SRI RAMAKRISHNA ENGINEERING COLLEGE

[Educational Service: SNR Sons Charitable Trust]
[Autonomous Institution, Reaccredited by NAAC with 'A+' Grade]
[Approved by AICTE and Permanently Affiliated to Anna University, Chennai]
[ISO 9001:2015 Certified and all eligible programmes Accredited by NBA]
VATTAMALAIPALAYAM, N.G.G.O. COLONY POST, COIMBATORE- 641 022.

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

Class: II B.Tech AI&DS **Semester: IV**

Certified that this is the bonafide record of work done by _____ in the **20AD273 - Machine Learning Laboratory** of this institution for IV semester during the academic year 2023 - 2024.

Faculty In-Charge **Head of the Department**

Date:

Register No.

Submitted for the IV Semester B.Tech Autonomous Practical Examination held on _____.

Internal Examiner **Subject Expert**

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE - VISION AND MISSION

VISION

- To achieve academic excellence in the domain of Artificial Intelligence and Data Science and produce globally competent professionals to solve futuristic societal challenges

MISSION

- To actively engage in the implementation of innovative intelligent solutions for interdisciplinary Artificial Intelligence based solutions with ethical standards
- To promote research, innovation and entrepreneurial skills through industry and academic collaboration

PROGRAM EDUCATIONAL OBJECTIVES (PEOS)

The graduates of this program after four to five years will,

PEO 1: Design and develop solutions for real-world problems based on business and societal needs, as skilled professionals or entrepreneurs.

PEO 2: Apply Artificial Intelligence and Data Science knowledge and skills to develop innovative solutions for multi-disciplinary problems, adhering to ethical standards

PEO 3: Engage in constructive research, professional development and life-long learning to adapt with emerging technologies

PROGRAM OUTCOMES (POs)

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

2. **Problem analysis:** Identity, formulates, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for public health and safety, and the cultural, societal, and environmental considerations.

4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.

6. **The engineer and society:** Apply to reason informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9. **Individual and teamwork:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

12. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life–long learning in the broadest context of technological change.

PROGRAM SPECIFIC OUTCOMES (PSOs)

Graduates of Artificial Intelligence and Data Science at the time of graduation will be able to

PSO 1: Analyze, design and build sustainable intelligent solutions to solve challenges imposed by industry and society.

PSO2: Demonstrate data analysis skills to achieve effective insights and decision making to solve real-life problems.

PSO3: Apply mathematical and statistical models to solve the computational tasks, and model real-world problems using appropriate AI / ML algorithms.

COURSE OUTCOMES (COs)

CO1: Apply the concepts of Machine Learning to solve real-world problems.

CO2: Implement basic algorithms in clustering & classification applied to text & numeric data.

CO3: Develop fundamental and advanced neural network algorithms for solving real-world data.

CO4: Communicate the programming results through reports.

TABLE OF CONTENTS

Ex. No: 1a	INTRODUCTION TO PYTHON PROGRAMMING -NUMPY, PANDAS

AIM:

Write a program to import NumPy and Pandas libraries.

I. NumPy

- NumPy stands for Numerical Python.
- It is one of the most widely used library.
- As it contains the code related to numerical details it is most popular around data science and machine learning as both these fields need a lot of numerical logic getting applied in it.
- It is used whenever the situation in coding arises in working with an array.
- It does have methods that is made up for algebra related logics.
- This NumPy was made in the year 2005.

1. Insert array.

Program:

```
import numpy as np
arr=np.array(([17,24,3,7]))
print(arr)
print(type(arr))
```

Output:

```
[17 24  3  7]
<class 'numpy.ndarray'>
```

2. Search the maximum and minimum element in the given array using NumPy

Program:

```
import numpy as np
arr=np.arange(15).reshape(5,3)
print(arr)
print()
print("Maximum of the array:")
print(np.argmax(arr))
print("Minimum of the array:")
```

```
print(np.argmin(arr))
print("Maximum value of the each columns")
print(np.argmax(arr,axis=0))
print("Maximum value of the each row")
print(np.argmax(arr,axis=1))
```

Output:

```
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]
 [12 13 14]]

Maximum of the array:
14
Minimum of the array:
0
Maximum value of the each columns
[4 4 4]
Maximum value of the each row
[2 2 2 2 2]
```

3. **Find the mean of every NumPy array in the given list**

Program:

```
import numpy as np
list=[np.array([17,24,3,25,7,5]),np.array([2,4,6,8,10]),np.array([1,3,5,7,9])]
result=[]
for i in range(len(list)):
        result.append(np.mean(list[i]))
print("Solution:")
print(result)
```

Output:

```
Solution:
[13.5, 6.0, 5.0]
```

II. Pandas

- The main role of the Pandas library is to analyse the data.
- It is open source in nature.
- It is used in relational data.
- On the top of NumPy library Pandas library is present.
- By using pandas, you can reshape, analyze, and change your data very easily.

Pandas supports two data structures:

1. **Series:**

   - It is an array.
   - It can hold any kind of data types like integer, float, character etc.
   - It points to the column.

2. **Data Frame:**

   - It handles 3 parts, mainly data, columns and rows.

1. **Create a simple Pandas Series from a list**

Program:

```
import pandas as pd
a=[17,24,3,25,7,"Moule"]
var=pd.Series(a)
print("Solution:")
print(var)
```

Output:

```
Solution:
0       17
1       24
2        3
3       25
4        7
5    Moule
dtype: object
```

2. **Key/Value Objects as Series**

Program:

```
import pandas as pd
var1={"Name": "Moulesh","Rollno":2111032,"Branch":"AI & DS"}
var=pd.Series(var1)
print("Solution:")
print(var)
```

Output:

```
Solution:
Name      Moulesh
Rollno    2111032
Branch    AI & DS
dtype: object
```

3. Data Frames

Program:

```
import pandas as pd
data={"Semester":["sem1","sem2","sem3","sem4","sem5","sem6"],"CGPA":[7,7.2,7.4,7.6,7.8
,8]}
var=pd.DataFrame(data)
print("Data Frame:")
print(var)
```

Output:

```
Data Frame:
  Semester  CGPA
0     sem1   7.0
1     sem2   7.2
2     sem3   7.4
3     sem4   7.6
4     sem5   7.8
5     sem6   8.0
Semester    sem1
CGPA         7.0
Name: 0, dtype: object
```

4. Locate Row

Program:

```
import pandas as pd
data={"Semester":["sem1","sem2","sem3","sem4","sem5","sem6"],"CGPA":[7,7.2,7.4,7.6,
7.8,8]}
var=pd.DataFrame(data)
print("Data Frame:")
print(var)
print(var.loc[0])
print(var.loc[[0,1]])
```

Output:

```
Data Frame:
  Semester  CGPA
0     sem1   7.0
1     sem2   7.2
2     sem3   7.4
3     sem4   7.6
4     sem5   7.8
5     sem6   8.0
Semester    sem1
CGPA         7.0
Name: 0, dtype: object
  Semester  CGPA
0     sem1   7.0
1     sem2   7.2
```

5. **Pandas Read CSV**

Program:

```
from google.colab import files
uploaded = files.upload()
import pandas as pd
import io
df2 = pd.read_csv(io.BytesIO(uploaded['Student_Marks.csv']))
print(df2.to_string())
```

Output:

```
   number_courses  time_study   Marks
0               3       4.508  19.202
1               4       0.096   7.734
2               4       3.133  13.811
3               6       7.909  53.018
4               8       7.811  55.299
5               6       3.211  17.822
```

RESULT:

Thus, the python program to import NumPy and Pandas has been executed successfully.

