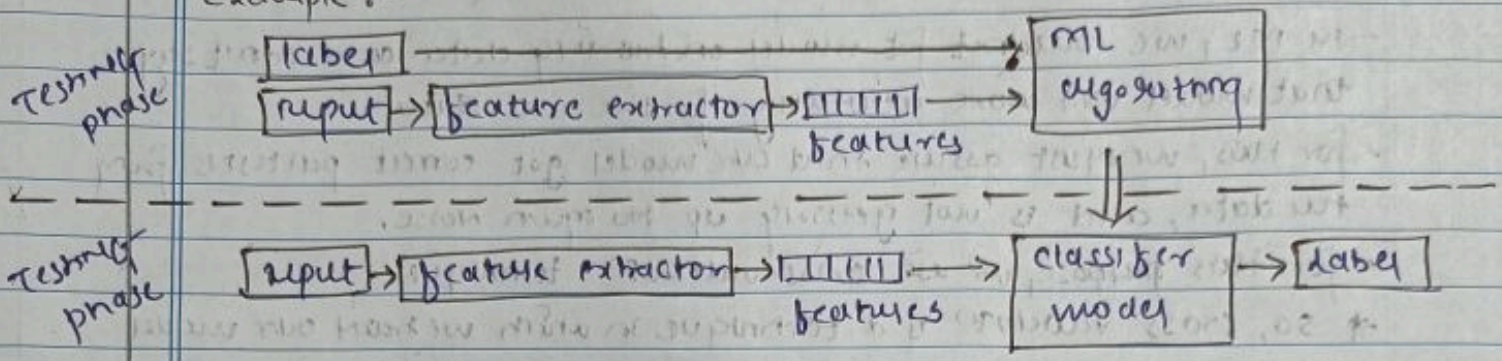
4] The importance of this step is that it makes you understand that you don't have any garbage value coming in.

5] Train the algorithm: Good clean data from the first two steps is given as input to the algorithm. Extracts information or knowledge. This knowledge is mostly used storing format that readily useable by machine for next 2 steps. Complete data is used in next step.

6] Test the algorithm: In this step the information learned in the previous step is used. When you are checking an algorithm, you will test it to find out whether it works properly or not. In Supervised case, you have some known values, & in unsupervised measures to evaluate. mostly prb occur in step 1 for collection or preparation of data; need to check.

7] Use it: In this step a real program is developed to do some task, & once again it is checked if all previous steps work properly as you expected. you might encounter some new data & have to revisit step 1-5

example :-



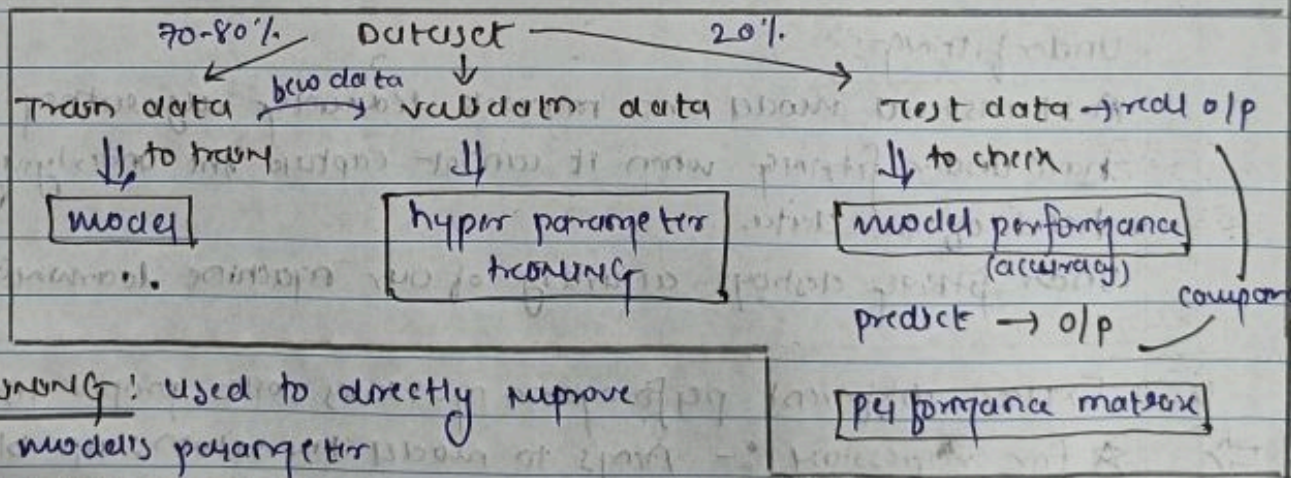
3. Explain Training, Testing and validation datasets

When a labelled dataset is used to train machine learning models, it is common to break up dataset into three parts.

Train, Test and validation dataset

① regression
② classification

dataset → I/p: f1 f2 f3 f4 → O/p



Training: used to directly improve model's parameter

validation: used to evaluate model's performance while optimizing the model's hyperparameter.

Test: used to evaluate a model after hyperparameter optimization is complete.

PTO

Cross validation:

- In ML, we couldn't fit model on training data and can't say that model will work accurately for the real data.
- for this, we must assume that our model got correct patterns from the data, and is not getting up too much noise.
- for this purpose, we use cross-validation technique.

* So, cross validation is a technique in which we train our model using subset of dataset & then evaluate using complementary subset of the dataset.

Overfitting:

A statistical model is said to be over fitted, when we train it with a lot of data.
When model gets trained with so much of data, it starts learning from the noise and inaccurate data entries in our data set.

Under fitting:

A statistical model or a machine learning algorithm said to have under fitting when it cannot capture the underlying trend of the data.
Under fitting destroys accuracy of our machine learning model.

---

4. Explain different performance matrics with proper illustration.

⇒ * For regression :- Aims to model the relationship b/w a certain no. of features and a continous target variable.

- performance matrics:

① Mean absolute error (MAE):

$$MAE = \frac{1}{M} \sum_{i=1}^{M} |y_i - \hat{y}_i|$$

... y_i = actual expected o/p
$\hat{y}_i$ = model's predict...

PTO

② Mean squared error (MSE):

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

∴ error term is squared and more sensitive

③ Root mean squared error (RMSE)

$$RMSE = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$$

∴ take square root of MSE, give ease to RMSE.