Ex. No: 1b	BASIC PROGRAMS IN SCIPY, MATPLOTLIB, SCIKIT LEARN

AIM:

To perform a program to import SciPy, Matplotlib and Scikit learn.

III SciPy

- It uses scientific and mathematical logic.
- It stands for "Scientific Python".
- Manipulating N-dimension array is done through SciPy.
- Some sub packages of SciPy are as follows:
  - SciPy, cluster: K mean algorithm and such similar algorithms can be done using this library.
  - Scipy.io: Inputs and outputs are handled here.
- SciPy in Python is an open-source library used for solving mathematical, scientific, engineering, and technical problems. It allows users to manipulate the data and visualize the data using a wide range of high-level Python commands.
- The SciPy library supports integration, gradient optimization, special functions, ordinary differential equation solvers, parallel programming tools, and many more.

1.Solving a set of linear equations:

Program:

```
import numpy as np
from scipy import linalg
a = np.array([[2, 1], [4,-5]])
b = np.array([[5], [7]])
x = linalg.solve(a, b)
print(x)
print("\n Checking the results, the values must be zeros")
print(a.dot(x) - b)
```

Output:

```
[[2.28571429]
 [0.42857143]]

 Checking the results, the values must be zeros
[[0.]
 [0.]]
```

1. **Finding determinants of matrices**

Program:

```
from scipy import linalg
import numpy as np
mat = np.array([[1,2,3],[4,5,6],[7,8,5]])
x = linalg.det(mat)
print(f'Determinant of the matrix\n{mat} \n is {x}')
```

Output:

```
Determinant of the matrix
[[1 2 3]
 [4 5 6]
 [7 8 5]]
 is 12.0
```

2. **Finding the inverse of a matrix**

Program:

```
from scipy import linalg
import numpy as np
mat = np.array([[1,2,3],[4,5,6],[7,8,5]])
x = linalg.det(mat)
print("Inverse of the matrix")
mat2 = linalg.inv(mat)
print(mat2)
```

Output:

```
Inverse of the matrix
[[-1.91666667  1.16666667 -0.25      ]
 [ 1.83333333 -1.33333333  0.5       ]
 [-0.25        0.5        -0.25      ]]
```

IV. MATPLOTLIB

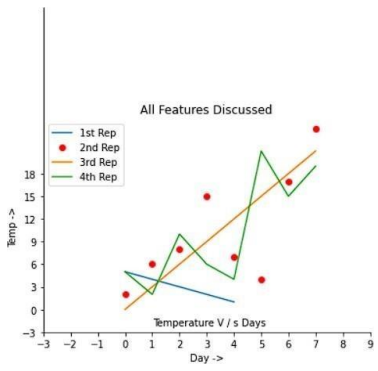
- It is used to plot graphs.
- It is open source.
- In python you need to install matplotlib pip otherwise the code will not execute.

1. **Basic functions that are often used in matplotlib.**

Program:

```
import matplotlib.pyplot as plt
a = [5,4,3,2,1]
b = [2, 6, 8, 15, 7, 4, 17, 24]
plt.plot(a)
plt.plot(b, "or")
plt.plot(list(range(0, 22, 3)))
plt.xlabel('Day ->')
plt.ylabel('Temp ->')
c = [5, 2, 10, 6, 4, 21, 15, 19]
plt.plot(c, label = '4th Rep')
ax = plt.gca()
ax.spines['right'].set_visible(False)
ax.spines['top'].set_visible(False)
ax.spines['left'].set_bounds(-3, 40)
plt.xticks(list(range(-3, 10)))
plt.yticks(list(range(-3, 20, 3)))
ax.legend(['1st Rep', '2nd Rep', '3rd Rep', '4th Rep'])
plt.annotate('Temperature V / s Days', xy = (1.01, -2.15))
plt.title('All Features Discussed')
plt.show()
```

Output:

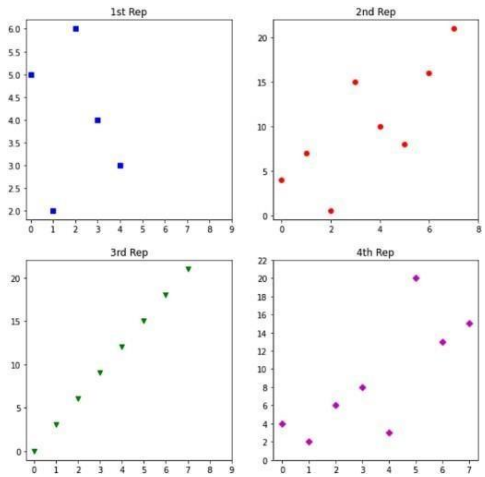


2. Subplot

Program:

```
a = [5, 2, 6, 4, 3]
b = [4, 7, 0.5, 15, 10, 8, 16, 21]
c = [4, 2, 6, 8, 3, 20, 13, 15]
fig = plt.figure(figsize =(10, 10))
sub1 = plt.subplot(2, 2, 1)
sub2 = plt.subplot(2, 2, 2)
sub3 = plt.subplot(2, 2, 3)
sub4 = plt.subplot(2, 2, 4)
sub1.plot(a, 'sb')
sub1.set_xticks(list(range(0, 10, 1)))
sub1.set_title('1st Rep')
sub2.plot(b, 'or')
sub2.set_xticks(list(range(0, 10, 2)))
sub2.set_title('2nd Rep')
sub3.plot(list(range(0, 22, 3)), 'vg')
sub3.set_xticks(list(range(0, 10, 1)))
sub3.set_title('3rd  Rep')
sub4.plot(c, 'Dm')
sub4.set_yticks(list(range(0, 24, 2)))
sub4.set_title('4th Rep')
plt.show()
```

Output:



V. Scikit Learn.

- Simple and efficient tools for data mining and data analysis. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means, etc.
- Accessible to everybody and reusable in various contexts.
- Built on the top of NumPy, SciPy, and matplotlib.
- Open source, commercially usable – BSD license.

1. Load a dataset

Program:

```
from sklearn.datasets import load_iris
iris = load_iris()
X = iris.data
y = iris.target
feature_names = iris.feature_names
target_names = iris.target_names
print("Feature names:", feature_names)
print("Target names:", target_names)
print("\nType of X is:", type(X))
print("\nFirst 5 rows of X:\n", X[:5])
```

Output:

```
Feature names: ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
Target names: ['setosa' 'versicolor' 'virginica']

Type of X is: <class 'numpy.ndarray'>

First 5 rows of X:
 [[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]]
```

2. Splitting the dataset

Program:

```
from sklearn.datasets import load_iris
iris = load_iris()
X = iris.data
```

```
y = iris.target
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=1)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

Output:

```
(90, 4)
(60, 4)
(90,)
(60,)
```

RESULT:

Thus, the program to import SciPy, Matplotlib and Scikit Learn has been executed successfully.

Ex. No: 2	DATA AND TEXT CLUSTERING USING K- MEANS ALGORITHM

AIM:

To implement k-Means clustering for a dataset using pandas and sklearn.

ALGORITHM:

K-Means Clustering is an Unsupervised Learning algorithm, which groups the unlabeled dataset into different clusters. Here K defines the number of pre-defined clusters that need to be created in the process, as if K=2, there will be two clusters, and for K=3, there will be three clusters, and so on.

The working of the K-Means algorithm is explained in the below steps:

Step-1: Select the number K to decide the number of clusters.

Step-2: Select random K points or centroids. (It can be other from the input dataset).

Step-3: Assign each data point to their closest centroid, which will form the predefined K clusters.

Step-4: Calculate the variance and place a new centroid of each cluster.

Step-5: Repeat the third steps, which means reassign each datapoint to the new closest centroid of each cluster.

Step-6: If any reassignment occurs, then go to step-4 else go to FINISH.

Step-7: The model is ready.

Data clustering using K-means:

PROGRAM:

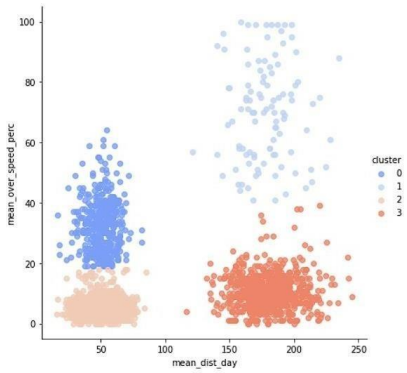
```
import pandas as pd
data = pd.read_csv("driver.csv", index_col="id")
data.head()
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=4)
kmeans.fit(data)
kmeans.cluster_centers_
```

```
kmeans.labels_
import numpy as np
unique, counts = np.unique(kmeans.labels_, return_counts=True)
dict_data = dict(zip(unique, counts))
dict_data
import seaborn as sns
data["cluster"] = kmeans.labels_
sns.lmplot('mean_dist_day', 'mean_over_speed_perc', data=data, hue='cluster', palette='coolwarm', size=6, aspect=1, fit_reg=False)
kmeans.inertia_
kmeans.score
data
```

Output:

id	mean_dist_day	mean_over_speed_perc	cluster
3423311935	71.24	28	0
3423313212	52.53	25	0
3423313724	64.54	27	0
3423311373	55.69	22	0
3423310999	54.58	25	0
...	...	...	...
3423310685	160.04	10	3
3423312600	176.17	5	3
3423312921	170.91	12	3
3423313630	176.14	5	3
3423311533	168.03	9	3

4000 rows × 3 columns



Text clustering using K-means:

PROGRAM:

```
df= pd.read_csv("movies_metadata.csv")
df.head()
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import KMeans
documents = df['overview'].values.astype("U")
 documents
vectorizer = TfidfVectorizer(stop_words='english')
features = vectorizer.fit_transform(documents)
k = 30
model = KMeans(n_clusters=k, init='k-means++', max_iter=100, n_init=1)
model.fit(features)
df['cluster'] = model.labels_
```

```
df.head()
clusters = df.groupby('cluster')
for cluster in clusters.groups:
    f = open('cluster'+str(cluster)+ '.csv', 'w')
    data = clusters.get_group(cluster)[['title','overview']]
    f.write(data.to_csv(index_label='id'))
f.close()
print("Cluster centroids: \n")
order_centroids = model.cluster_centers_.argsort()[:, ::-1]
terms = vectorizer.get_feature_names()
for i in range(k):
    print("Cluster %d:" % i)
    for j in order_centroids[i, :10]:
        print (' %s' % terms[j])
    print('_________')
```

Output:

	id	title	overview
0	0	Toy Story	Led by Woody, Andy's toys live happily in his ...
1	1	Jumanji	When siblings Judy and Peter discover an encha...
2	2	Grumpier Old Men	A family wedding reignites the ancient feud be...
3	3	Waiting to Exhale	Cheated on, mistreated and stepped on, the wom...
4	4	Father of the Bride Part II	Just when George Banks has recovered from his ...

```
Cluster centroids:

Cluster 0:
 town
 young
 wife
 time
 new
 finds
 people
 years
 help
 men
------------
Cluster 1:
 story
 true
 tells
 based
 love
 life
 film
 man
 set
 young
------------
Cluster 2:
 mother
 daughter
 father
 son
 family
 young
 old
 home
 life
 year
------------
Cluster 3:
 woman
 young
 husband
 man
 life
 love
 finds
 married
 tries
 beautiful
------------
Cluster 4:
 man
 young
 wife
 life
 old
 finds
 love
 murder
 town
 daughter
------------
Cluster 5:
 life
 new
 love
 day
 time
 old
 young
 years
 wife
 finds
------------
```

RESULT:

Thus, the program to implement K-Means clustering for a dataset using Pandas and SKlearn has been executed successfully.

Ex. No: 3a	REGRESSION USING DECISION TREE

AIM:

To find the decision tree classifier using Sklearn for given data set.

ALGORITHM:

Decision Tree is a supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.

Step-1: Begin the tree with the root node, says S, which contains the complete dataset.

Step-2: Find the best attribute in the dataset using Attribute Selection Measure(ASM).

Step-3: Divide the S into subsets that contains possible values for the best attributes.

Step-4: Generate the decision tree node, which contains the best attribute.

Step-5: Recursively make new decision trees using the subsets of the dataset created in step -**3.** Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node.

PROGRAM:

```
import pandas as pd
import numpy as np
from google.colab import files
uploaded = files.upload()
train_data_m = pd.read_csv("PlayTennis.csv")
#importing the dataset from the disk
train_data_m.head()
```

Calculating the entropy of the whole dataset:

```
def calc_total_entropy(train_data, label, class_list):
    total_row = train_data.shape[0]
    total_entr = 0
    for c in class_list: #for each class in the label
        total_class_count = train_data[train_data[label] == c].shape[0]
        total_class_entr = -(total_class_count/total_row)*np.log2(total_class_count/total_row)
        total_entr += total_class_entr
    return total_entr
```

Calculating the entropy for the filtered dataset:

```
def calc_entropy(feature_value_data, label, class_list):
    class_count = feature_value_data.shape[0]
    entropy = 0
    for c in class_list:
        label_class_count = feature_value_data[feature_value_data[label] == c].shape[0]
        entropy_class = 0
        if label_class_count != 0:
            probability_class = label_class_count/class_count
            entropy_class = - probability_class * np.log2(probability_class)
        entropy += entropy_class
    return entropy
```

Calculating information gain for a feature:

```
def calc_info_gain(feature_name, train_data, label, class_list):
    feature_value_list = train_data[feature_name].unique()
    total_row = train_data.shape[0]
    feature_info = 0.0
   for feature_value in feature_value_list:
```