④ R-squared

$$R_2 = 1 - \frac{SS_{RES}}{SS_{TOT}} \Rightarrow 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}$$

Calculated by dividing sum of squares of residuals (SS_{res}) from regression model by total sum of squares (SS_{tot}) of errors from the average model & then subtract it from 1.
Also called as coefficient of determinant

⑤ Adjusted R-squared

$$\text{Adjusted } R^2 = 1 - \frac{(1-(R)^2)(N-1)}{N-p-1}$$

∴ N – total sample size
p – no. of predictors

☆ performance metrics for classification :-
Based on the training set of data containing records whose class label is known.

① Accuracy

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + FN + TN}$$

∴ TN → True Negative
TP → True +ve
FP → false +ve
FN → false −ve

ratio of no. of correct predictions & total no. of predictn.

② precession and Recall

$$\text{precision} = \frac{TP}{TP + FP} \qquad \text{Recall} = \frac{TP}{TP + FN}$$

- recall is a useful metric in case of cancer detection, where we want to minimize no. of false -ve.
- precision is useful when we want to reduce the number of false +ve. eg:- email received in spam or not.

③ specificity :

$$\text{specificity} = \frac{\text{True Negative}}{TN + FP}$$

④

∴ f1 score

$$f1 = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

⑤ AUC-ROC

area under the curve - receiver operating characteristics for checking any classification characteristic models performance... plotted b/w FPR (x-axis) and TPR (y-axis)

$$TPR = \frac{TP}{FP + FN} \quad ; \quad FPR = \frac{FP}{FP + TN}$$

★ Kappa statistics :-

measures inter-rater agreement for qualitative (categorical) items.

$$K = \frac{P_o - P_e}{1 - P_e}$$

... P_o → relative observed agreement among raters
P_e → hypothetical prob. of chance agreement.

5. write a short note on :-

a) Issues in Machine learning

① Inadequate training data

- noisy data and - Generalizing of o/p data
- Incorrect data

PTO

② poor quality of data
- inadequate training data leads to less accuracy in classification and low quality results.

③ non-representative training data
- Results in a inaccurate (less accurate) predictions.

④ overfitting and underfitting

⑤ monitoring and maintenance

⑥ Getting Bad recommendations

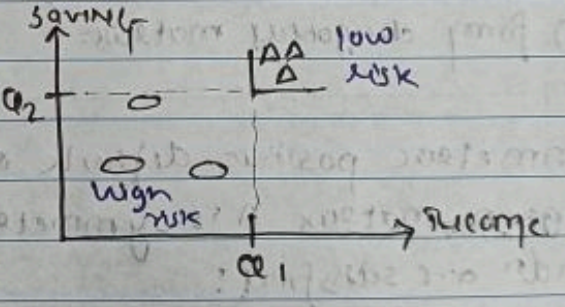
⑦ Data Bias
- when certain elements of dataset are heavily weighted...

---

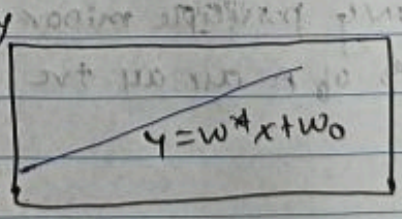
b) machine learning approaches:

① Learning association
- Market Basket Analysis if A then B
- if Bread, then milk as well

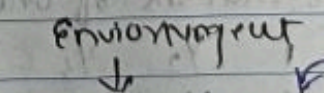
② classification:
- If income > Q_1 & savings > Q_2 then low-risk else high-risk



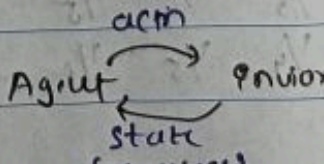
③ Regression
- when o/p is form of numeric
i.e. $y = w^T x + w_0$



④ Unsupervised learning
- only inputs, no prior knowledge
- clustering (grp of once)



⑤ Reinforcement learning
- sequence of actn (reward based)



PTO

c) Diagonalization of a matrix
- squared matrix that can be transformed into a diagonal matrix, i.e. matrix with non-diagonal elements as zeros.
- mathematical relation

$A = PDP^{-1}$ (or) $D = P^{-1}AP$

↑ to be diagonalized ↑ columns are eigen vector

example: $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

* Methods of diagonalizing a matrix
- Based on finding eigen values and eigen vectors
- steps
  1) find eigen values
  2) find eigen vectors associated with eigen values
  3) check eigen values are distinct
  4) form diagonal matrix.

---

6. Symmetric positive definite matrices explain with example...
- A given matrix A is symmetric +ve definite if the following condn are satisfied:

(1) Every leading principle minor det(Ak) is +ve
(2) eigen values of A are all +ve

[Remark
- $\det(A) > 0$
- doesn't mean A is symmetric +ve always.

⇨ Example:

① hilbert matrix of order 4×4

$$H_4 = \begin{bmatrix} 1 & 1/2 & 1/3 & 1/4 \\ 1/2 & 1/3 & 1/4 & 1/5 \\ 1/3 & 1/4 & 1/5 & 1/6 \\ 1/4 & 1/5 & 1/6 & 1/7 \end{bmatrix}$$

properties of SPDM's
→ A^{-1} is +ve definite
→ $x^2 = A$
→ A has unique factorization
i.e. $A = R^T R$ upper triangular with +ve diagonal elements

② pascal matrix

$$P_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \\ 1 & 3 & 6 & 10 \\ 1 & 4 & 10 & 20 \end{bmatrix}$$

③ test-matrix (wilson-matrix)

$$W = \begin{bmatrix} 5 & 7 & 6 & 5 \\ 7 & 10 & 8 & 7 \\ 6 & 8 & 10 & 9 \\ 5 & 7 & 9 & 10 \end{bmatrix}$$

⇒ Another definition

let A be nxn matrix A is symmetric +ve definite if:

① it is symmetric (A is equal to its transpose A^T) and

② $x^T A x > 0$ for all non-zero vector x.

*Remark:

$a_{ij} > 0$ for all i and
$a_{ij} < \sqrt{a_{ii} \cdot a_{jj}}$ for all $i \neq j$

} → +ve diagonal elements &
→ diagonally dominant
i.e $a_{ii} > \sum_{j \neq i} |a_{ij}|$ for all i.

7. Explain the terms:

Norms :- A norm is a way to measure the size of vector, a matrix or a tensor

in other words, norms are a class of functions that enable to quantify the magnitude of a vector.

⇒ L1 and L2 Norm

L1 norm is calculated as sum of absolute values of vector

L2 norm is calculated as square-root of sum of squared vector values

PTO

Euclidian Norm:

$x = (x_1, x_2 \cdots x_n)$ is given as,

$\|x\| = \sqrt{x_1^2 + x_2^2 + \cdots x_n^2}$ ← gives distance from origin...

Inner product:

- Generalization of dot product
- In vector space → way to represent multiply vectors together and the result of this multiplication is scalar.

⇒ Every inner product space induces a Norm, called its canonical form, and is denoted by $\|x\| = \sqrt{\langle x, x \rangle}$.

⇒ With this Norm, every innerproduct becomes a Normed vector space.

Length of vector:

Distance of a vector in standard position from the origin.
Also known as magnitude of a vector.

⇒ Suppose $\bar{u} = xi + yj$, then length of $\bar{u}$,

$|\bar{u}| = \sqrt{x_1^2 + y_1^2}$

⇒ If vector has 3 components

$\bar{u} = xi + yj + zk$ then $|\bar{u}| = \sqrt{x^2 + y^2 + z^2}$

Determinant:

- Scalar value
- Function of entries of square matrix.
- Unique map represented by the matrix → isomorphism

eg:- In case of 2×2 matrix, $|A| = \begin{bmatrix} a & b \\ c & d \end{bmatrix} = ad - bc$

Trace:

Trace of square matrix A, denoted by $tr(A)$ is defined as be the

PTO

Sum of elements on main diagonal of A
- only defined for a square matrix (n×n), Thus,

$$tr(A) = \sum_{i=1} a_{ii} = a_{11} + a_{22} + \cdots a_{nn}$$

... a_{ii} → ith row & ith column ...

8. find the Eigen values & Eigen vectors of the following matrix.

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 0 & -2 & 6 \\ 0 & 0 & -3 \end{bmatrix} \qquad |A| = 6$$

⇨ Characteristic eqn is $|A - \lambda I| = 0$

$$\begin{vmatrix} 1-\lambda & 2 & 3 \\ 0 & -2-\lambda & 6 \\ 0 & 0 & -3-\lambda \end{vmatrix} = 0$$

λ^3 - (sum of diagonal elements) λ^2 + (sum of minor diagonal elements) $\lambda - |A| = 0$

$\lambda^3 - (-4)\lambda + (6 + (-3) + (-2))\lambda - 6 = 0$

$\lambda^3 + 4\lambda + \lambda - 6 = 0$

$\lambda = 1, -2, -3$ are eigen values

for $\lambda = 1$ $[A - \lambda I][X] = 0$

$$\begin{bmatrix} 0 & 2 & 3 \\ 0 & -3 & 6 \\ 0 & 0 & -4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = 0$$

using Cramer's rule,

$$\frac{x_1}{\begin{vmatrix} 2 & 3 \\ -3 & 6 \end{vmatrix}} = \frac{-x_2}{\begin{vmatrix} 0 & 3 \\ 0 & 6 \end{vmatrix}} = \frac{x_3}{\begin{vmatrix} 0 & 2 \\ 0 & -3 \end{vmatrix}} \Rightarrow \frac{x_1}{21} = \frac{-x_2}{0} = \frac{x_3}{0}$$

$$\Rightarrow x_1/1 = -x_2/0 = -x_3/0$$

$$v_1 \Rightarrow \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

for $\lambda = -2$

$$\begin{bmatrix} 3 & 2 & 3 \\ 0 & 0 & 6 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = 0$$

$$\frac{x_1}{\begin{vmatrix} 2 & 3 \\ 0 & 6 \end{vmatrix}} = \frac{-x_2}{\begin{vmatrix} 3 & 3 \\ 0 & 6 \end{vmatrix}} = \frac{x_3}{\begin{vmatrix} 3 & 2 \\ 0 & 0 \end{vmatrix}} \Rightarrow \frac{x_1}{12} = \frac{-x_2}{18} = \frac{x_3}{0}$$

$$\Rightarrow \frac{x_1}{2} = \frac{-x_2}{3} = \frac{x_3}{0}$$

$$v_2 \Rightarrow \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ -3 \\ 0 \end{bmatrix}$$

for $\lambda = -3$

$$\begin{bmatrix} 4 & 2 & 3 \\ 0 & 1 & 6 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = 0$$

$$\frac{x_1}{\begin{vmatrix} 2 & 3 \\ 1 & 6 \end{vmatrix}} = \frac{-x_2}{\begin{vmatrix} 4 & 3 \\ 0 & 6 \end{vmatrix}} = \frac{x_3}{\begin{vmatrix} 4 & 2 \\ 0 & 1 \end{vmatrix}} \Rightarrow \frac{x_1}{9} = \frac{-x_2}{24} = \frac{x_3}{4}$$

$$\Rightarrow \frac{x_1}{9} = \frac{-x_2}{24} = \frac{x_3}{4}$$

$$v_3 = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 9 \\ -24 \\ 4 \end{bmatrix}$$

//

9. Explain Singular value Decomposition (SVD) with example

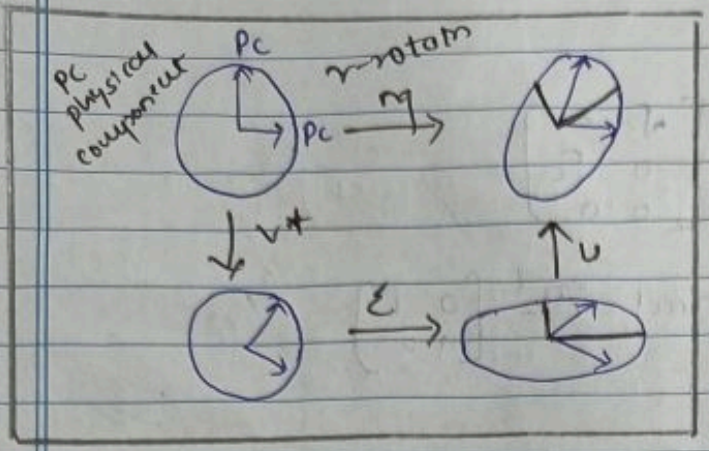
SVD provides another way to factorize the matrix into the singular vectors and singular values.

SVD is used widely both in calculation of matrix operation such as matrix inverse, but also a data Reduction method.