```
            feature_value_data = train_data[train_data[feature_name] == feature_value]
            feature_value_count = feature_value_data.shape[0]
            feature_value_entropy = calc_entropy(feature_value_data, label,class_list)
            feature_value_probability = feature_value_count/total_row
            feature_info += feature_value_probability * feature_value_entropy
        return calc_total_entropy(train_data, label, class_list) -feature_info
```

Finding the most informative feature (feature with highest information gain):

```
    def find_most_informative_feature(train_data, label, class_list):
        feature_list = train_data.columns.drop(label)
        max_info_gain = -1
        max_info_feature = None
        for feature in feature_list:
            feature_info_gain = calc_info_gain(feature, train_data, label, class_list)
            if max_info_gain &lt; feature_info_gain:
                max_info_gain = feature_info_gain
                max_info_feature = feature
        return max_info_feature
```

Adding a node to the tree:

```
    def generate_sub_tree(feature_name, train_data, label, class_list):
        feature_value_count_dict = train_data[feature_name].value_counts(sort=False)
        tree = {}
        for feature_value, count in feature_value_count_dict.iteritems():
            feature_value_data = train_data[train_data[feature_name] == feature_value]
            assigned_to_node = False
            for c in class_list:
                class_count = feature_value_data[feature_value_data[label] == c].shape[0]
```

```
            if class_count == count:
                tree[feature_value] = c
                train_data = train_data[train_data[feature_name] != feature_value]
                assigned_to_node = True
            if not assigned_to_node:
                        tree[feature_value] =”?”
        return tree, train_data
```

Performing ID3 Algorithm and generating Tree

```
    def make_tree(root, prev_feature_value, train_data, label, class_list):
        if train_data.shape[0] != 0:
            max_info_feature = find_most_informative_feature(train_data, label, class_list)
            tree, train_data = generate_sub_tree(max_info_feature, train_data, label, class_list)
            next_root = None
            if prev_feature_value != None:
                root[prev_feature_value] = dict()
                root[prev_feature_value][max_info_feature] = tree
                next_root = root[prev_feature_value][max_info_feature]
            else:
                root[max_info_feature] = tree
                next_root = root[max_info_feature]
            for node, branch in list(next_root.items()):
                if branch == “?”::
                    feature_value_data = train_data[train_data[max_info_feature] == node]
                    make_tree(next_root, node, feature_value_data, label, class_list)
```

Finding unique classes of the label and Starting the algorithm

```
def id3(train_data_m, label):
    train_data = train_data_m.copy()

    tree = {}

    class_list = train_data[label].unique()

    make_tree(tree, None, train_data, label, class_list)

    return tree
```

Print the tree

```
tree = id3(train_data_m, 'Play Tennis')

print(tree)
```

Output:

```
{
   'Outlook':
   {
       'Rain':
       {
           'Wind':
           {
               'Strong': 'No',
               'Weak': 'Yes'
           }
        },
       'Sunny':
       {
           'Humidity':
           {
               'High': 'No',
               'Normal': 'Yes'
           }
       },
       'Overcast': 'Yes'
    }
}
```

RESULT:

Thus, the program to find decision tree classifier using sklearn for given dataset has been executed successfully.

Ex. No: 3b	DECISION TREE CLASSIFIER

AIM:

To find the decision tree regression using Sklearn for given data set.

ALGORITHM:

Decision tree regression observes features of an object and trains a model in the structure of a tree to predict data in the future to produce meaningful continuous output. Continuous output means that the output/result is not discrete, i.e., it is not represented just by a discrete, known set of numbers or values. In a regression tree, a regression model is fit to the target variable using each of theindependent variables. The data is then split at several points for each independentvariable.

Step 1: Import the required libraries.

Step 2: Initialize and print the dataset.

Step 3: Fit regression model

Step 4: Predict and plot the results

Data Set Characteristics:

Number of Instances: 20640

Number of Attributes: 8 numeric, predictive attributes and the target

Attribute Information:

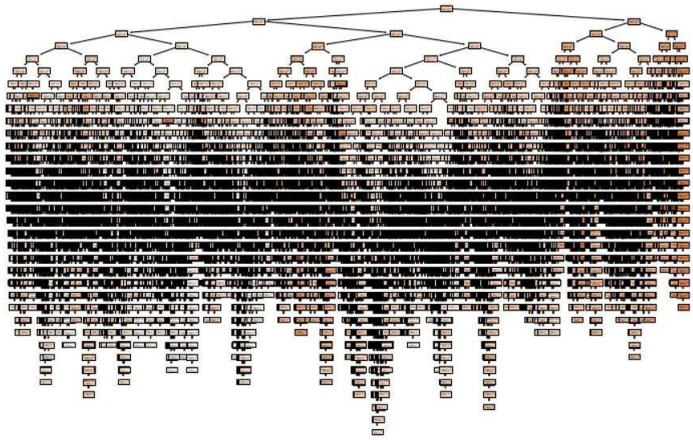
- **MedInc** - median income in block group
- **HouseAge** - median house age in block group
- **AveRooms** - average number of rooms per household
- **AveBedrms** - average number of bedrooms per household
- **Population** - block group population
- **AveOccup** - average number of household members
- **Latitude** - block group latitude
- **Longitude** - block group longitude

Missing Attribute Values: None

PROGRAM:

```
from sklearn.datasets import fetch_california_housing
from sklearn.tree import  DecisionTreeRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn import tree
housing =fetch_california_housing()
X_train, X_test, y_train, y_test = train_test_split(housing.data, housing.target, test_size=0.2,
random_state=42)
dt = DecisionTreeRegressor(random_state=42)
dt.fit(X_train,y_train)
y_pred = dt.predict(X_test)
import matplotlib.pyplot as plt
plt.figure(figsize=(12,8))
tree.plot_tree(dt, filled=True,feature_names=housing.feature_names)
plt.show()
```

Output:



RESULT:

Thus, the program to find the decision tree using Sklearn from the given dataset has been executed successfully.

Ex. No: 4	BAGGING IN CLASSIFICATION

AIM:

To write a python program to demonstrate bagging in classification for breast cancer prediction.

ALGORITHM:

Bagging classifier is an ensemble meta-estimator that fits base classifiers each on random subsets of the original dataset and then aggregate their individual predictions (either by voting or by averaging) to form a final prediction. Such a meta-estimator can typically be used as a way to reduce the variance of a black-box estimator (e.g., a decision tree), by introducing randomization into its construction procedure and then making an ensemble out of it.

Classifier generation:

Let N be the size of the training set.
for each of t iterations:
sample N instances with replacement from the original training set.
apply the learning algorithm to the sample.
store the resulting classifier.

Classification:

for each of the t classifiers:
predict class of instance using classifier.
return class that was predicted most often.

PROGRAM:

```
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import make_pipeline
from sklearn.ensemble import BaggingClassifier
```