It is a method of decomposing a matrix into 3 other matrices

$$M = U \cdot \Sigma V^*$$

- M → matrix
- U → orthogonal
- Σ → diagonal
- V^* → orthogonal



* property:

→ $a_{ij} = \sum_{k=1}^{n} u_{ik} \varepsilon_k v_{jk}^*$

→ $\varepsilon_{i+1} \leq \varepsilon_i$,

$U^T U = V V^T = I$ (orthogonal)

→ $AA^T U = U S^2$, $A^T A V = V S^2$

→ $U = AVS^{-1}$, $A^T = VSU^T$

Example: how to find V, Σ, U?

Let take this example, $A = \begin{bmatrix} 1 & 1 \\ 0 & 1 \\ -1 & 1 \end{bmatrix}$

Soln:- Step 1: find $A^T A$

$$\begin{bmatrix} 1 & 0 & -1 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 0 & 1 \\ -1 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix}$$

Step 2: find eigen value of $A^T A$

$$\begin{vmatrix} \lambda-2 & 0 \\ 0 & \lambda-3 \end{vmatrix} \quad \lambda_1 = 3 \text{ and } \lambda_2 = 2$$

Step 3: find singular value the singular value

$\sigma_1 = \sqrt{\lambda_1}$ and $\sigma_2 = \sqrt{\lambda_2}$

$\sigma_1 = \sqrt{3}$ and $\sigma_2 = \sqrt{2}$

Step 4: find Eigen vector for each value

for $\lambda = 3 \Rightarrow v_1 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \Rightarrow \Lambda_1$

for $\lambda = 2 \Rightarrow v_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \Rightarrow \Lambda_2$

Step 5: calculate U, V and Σ matrices

$U = \{u_1, u_2, \ldots\} = \{ 1/\sigma_1 A v_1, 1/\sigma_2 A v_2 \ldots \}$

$V = [\Lambda_1, \Lambda_2]$

$$U = \begin{bmatrix} 1/\sqrt{3} & 1/\sqrt{2} & 1/\sqrt{6} \\ 1/\sqrt{3} & 0 & -2/\sqrt{6} \\ 1/\sqrt{3} & -1/\sqrt{2} & 1/\sqrt{6} \end{bmatrix}$$

$$\Sigma = [c_1 \; c_2]\, I$$

$$\Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \sqrt{3} & 0 \\ 0 & \sqrt{2} \end{bmatrix} = \begin{bmatrix} \sqrt{3} & 0 \\ 0 & \sqrt{2} \\ 0 & 0 \end{bmatrix}$$

$V = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$, But we need $V^T = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$

applications:

- SVD has many application in various fields, including machine learning, signal processing, image compression, and recommendation system.
- for example, SVD is used in recommendation system to factorize a user-item rating matrix into two low-dimensional matrices representing user and items features, respectively.
- SVD can also be used for image compression by reducing the dimensionality of the image matrix while preserving its important features.
- additionally, SVD is used in data analysis to identify the most important features in a dataset and reduce its dimensionality.

★ Global applications:

- Image compression
- Market Basket analysis
- political spectrum analysis
- SVD and PCA for gene expression data
- latent semantic indexing (LSI) for web document search.

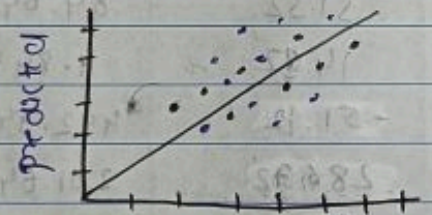
10. Explain Linear models

⇒ The linear model is one of the most straight forward models in machine learning. It is the Building block for many complex machine learning algorithms, including deep neural networks. Linear models predict the target variable using a linear function of the input features.

* mathematically, it can be written as, $y = w^T x$

x => Feature matrix, y => target variable, w => learned weight vector.

- we apply transformation function or threshold for the classification problem to convert continuous-valued variable y into discrete category.
- Easy to implement & interpret & are helpful in solving many real-life use cases.



- training set
- test set

fig:- Linear model

Linear regressions:

Linear regression is a statistical approach that predicts the result of response variable by combining numerous influencing factors. It attempts to represent the linear connection between features (independent variables) and the target (dependent variable). The cost function enables us to find the best possible value for model parameters. A detailed discussion given below

* mathematically it can be written as,

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots \beta_n x_n + \varepsilon \dots$$

β_0 => intercept, $\beta_1, \beta_2, \beta_n$ are coefficient & independent variables $x_1, x_2 \dots x_n$

ε => Error.

PTO

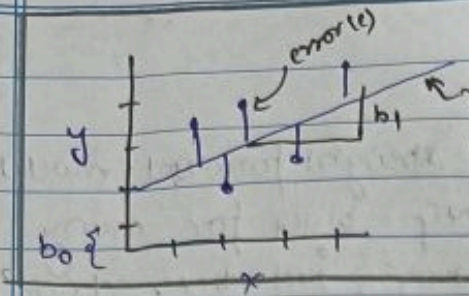


fig:- linear regression

→ several assumptn of linear regression
* linearity * independence, * Normality, etc...

11. The data for midterm & final exam grades obtained for students in ML subject are given in table below. Use method of least square using regression to predict final exam grade of a student received 94 marks in midterm...

midterm (x)	final (y)	$x_i - \bar{x}$	$y_i - \bar{y}$	$(x_i - \bar{x})(y_i - \bar{y})$	$(x_i - \bar{x})^2$
72	84	0.2	12.6	2.52	0.04
50	53	-21.8	-18.4	401.2	475.24
81	77	9.2	5.6	51.52	84.64
74	78	2.2	6.6	14.52	4.84
94	90	22.2	18.6	412.92	492.84
86	75	14.2	3.6	-51.12	201.64
59	49	-12.8	-22.4	286.72	163.84
83	79	11.2	7.6	85.12	125.44
86	77	14.2	5.6	79.52	201.64
33	52	-38.8	-19.4	752.72	1505.44
$\bar{n}$ 71.8	71.4			213.788	325.596

* normal eqn are :-

$$b_1 = \frac{(x_i - \bar{x})(y_i - \bar{y})}{(x_i - \bar{x})^2} = \frac{213.788}{325.596} \Rightarrow 0.6566$$

$b_0 = 71.4 - (0.655 \times 71.8) \Rightarrow 24.26$ $\quad \because b_0 = \bar{y} - b_1(\bar{x})$

* regression eqn :- (To predict exam grade at 94)

$y = b_1(x) + b_0 \Rightarrow (0.6566 \times 94) + 24.26 \Rightarrow 85.98$ //

Name :- Prathamesh S. Chikankar

Roll No. :- AIML11 Branch :- CS E- (AI & ML)

Year :- TE Subject :- Machine Learning (ML)

Topic :- Assignment No. 02

Sign :- Prathu Date :- August '23

1. Explain support vector machine & its kernels.

> A support vector machine (SVM) is a type of supervised machine learning algorithm used for classification & regression analysis.

- It works by finding a hyperplane in a high-dimensional space that maximally separates the different classes of data points.

> SVMs can be used for both linear and nonlinear classification by using different kernel functn. A kernel functn transforms the data into a higher-dimensional feature space where a linear hyperplane can separate the data.

- The most commonly used kernels in SVMs are:

> Linear kernel: This is the simplest kernel functn and is used for linearly separable data. It finds a linear hyperplane that used for separating the data.

> polynomial kernel: This kernel functn is used for data that is not linearly separable. It maps the data into higher-dimensional feature space using a polynomial functn.

> Gaussian kernel: where prior knowledge is not available, for this type of application Gaussian kernel is used.

$$K(x, y) = \exp\left(-\frac{\|x-y\|^2}{2\sigma^2}\right)$$

> Sigmoid kernel: used as proxy for neural NW.

$$k(x, y) = \tanh(ax^d y + c)$$

> Linear kernel: $K(x, x') = x^T x'$

> polynomial kernel: $k(x_j, x_k) = (x_j \cdot x_k + 1)^p$

How margin is computed?

we would like to find the data point nearest to the separating hyperplane and also make sure that this point should be far away from

PTO

the separating line as possible. This is called margin. We would like to find greatest possible margin, because if we learned our classifier on limited data or made a mistake, we would want it to be as robust as possible.

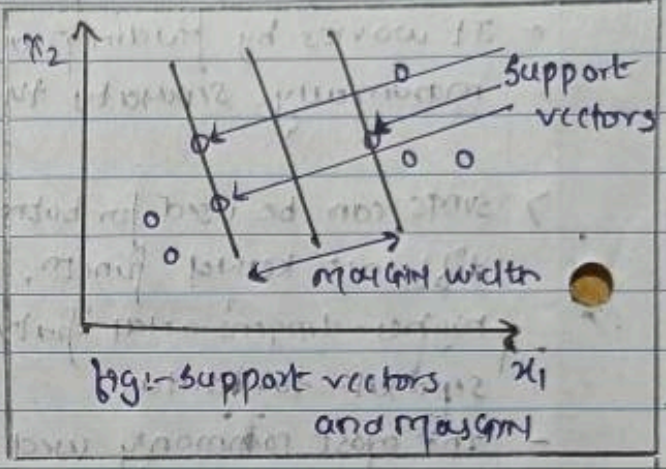


fig:- Support vectors and margin

Support vectors are the points which are nearest to the separating hyperplane. We have to maximize the distance b/w the support vectors and the separating line.

Distance b/w a hyperplane (w, b) & a point x is calculated as,

$$\text{Distance} = \frac{|w^T x + b|}{\|w\|}$$

How optimal hyperplane is decided?

> A hyperplane is formally defined by $f(x) = w^T x + b$

$w \Rightarrow$ weight vector, $b \Rightarrow$ bias.

> By scaling : $|w^T x + b| = 1$

> The general numerator $= 1$, $\therefore$ Distance$_{SV} = \frac{|w^T x + b|}{\|w\|} = \frac{1}{\|w\|}$

> Margin is twice, the distance to nearest samples ... $\therefore M = 2/\|w\|$

2. Explain Hebbian learning rule with example.

> Hebbian learning is a type of unsupervised learning rule that explains how the strength of the connection b/w neurons in the brain can change as a result of experience.

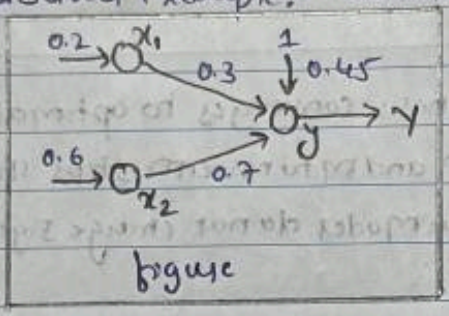
> It is based on the principle that when two neurons fire together, the strength of the connection between them is increased.

> For example, consider a neural n/w with two input neuron and one output neuron. If the input neuron fire together and cause the output

PTO

neuron to fire, the strength of the connection between the input and output neurons is increased. This means that in the future, when the same data input neurons fire together, the output neuron is more likely to fire.

*Solved example:- calculating net input for NN shown in fig with bias.



figure

Soln:- net consists of two input neurons, a bias and an output neuron.

inputs are $[x_1, x_2] = [0.2, 0.6]$ & weights are $[w_1, w_2] = [0.3, 0.7]$

since bias is included $b = 0.45$ & bias input x_0 is equal to 1, then net i/p is given by,

$y_{in} = b + x_1w_1 + x_2w_2 \Rightarrow 0.45 + (0.2)(0.3) + (0.6)(0.7)$

$\Rightarrow 0.45 + 0.06 + 0.42 \Rightarrow 0.93$ ... is the Net i/p.

---

3. E-M algorithm : (expectation maximization algorithm)

The E-M algorithm is a type of iterative algorithm used for finding the maximum likelihood estimate of parameters in statistical models with latent variables. The algorithm consist of two steps are repeated iteratively until the model converges to the optimal soln.

> Initial Hypothesis w.r.t E-M algorithm :

The E-M algorithm starts with an initial guess for the model parameters. This guess is called the initial hypothesis. The initial hypothesis can be chosen randomly or based on prior knowledge about the model.

> Expectation step w.r.t. E-M algorithm :

In the expectation step, the algorithm computes the expected value of the latent variable given the observed data and the current estimate of the model parameters. This step is called the expectation step because it computes the expected value of the latent variables.

> Maximization step w.r.t. E-M algorithm :

In the maximization step, the algorithm updates the estimate of the model

PTO

parameters based on the expected value of the latent variables computed in the expectation step. This step is called maximization step because it maximizes the likelihood of the observed data given the current estimate of the model parameters.