```
from sklearn.model_selection import GridSearchCV
bc = datasets.load_breast_cancer()
X = bc.data
y = bc.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=1,
stratify=y)
pipeline = make_pipeline(StandardScaler(),LogisticRegression(random_state=1))
bgclassifier = BaggingClassifier(base_estimator=pipeline, n_estimators=100,
max_features=10,max_samples=100, random_state=1, n_jobs=5)
bgclassifier.fit(X_train, y_train)
print('Model test Score: %.3f, ' %bgclassifier.score(X_test, y_test), 'Model training Score:
%.3f' %bgclassifier.score(X_train, y_train))
```

Output:

```
Model test Score: 0.958,  Model training Score: 0.972
```

RESULT:

Thus, the program to demonstrate bagging in classification for breast cancer prediction has been performed and executed successfully.

Ex. No: 5a	DATA CLASSIFICATION WITH NEURAL NETWORKS

AIM:

To write a python program to implement Data Classification with Neural Network.

ALGORITHM:

Step 1: Import Tensorflow, Keras, and Numpy packages.

Step 2: Import fashion dataset and assign to the variable fashion.

Step 3: Load the dataset and train the data set.

Step 4: Assign the index values for the dataset and print image that was tested and trained.

Step 5: Print the flatten, dense for the layers by importing keras dataset.

Step 6: Calculate the accuracy and predicted values for the dataset and print the values.

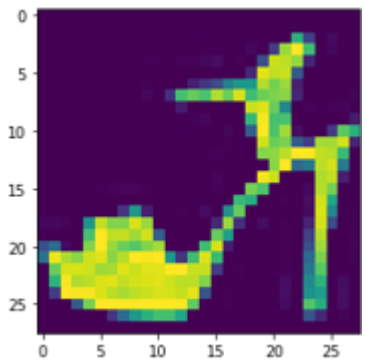
PROGRAM:

```
import tensorflow as tf
from tensorflow import keras
import numpy as np
import matplotlib.pyplot as plt
fashion = keras.datasets.fashion_mnist
(xtrain, ytrain), (xtest, ytest) = fashion.load_data()
imgIndex = 9
image = xtrain[imgIndex]
print("Image Label :",ytrain[imgIndex])
plt.imshow(image)
print(xtrain.shape)
print(xtest.shape)
model = keras.models.Sequential([
```

```
    keras.layers.Flatten(input_shape=[28, 28]),
    keras.layers.Dense(300, activation="relu"),
    keras.layers.Dense(100, activation="relu"),
    keras.layers.Dense(10, activation="softmax")
])
print(model.summary())
xvalid, xtrain = xtrain[:5000]/255.0, xtrain[5000:]/255.0
yvalid, ytrain = ytrain[:5000], ytrain[5000:]
model.compile(loss="sparse_categorical_crossentropy",
              optimizer="sgd",
              metrics=["accuracy"])
history = model.fit(xtrain, ytrain, epochs=30,
                    validation_data=(xvalid, yvalid))
new = xtest[:5]
predictions = model.predict(new)
print(predictions)
classes = np.argmax(predictions, axis=1)
print(classes)
```

Output:

Image Label: 5



(60000, 28, 28)
(10000, 28, 28)

Model: "sequential_3"

Layer (type)	Output Shape	Param #
flatten_3 (Flatten)	(None, 784)	0
dense_6 (Dense)	(None, 300)	235500
dense_7 (Dense)	(None, 100)	30100
dense_8 (Dense)	(None, 10)	1010

Total params: 266,610
Trainable params: 266,610
Non-trainable params: 0

None

Epoch 1/30
1407/1407 [==============================] - 5s 3ms/step - loss: 2.3027 - accuracy: 0.0979 - val_loss: 2.3027 - val_accuracy: 0.1006
Epoch 2/30
1407/1407 [==============================] - 4s 3ms/step - loss: 2.3027 - accuracy: 0.0985 - val_loss: 2.3027 - val_accuracy: 0.1006
Epoch 3/30
1407/1407 [==============================] - 4s 3ms/step - loss: 2.3027 - accuracy: 0.0992 - val_loss: 2.3028 - val_accuracy: 0.1006
Epoch 4/30
1407/1407 [==============================] - 4s 3ms/step - loss: 2.3027 - accuracy: 0.0996 - val_loss: 2.3027 - val_accuracy: 0.1006
Epoch 5/30
1407/1407 [==============================] - 4s 3ms/step - loss: 2.3027 - accuracy: 0.0990 - val_loss: 2.3027 - val_accuracy: 0.1006
Epoch 6/30
1407/1407 [==============================] - 4s 3ms/step - loss: 2.3027 - accuracy: 0.1009 - val_loss: 2.3027 - val_accuracy: 0.1006
Epoch 7/30
1407/1407 [==============================] - 4s 3ms/step - loss: 2.3027 - accuracy: 0.0978 - val_loss: 2.3028 - val_accuracy: 0.0936
Epoch 8/30
1407/1407 [==============================] - 4s 3ms/step - loss: 2.3027 - accuracy: 0.0993 - val_loss: 2.3028 - val_accuracy: 0.1006
Epoch 9/30
1407/1407 [==============================] - 4s 3ms/step - loss: 2.3027 - accuracy: 0.0975 - val_loss: 2.3028 - val_accuracy: 0.1006
Epoch 10/30
1407/1407 [==============================] - 4s 3ms/step - loss: 2.3027 - accuracy: 0.1004 - val_loss: 2.3029 - val_accuracy: 0.1006
Epoch 11/30
1407/1407 [==============================] - 4s 3ms/step - loss: 2.3027 - accuracy: 0.0984 - val_loss: 2.3029 - val_accuracy: 0.1006
Epoch 12/30
1407/1407 [==============================] - 4s 3ms/step - loss: 2.3027 - accuracy: 0.0976 - val_loss: 2.3026 - val_accuracy: 0.1006
Epoch 13/30

1407/1407 [==============================] - 5s 3ms/step - loss: 2.3027 - accuracy: 0.0996 - val_loss: 2.3027 - val_accuracy: 0.1018
Epoch 14/30
1407/1407 [==============================] - 5s 3ms/step - loss: 2.3027 - accuracy: 0.1003 - val_loss: 2.3026 - val_accuracy: 0.1006
Epoch 15/30
1407/1407 [==============================] - 5s 3ms/step - loss: 2.3027 - accuracy: 0.0993 - val_loss: 2.3028 - val_accuracy: 0.1006
Epoch 16/30
1407/1407 [==============================] - 4s 3ms/step - loss: 2.3027 - accuracy: 0.0984 - val_loss: 2.3027 - val_accuracy: 0.1006
Epoch 17/30
1407/1407 [==============================] - 4s 3ms/step - loss: 2.3027 - accuracy: 0.0993 - val_loss: 2.3029 - val_accuracy: 0.1006
Epoch 18/30
1407/1407 [==============================] - 4s 3ms/step - loss: 2.3027 - accuracy: 0.0991 - val_loss: 2.3029 - val_accuracy: 0.1006
Epoch 19/30
1407/1407 [==============================] - 5s 3ms/step - loss: 2.3027 - accuracy: 0.0969 - val_loss: 2.3028 - val_accuracy: 0.1006
Epoch 20/30
1407/1407 [==============================] - 5s 3ms/step - loss: 2.3027 - accuracy: 0.0990 - val_loss: 2.3028 - val_accuracy: 0.1006
Epoch 21/30
1407/1407 [==============================] - 5s 3ms/step - loss: 2.3027 - accuracy: 0.0985 - val_loss: 2.3027 - val_accuracy: 0.1006
Epoch 22/30
1407/1407 [==============================] - 5s 3ms/step - loss: 2.3027 - accuracy: 0.0994 - val_loss: 2.3028 - val_accuracy: 0.1006
Epoch 23/30
1407/1407 [==============================] - 5s 3ms/step - loss: 2.3027 - accuracy: 0.1000 - val_loss: 2.3028 - val_accuracy: 0.1006
Epoch 24/30
1407/1407 [==============================] - 5s 3ms/step - loss: 2.3027 - accuracy: 0.0986 - val_loss: 2.3027 - val_accuracy: 0.1006
Epoch 25/30
1407/1407 [==============================] - 4s 3ms/step - loss: 2.3027 - accuracy: 0.0995 - val_loss: 2.3028 - val_accuracy: 0.1006
Epoch 26/30
1407/1407 [==============================] - 5s 3ms/step - loss: 2.3027 - accuracy: 0.0987 - val_loss: 2.3027 - val_accuracy: 0.1006
Epoch 27/30
1407/1407 [==============================] - 5s 3ms/step - loss: 2.3027 - accuracy: 0.0990 - val_loss: 2.3026 - val_accuracy: 0.1006
Epoch 28/30
1407/1407 [==============================] - 5s 3ms/step - loss: 2.3027 - accuracy: 0.0982 - val_loss: 2.3028 - val_accuracy: 0.1006
Epoch 29/30
1407/1407 [==============================] - 5s 3ms/step - loss: 2.3027 - accuracy: 0.1001 - val_loss: 2.3027 - val_accuracy: 0.1006
Epoch 30/30
1407/1407 [==============================] - 5s 3ms/step - loss: 2.3027 - accuracy: 0.1001 - val_loss: 2.3027 - val_accuracy: 0.1006

[[9.9999964e-01 2.4463721e-32 0.0000000e+00 3.4137151e-09 0.0000000e+00

0.0000000e+00 3.3486748e-07 1.0429964e-13 1.8272030e-38 1.7518649e-36]

[0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00
0.0000000e+00 1.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00]
[0.0000000e+00 1.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00
0.0000000e+00 9.3842902e-14 8.4208933e-12 7.5564799e-34 0.0000000e+00]
[0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00
0.0000000e+00 9.9307209e-01 6.9278893e-03 2.5936122e-36 0.0000000e+00]
[2.3715086e-03 0.0000000e+00 2.3619191e-19 3.4704832e-26 0.0000000e+00
0.0000000e+00 1.2520461e-01 8.7242389e-01 1.0916292e-16 0.0000000e+00]]

[0 6 1 6 7]

RESULT:

Thus, the program to implement Neural networks to classify data using fashion dataset has been executed successfully

Ex. No: 5b	TEXT CLASSIFICATION WITH NEURAL NETWORKS

AIM:

To write a python program to implement Text Classification with Neural Network.

ALGORITHM:

Step 1: Import Keras models, layers, optimizers and adam tokenizer.

Step 2: Import pad_sequence from keras preprocessing.

Step 3: Tokenize each pad_sequence and its text.

Step 4: convert lables to one hot encoded vector using sequences.

Step 5: Split the data into training and testing sets.

Step 6: After train and test, define the Neural network architecture.

Step 7: Train, test evaluates the model.

Step 8: Plot the training and validation accuracy over epochs.

Step 9: Predict classes for test data, create confusion matrix and plot confusion matrix.

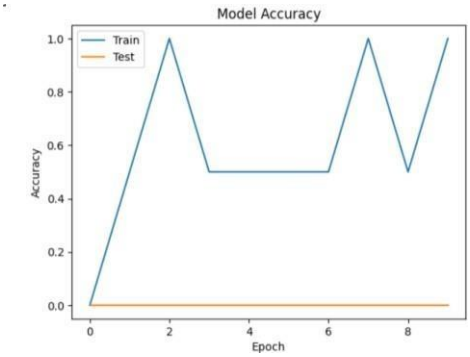
PROGRAM:

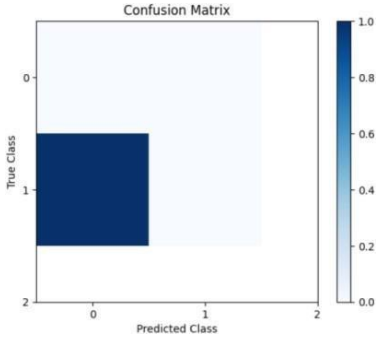
```
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.optimizers import Adam
from keras.preprocessing.text import Tokenizer
#from keras.preprocessing.sequence import pad_sequences
from keras_preprocessing.sequence import pad_sequences
!pip install Keras-Preprocessing