Explain with example how initial hypothesis converges to optimal soln?

> The E-M algorithm repeats the expectation and maximization steps iteratively until convergence, where the parameters of the model do not change significantly between iterations.

> for example, consider the problem of clustering a set of data points into two clusters. We can use the E-M algorithm to estimate the mean & variance of each cluster. The initial hypothesis for the mean and variance of each cluster is chosen randomly. In the expectation step, we compute the probability that each cluster given the current estimate of the mean and variance of each.

- In the maximization step, we update the estimate of the mean & variance of each cluster based on the expected probabilities computed in the expectation step.
- We repeat the expectation & maximization steps until the estimate of the mean and variance of each cluster converges to the optimal solution. Because the likelihood of the observed data is maximized at the optimal solution.

---

5. Explain perceptron learning rule, Delta learning rule.

> Both the perceptron learning rule and the Delta learning rule are algorithm used to adjust the weights of a neural n/w during training, with the goal of minimizing the error b/w the predicted o/p & actual o/p.

> The perceptron learning rule is a simple algorithm that is used to train a single-layer neural n/w, also known as a perceptron. It works by adjusting the weights of the inputs to the perceptron based on the difference b/w the predicted output and the actual o/p. If the predicted o/p is greater than the

PTO

actual o/p, the weights of the inputs that contributed to the predicted o/p are decreased, and if the predicted o/p is less than the actual o/p, the weights are increased. The amount by which the weights are adjusted is determined by a learning rate parameter, which controls the rate of convergence of the algorithm.

The Delta learning Rule, also known as the LMS - Widrow Hoff learning rule, is a more general algorithm that can be used to train neural NWs with multiple layers. It worked by calculating the difference b/w the predicted o/p and the actual o/p, and then adjusting the weights of the inputs to each neuron in the NWs based on the error. The amount by which the weights are adjusted is proportional to the product of the error and the input to the neuron, and is also controlled by a learning rate parameter. The Delta learning rule is commonly used in training neural NWs for regression problems, where the goal is to predict a continuous o/p variables.

---

4. Explain McCulloch-pitts model.

> The M-P (Mc-Culloch-pitts) Neron is the neural NWs.

> It is usually called a M-P neuron.

- (1) The M-P neurons are connected by directed weighted paths.
- (2) Activation of a M-P neuron is binary, that is, at any time step. the neuron may or may not be fire...
- (3) The weights associated with the links are of two types!
  - a) they may be excitatory (weight +ve) or
  - b) they may be inhibitory (weight -ve)
- (4) Threshold plays major role
- (5) Any non-zero inhibitory i/p will prevent the neuron from firing
- (6) fig:- as shown aside...

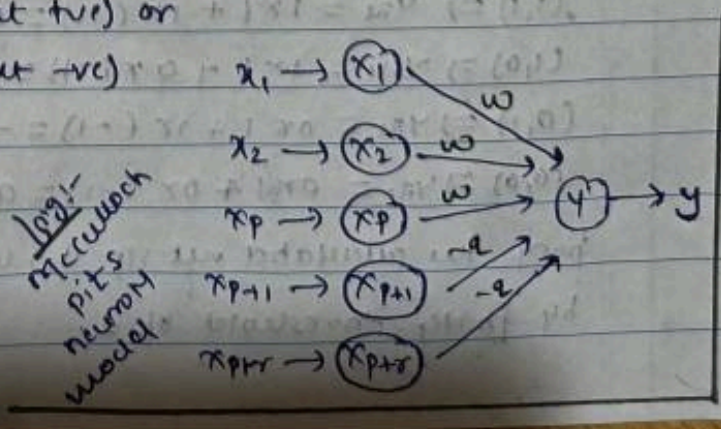


fig:- McCulloch pits neuron model

PTO

⇒ Determine weights and threshold for the given data using Mc Culloch-pitts neuron model. plot all data points and show separating hyperplane.

X1	X2	D
0	0	0
0	1	0
1	0	1
1	1	0

Soln:- Implementing AND NOT function

Response true → if first i/p - true
second i/p - false
& for all other values → response = false

Case 1: Assume that both weights w_1 and w_2 are excitatory.
i.e. $w_1 = 1$ and $w_2 = 1$
then for four inputs we calculate the net input using

$$y_{in} = x_1 w_1 + x_2 w_2$$

Now, we calculate net i/p as follows:

$(1,1) \Rightarrow y_{in} = 1\times1 + 1\times1 = 2$
$(1,0) \Rightarrow y_{in} = 1\times1 + 0\times1 = 1$
$(0,1) \Rightarrow y_{in} = 0\times1 + 1\times1 = 1$
$(0,0) \Rightarrow y_{in} = 0\times1 + 0\times1 = 0$

From the calculated net i/p it is not possible to fire the neuron for i/p (1,0) only.. hence these weights are not suitable...

Case 2: Assume one weight as excitatory & other as inhibitory
i.e. $w_1 = 1$ and $w_2 = -1$, Now, we calculate net input as,

$(1,1) \Rightarrow y_{in} = 1\times1 + 1\times(-1) = 0$
$(1,0) \Rightarrow y_{in} = 1\times1 + 0\times(-1) = 1$
$(0,1) \Rightarrow y_{in} = 0\times1 + 1\times(-1) = -1$
$(0,0) \Rightarrow y_{in} = 0\times1 + 0\times(-1) = 0$

From the calculated net i/p it is now possible to fire the neuron for input (1,0) by fixing threshold of 1.

PTO

i.e. $T \geq 1$. These $w_1 = 1$ and $w_2 = -1$
the o/p of the neuron can be written as,

$$y = f(y_{in}) = \begin{cases} 1 & \text{if } y_{in} \geq 1 \\ 0 & \text{if } y_{in} < 1 \end{cases}$$

---

6. Explain Multi-layer perceptron NN with example...

⇒ A multi-layer perceptron (MLP) is a type of neural NN that consists of multiple layers of neurons, including an input layer, one or more hidden layers and an o/p layer.
Each neuron in the NN is connected to every neuron in the adjacent layers by a weight, which represents the strength of the connection b/w the neurons. The MLP uses a process called forward propogation to calculate the o/p of the NN for a given i/p and a process called backpropogation to adjust the weights of the connection b/w the neurons during training.

To understand how an MLP works, lets consider an example of a neural NN that is trained to classify images of handwritten digits into the correct numerical value (0-9). The input to the NN is a grayscale image of size 28×28 pixels, which is flattened into a vector of 784 i/p nodes. The o/p of the NN is a vector of 10 o/p nodes, representing the probability that the i/p image corresponds to each of the possible numerical value...

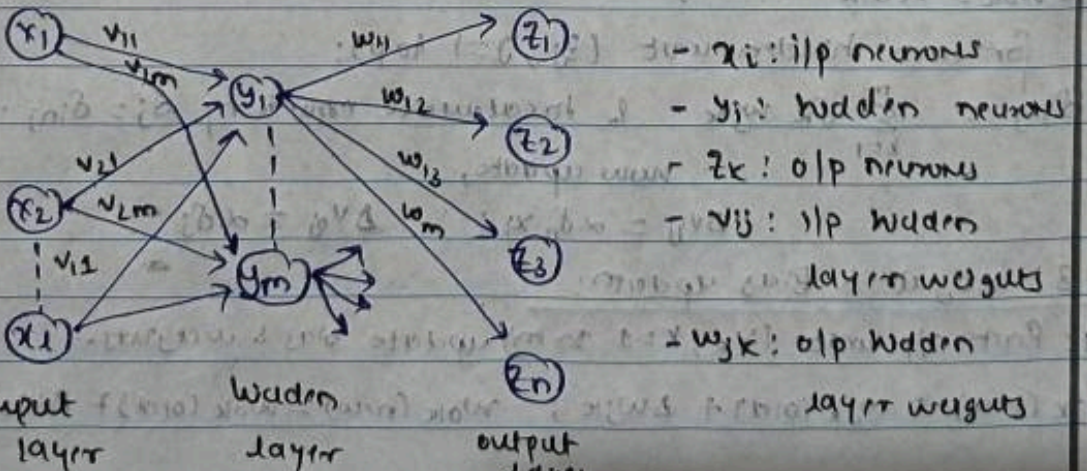


Fig:- A multilayer feed forward NN

7. Explain error back propogation algorithm in detail:

- Back-propogation n/w algorithm is applied to multilayer feed-forward NNW consisting of processing elements..
- The NNW connected to back-propogation learning algorithm are also called as back-propogation NNW (BPN)
- This algorithm provides a procedure for changing the weight in back-propogation NNW (BPN) to the given i/p pattern correctly..

Algorithm:

> Step 1: Small random values for weights and learning rate..
> Step 2: Carry on the steps 3-10 when stopping condition fails.
> Step 3: Carry on steps 3-9 for each training pair..

Phase 1: feed-forward phase..

> Step 4: Each i/p receives i/p signal x_i and sends to hidden unit for $i = 1$ to n.
> Step 5: calculate net i/p.

$$z_{inj} = v_{0j} + \sum_{i=1}^{n} x_i v_{ij}$$

> Step 6: for each o/p y_k ($k = 1$ to m)

$$y_{ink} = w_{0k} + \sum_{j=1}^{p} z_j w_{jk}$$

Phase 2: (Back-propogation of error)

> Step 7: compute error y_k ($k = 1$ to m)

$\delta_k = (t_k - y_k) f'(y_{ink})$

$\Delta w_{jk} = \alpha \delta_k z_j$; $\Delta w_{0k} = \alpha \delta_k$ } sent back hidden layer backwards...

> Step 8: for each hidden unit (z_j, $j = 1$ to p);

$\delta_{inj} = \sum_{k=1}^{m} \delta_k w_{jk}$ & to calculate error term $\delta_j = \delta_{inj} \cdot f'(z_{inj})$

now update,

$\Delta v_{ij} = \alpha \delta_j x_i$; $\Delta v_{0j} = \alpha \delta_j$

Phase 3: weight & Bias update:

> Step 9: Each o/p unit (y_k; $k = 1$ to m) update bias & weights..

$w_{jk}(new) = w_{jk}(old) + \Delta w_{jk}$; $w_{0k}(new) = w_{0k}(old) + \Delta w_{0k}$

PTO

Each hidden unit (Z_j, $j = 1$ to p) update its bias & weights:

$V_{ij}(new) = V_{ij}(old) + \Delta V_{ij}$ & $V_{0j}(new) = \Delta V_{0j} + V_{0j}(old)$

> Step 10: check for stopping condition:

---

8. Explain classification with logistic regression & sigmoid function?

Logistic regression: Type of linear classification algorithm used to model the probability of a binary class variable, given one or more independent variables. In logistic regression, we use a sigmoid function as the activation function to transform the o/p of a linear model into probability score...

Sigmoid function: mathematical function that produces an S-shaped curve. It takes any real-valued no. & maps it to the value b/w 0 and 1. the sigmoid function is defined as follows:

$$\text{Sigmoid}(x) = 1 / (1 + e^{-x})$$ ... mathematical constant $e = 2.71828$

⇒ In the context of logistic regression, we use the sigmoid function to transform the o/p of a linear model ('z') into probability score 'p', which represents the likelihood of a Binary outcome (eg:- 1 or 0):

$$z = b_0 + b_1 * x_1 + b_2 * x_2 + \dots b_n * x_n$$
$$p = \text{sigmoid}(z) \quad \# \text{sigmoid}(z)$$

Here, 'x1', 'x2', 'x3' ... 'xn' are independent variables...

'b0', 'b1' ... 'bn', are coefficient or weights of linear model.

b0 → bias or intercept... 'p' → probability... +ve (1)

- To make the prediction, we use a threshold value (usually 0.5) to classify the outcome either 0 or 1. If the probability 'p' is greater than or equal to threshold we predict +ve outcome (1), otherwise, we predict (-ve) outcome (0)...
- In summary, logistic regression is a type of linear classification algorithm used to model the probability of a Binary class variable, and sigmoid function used as activation function to transform o/p of a linear model into probability score b/w 0 and 1.

9. What is curse of dimensionality and its curse?

Refers to the difficulties that arise when working with high-dimensional data. As the no of features or dimensions increase, the amount of data required to maintain the same level of performance also increases exponentially.

The core of curse of dimensionality depends on specific problem & context.

① feature selection: Select subset of most relevant data.

② feature extraction: transform high-dimensional data into low-dimensional

③ regularization: used to penalize complex models.

④ sampling: used to reduce amount of data used.

⑤ model selection: choosing model that is less sensitive to curse.

Explain dimensionality reduction with its different techniques!

process of reducing the no. of features & dimensions in a dataset while preserving as much relevant information as possible.

Two main approaches to dimensionality reduction:

> 1) Feature selection: select subset of original features based on some criteria.

methods: - filter method: rank the feature.

- wrapper method: to evaluate importance of each feature.

- Embedded method: combine selection & model training into single step.

> 2) Feature extraction: transforming original features into new set of features.

methods: - principle component Analysis: find set of orthogonal vectors.

- linear discriminant analysis: maximize the separability.

10. Describe how principle component Analysis is carried out to reduce dimensionality of datasets.

PCA is a popular technique used for dimensionality reduction. PCA involves transforming a high-dimensional dataset into lower-dimensional space.

Steps involved in carrying out PCA:

> ① Standardized the data: To ensure that each feature has an equal

PTO

weights for the analysis...

> ② Calculate covariance matrix: measure relationship b/w the features. calculated by multiplying transpose of data matrix by data matrix itself.

> ③ Compute the eigen values & eigen vectors: can be calculated using matrix factorization techniques such as singular value decomposition (SVD).

> ④ Select principle components: eigen vector that explains most variance in the data. they are selected based on eigen values, with first principle component...

> ⑤ Transforming the data: transform data (original) to the lower dimensional space using the selected principle components. This is done by the multiplying data matrix by matrix of selected eigen vectors...

---

11. Implement AND function using perceptron NN for bipolar i/p & targets.

x_1	x_2	t
1	1	1
1	-1	-1
-1	1	-1
-1	-1	-1

Soln:- $y_{in} = b + x_1 w_1 + x_2 w_2$

$\alpha = 1$ ... learning rate...

$$y = f(y_{in}) = \begin{cases} 1 & \text{if } y_{in} > 0 \\ 0 & \text{if } y_{in} = 0 \\ -1 & \text{if } y_{in} < 0 \end{cases}$$

also,

$\Delta w_1 = \alpha t x_1$

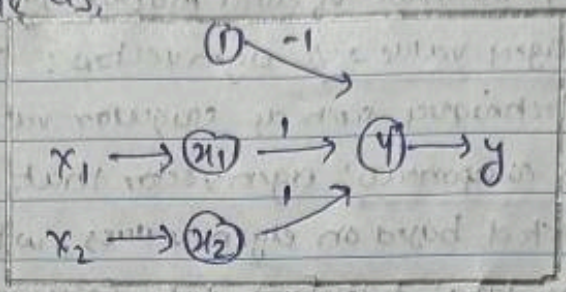
$\Delta w_2 = \alpha t x_2$

$\Delta b = \alpha t$

	input		Target	N/t i/p	cal.	weight changes			weights		
	x_1	x_2	t	(y_{in})	o/p (y)	Δw_1	Δw_2	Δb	w_1 0	w_2 0	w_0 0
EPOCH 1	1	1	1	0	0	1	1	1	1	1	1
	1	-1	-1	1	1	-1	1	-1	0	2	0
	-1	1	-1	2	1	1	-1	-1	1	1	-1
	-1	-1	-1	-3	-1	0	0	0	1	1	-1
EPOCH 2	1	1	1	1	1	0	0	0	1	1	-1
	1	-1	-1	-1	-1	0	0	0	1	1	-1
	-1	1	-1	-1	-1	0	0	0	1	1	-1
	-1	-1	-1	-3	-1	0	0	0	1	1	-1

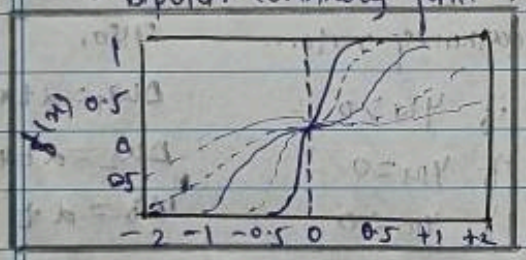
PTO

All the net i/p & actual o/p classified...
i.e. 1, 1, -1 as weights for $\alpha = 1$
Illustrated in figure as,



12. - Explain different activation functions:
The neuron as a processing node performs the operation of summation of its weighted i/p or the scalar product computation to obtain the net.

① Bipolar Activation function:
- Bipolar binary function: $f(net) \triangleq sgn(net) = \begin{cases} +1, & net > 0 \\ -1, & net < 0 \end{cases}$
- Bipolar continuous function: $f(net) \triangleq \frac{2}{1 + exp(-\lambda net)} - 1$



.... Bipolar sigmoid 1 ... Bipolar sigmoid 2
—— Bipolar sigmoid 4 - - - Bipolar sigmoid 8
λ —— Bipolar sigmoid ...

② Unipolar Activation function: By shifting & scaling the Bipolar activation...
- Unipolar continuous: $f(net) \triangleq 1 / 1 + exp(-\lambda net)$
- Unipolar Binary: $f(net) \triangleq \begin{cases} 1, & net > 0 \\ 0, & net < 0 \end{cases}$ limit $f(net)$ $\lambda \to \infty$

③ Identity function: Linear function:
$f(x) = x$ for all x

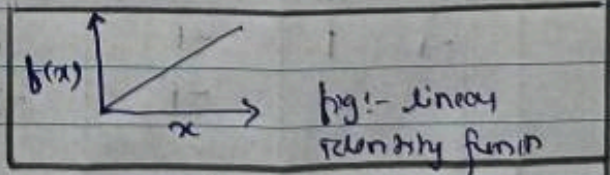


fig:- Linear identity function

④ Ramp function:

$$f(x) = \begin{cases} 1, & \text{if } x > 1 \\ x, & \text{if } 0 \le x \le 1 \\ 0, & \text{if } x < 0 \end{cases}$$

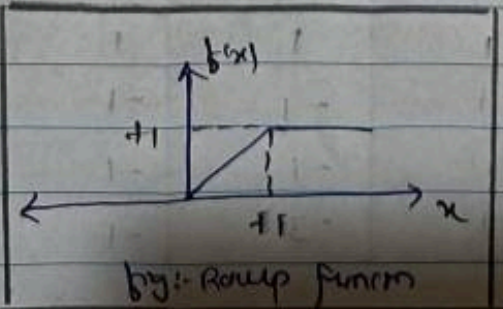


fig:- Ramp function