# Tokenize and pad text sequences
max_words = 1000
max_len = 50
tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)
x = pad_sequences(sequences, maxlen=max_len)
```

```
# Convert labels to one-hot encoded vectors
num_classes = len(np.unique(labels))
y = np.zeros((len(labels), num_classes))
for i, label in enumerate(labels):
    y[i, label] = 1

# Split data into training and testing sets
train_ratio = 0.8
num_train = int(train_ratio * len(texts))
x_train, y_train = x[:num_train], y[:num_train]
x_test, y_test = x[num_train:], y[num_train:]

# Define neural network architecture
model = Sequential()
model.add(Dense(64, input_shape=(max_len,), activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
optimizer = Adam(lr=0.01)
model.compile(optimizer=optimizer, loss='categorical_crossentropy',
 metrics=['accuracy'])

# Train model
batch_size = 32
num_epochs = 10
model.fit(x_train, y_train, batch_size=batch_size, epochs=num_epochs,
validation_data=(x_test, y_test))

# Evaluate model
loss, accuracy = model.evaluate(x_test, y_test, batch_size=batch_size)
print('Test loss:', loss)
print('Test accuracy:', accuracy)

import matplotlib.pyplot as plt
# Train model and save history
history = model.fit(x_train, y_train, batch_size=batch_size, epochs=num_epochs,
```

```
 validation_data=(x_test, y_test))

# Plot training and validation accuracy over epochs
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
from sklearn.metrics import confusion_matrix

# Predict classes for test data
y_pred = model.predict(x_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true_classes = np.argmax(y_test, axis=1)

# Create confusion matrix
conf_matrix = confusion_matrix(y_true_classes, y_pred_classes)

# Plot confusion matrix
plt.imshow(conf_matrix, cmap=plt.cm.Blues, interpolation='nearest')
plt.title('Confusion Matrix')
plt.colorbar()
tick_marks = np.arange(num_classes)
plt.xticks(tick_marks, range(num_classes))
plt.yticks(tick_marks, range(num_classes))
plt.xlabel('Predicted Class')
plt.ylabel('True Class')
plt.show()
```

OUTPUT:





RESULT:

Thus, the program to classify text using neural network has been executed successfully.

Ex. No: 6a	IMPLEMENT THE NAIVE BAYES CLASSIFIER FOR A SAMPLE DATASET STORED AS A .CSV FILE

AIM:

To find the Naive Bayes classifier using Sklearn for given data set.

ALGORITHM:

Naïve Bayes algorithm is a supervised learning algorithm, which is based on Bayes theorem and used for solving classification problems.

Step 1: Convert the data set into a frequency table

Step 2: Create Likelihood table by finding the probabilities

Step 3: Now, use Naive Bayesian equation to calculate the posterior probability for each class.

PROGRAM:

```
from google.colab import files
uploaded=files.upload()
import numpy as np
import pandas as pd
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
play_tennis = pd.read_csv("play_tennis.csv")
play_tennis.head()
number = LabelEncoder()
play_tennis['outlook'] = number.fit_transform(play_tennis['outlook'])
play_tennis['temp'] = number.fit_transform(play_tennis['temp'])
play_tennis['humidity'] = number.fit_transform(play_tennis['humidity'])
play_tennis['wind'] = number.fit_transform(play_tennis['wind'])
play_tennis['play'] = number.fit_transform(play_tennis['play'])
features = ["outlook", "temp", "humidity", "wind"]
target = "play"
features_train, features_test, target_train, target_test = train_test_split(play_tennis
```

```
[features],play_tennis[target],test_size = 0.33,random_state = 54)
model = GaussianNB()
print(model.fit(features_train, target_train))
pred  = model.predict(features_test)
accuracy = accuracy_score(target_test, pred)
print(accuracy)
print(pred)
print(model.predict([[1,2,0,1]]))
```

Output:

```
GaussianNB()
0.8
[1 1 1 1 1]
[1]
```

RESULT:

Thus, the program to find Naïve Bayes classifier using Sklearn for given dataset has been executed successfully.

Ex. No: 6b	CONSTRUCT A BAYESIAN NETWORK CONSIDERING MEDICAL DATA

AIM:

To write a Python program to implement a Bayesian network.

ALGORITHM:

Step 1: Start the program.

Step 2: Go to command prompt a Jupyter notebook a new a python3.

Step 3: Import the necessary libraries.

Step 4: Define the Bayesian network structure and the Conditional Probability Distributions (CPD).

Step 5: Associate the CPD with the network structure.

Step 6: Infer the posterior probability and save the model as .jpg format to view.

Step 7: Stop the program.

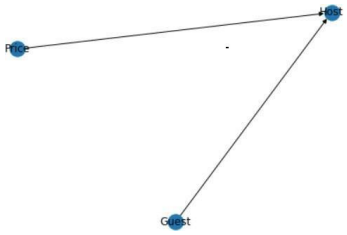
PROGRAM:

```
!pip install pgmpy
from pgmpy.models import BayesianNetwork
from pgmpy.factors.discrete import TabularCPD
import networkx as nx
import pylab as plt
model = BayesianNetwork([('Guest', 'Host'), ('Price', 'Host')])
cpd_guest = TabularCPD('Guest', 3, [[0.33], [0.33], [0.33]])
cpd_price = TabularCPD('Price', 3, [[0.33], [0.33], [0.33]])
cpd_host = TabularCPD('Host', 3, [[0, 0, 0, 0, 0.5, 1, 0, 1, 0.5],
                                  [0.5, 0, 1, 0, 0, 0, 1, 0, 0.5],
                                  [0.5, 1, 0, 1, 0.5, 0, 0, 0, 0]],
                      evidence=['Guest', 'Price'],evidence_card=[3, 3])
model.add_cpds(cpd_guest, cpd_price, cpd_host)
model.check_model()
from pgmpy.inference import VariableElimination
```

```
infer = VariableElimination(model)
posterior_p = infer.query(['Host'], evidence={'Guest': 2, 'Price': 2})
print(posterior_p)
nx.draw(model,with_labels=True)
plt.savefig('model.jpg')
plt.close()
```

Output:

```
True
+---------+-------------+
| Host    |   phi(Host) |
+=========+=============+
| Host(0) |      0.5000 |
+---------+-------------+
| Host(1) |      0.5000 |
+---------+-------------+
| Host(2) |      0.0000 |
+---------+-------------+
```



RESULT:

Thus, the program to implement a Bayesian network has been executed successfully.

Ex. No: 7	ARTIFICIAL NEURAL NETWORK

AIM:

To build an artificial neural network by implementing the backpropagation algorithm and test the same using appropriate data sets.

ALGORITHM:

Step 1: Go to command prompt→ jupyter notebook→ new →python 3.

Step 2: Import the necessary libraries.

Step 3: Define the sigmoid and derivative of sigmoid function function and initialize the variables.

Step 4: Perform background propagation, forward propagation for a random range of numbers.

Step 5: Print the actual and predicted output.

Step 6: stop the program.

PROGRAM:

```
import numpy as np
X = np.array(([2,9], [1,5],[3,6]),dtype=float)
y=np.array(([92],[86],[89]),dtype=float)
X = X/np.amax(X,axis=0)
y=y/100
def sigmoid(x):
    return 1/(1+np.exp(-x))
def derivatives_sigmoid(x):
    return x*(1-x)
epoch=5000
lr=0.1
inputlayer_neurons=2
hiddenlayer_neurons=3
output_neurons=1
wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))
```

```
bh=np.random.uniform(size=(1,hiddenlayer_neurons))
wout=np.random.uniform(size=(hiddenlayer_neurons,output_neurons))
bout=np.random.uniform(size=(1,output_neurons))
for i in range(epoch):
    hinp1=np.dot(X,wh)
    hinp=hinp1 + bh
    hlayer_act=sigmoid(hinp)
    outinp1=np.dot(hlayer_act,wout)
    outinp= outinp1+bout
    output = sigmoid(outinp)
    EO=y-output
    outgrad=derivatives_sigmoid(output)
    d_output=EO*outgrad
    EH=d_output.dot(wout.T)
    hiddengrad=derivatives_sigmoid(hlayer_act)
    d_hiddenlayer=EH*hiddengrad
    wout+=hlayer_act.T.dot(d_output)*lr
    wh+=X.T.dot(d_hiddenlayer)*lr
print('Input: \n'+str(X))
print('Actual Output: \n'+str(y))
print('Predicted Output: \n',output)
```

Output:

```
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.89552852]
 [0.8793998 ]
 [0.89495923]]
```

RESULT:

Thus, the program to Artificial Neural Network by implementing the back propagation algorithm has been executed successfully.

Ex. No: 8	SVM CLASSIFIER

AIM:

To write a python program to determine whether a person is malignant to cancer or not by implementing the SVM algorithm.

ALGORITHM:

Support Vector Machine (SVM) is a supervised machine learning algorithm used for both classification and regression.

Step 1: Import the dataset and print the details.

Step 2: Splitting the dataset into training and test set.

Step 3: Fit the SVM classifier to the training set.

Step 4: Predict the test set result.

Step 5: Print the metrics and classification report.

PROGRAM:

```
from sklearn import datasets
cancer_data = datasets.load_breast_cancer()
print(cancer_data.data[5])
print(cancer_data.data.shape)
print(cancer_data.target)
from sklearn.model_selection import train_test_split
cancer_data = datasets.load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(cancer_data.data, cancer_data.target, test
_size=0.4,random_state=109)
from sklearn import svm
cls = svm.SVC(kernel="linear")
cls.fit(X_train,y_train)
pred = cls.predict(X_test)
from sklearn import metrics
print("acuracy:", metrics.accuracy_score(y_test,y_pred=pred))
```

```
print("recall" , metrics.recall_score(y_test,y_pred=pred))
print(metrics.classification_report(y_test, y_pred=pred))
```

Output:

```
[1.245e+01 1.570e+01 8.257e+01 4.771e+02 1.278e-01 1.700e-01 1.578e-01
 8.089e-02 2.087e-01 7.613e-02 3.345e-01 8.902e-01 2.217e+00 2.719e+01
 7.510e-03 3.345e-02 3.672e-02 1.137e-02 2.165e-02 5.082e-03 1.547e+01
 2.375e+01 1.034e+02 7.416e+02 1.791e-01 5.249e-01 5.355e-01 1.741e-01
 3.985e-01 1.244e-01]
(569, 30)
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 1 0 0 0 0 0 0 0 0 1 0 1 1 1 1 1 0 0 1 0 0 1 1 1 1 0 1 0 0 1 1 1 1 0 1 0 0
 1 0 1 0 0 1 1 1 0 0 1 0 0 0 1 1 1 0 1 1 0 0 1 1 1 1 0 0 1 1 1 1 0 1 1 0 1 1
 1 1 1 1 1 1 0 0 0 1 0 0 1 1 1 0 0 1 0 1 0 0 1 0 0 1 1 0 1 1 0 1 1 1 1 1 0 1
 1 1 1 1 1 1 1 1 0 1 1 1 1 0 0 1 0 1 1 0 0 1 1 0 0 1 1 1 1 0 1 1 0 0 0 1 0
 1 0 1 1 1 0 1 1 0 0 1 0 0 0 0 1 0 0 0 1 0 1 0 1 1 0 1 0 0 0 0 1 1 0 0 1 1
 1 0 1 1 1 1 1 0 0 1 1 0 1 1 0 0 1 0 1 1 1 1 0 1 1 1 1 1 0 1 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 1 1 1 1 1 1 1 0 1 0 1 1 0 1 1 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1
 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 0 1 1 1 1 0 0 0 1 1
 1 1 0 1 0 1 0 1 1 1 0 1 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0 0
 0 1 0 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 0 1 1 0 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1
 1 0 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 0 1 1 1 1 1 0 1 1
 0 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1
 1 1 1 1 1 1 0 1 0 1 1 0 1 1 1 1 1 1 0 0 1 0 1 0 1 1 1 1 1 0 1 1 0 1 0 1 0 0
 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 0 0 0 0 0 0 1]
acuracy: 0.9649122807017544
recall 0.9782608695652174
              precision    recall  f1-score   support

           0       0.97      0.94      0.96        90
           1       0.96      0.98      0.97       138

    accuracy                           0.96       228
   macro avg       0.97      0.96      0.96       228
weighted avg       0.96      0.96      0.96       228
```

RESULT:

Thus, the program to determine whether a person is malignant to cancer or not by implementing the SVM algorithm has been executed successfully.

Ex. No: 9	APPLY EM ALGORITHM TO CLUSTER A SET OF DATA STORED IN A .CSV FILE

AIM:

To write a Python program to implement Expectation - Maximization (EM) algorithm for the iris dataset.

ALGORITHM:

The Expectation-Maximization (EM) algorithm is defined as the combination of various unsupervised machine learning algorithms, which is used to determine the local maximum likelihood estimates (MLE) or maximum a posteriori estimates (MAP) for unobservable variables in statistical models.

Step 1: Given a set of incomplete data, consider a set of starting parameters.

Step 2: Expectation step (E – step): Using the observed available data of the dataset, estimate (guess) the values of the missing data.

Step 3: Maximization step (M – step): Complete data generated after the expectation (E) step is used to update the parameters.

Step 4: Repeat step 2 and step 3 until convergence.

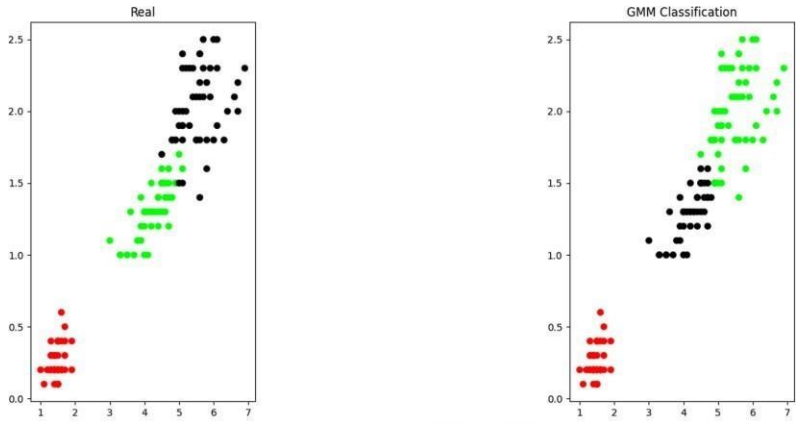
PROGRAM:

```
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
import sklearn.metrics as metrics
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
names=['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width','Class']
dataset = pd.read_csv("8-dataset.csv", names=names)
X = dataset.iloc[:, :-1]
label = {'Iris-setosa': 0,'Iris-versicolor':1,'Iris-virginica': 2}
y = [label[c] for c in dataset.iloc[:, -1]]
plt.figure(figsize=(14,7))
```

```
colormap=np.array(['red','lime','black'])
plt.subplot(1,3,1)
plt.title('Real')
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y])
gmm=GaussianMixture(n_components=3, random_state=0).fit(X)
y_cluster_gmm=gmm.predict(X)
plt.subplot(1,3,3)
plt.title('GMM Classification')
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y_cluster_gmm])
print('The accuracy score of EM: ',metrics.accuracy_score(y, y_cluster_gmm))
print('The Confusion matrix of EM:\n ',metrics.confusion_matrix(y, y_cluster_gmm))
```

Output:

```
The accuracy score of EM:  0.36666666666666664
The Confusion matrix of EM:
  [[50  0  0]
 [ 0  5 45]
 [ 0 50  0]]
```



RESULT:

Thus, the Python program to implement Expectation - Maximization (EM) algorithm for the iris dataset.

Ex. No: 10	BAYESIAN INFERENCE IN GENE EXPRESSION ANALYSIS

AIM:

To perform the Bayesian inference for gene expression analysis.

ALGORITHM:

Step-1: Start the program.

Step-2: Import the libraries.

Step-3: Load the single-molecule mRNA FISH data.

Step-4: Define functions for calculating the log likelihood, prior posterior and probabilities formRNA distribution.

Step-5: Evaluate thermodynamic constraint.

Step-6: Read the MCMC chain and plot a corner plot.

Step-7: Compare the empirical and theoretical prediction.

Step-8: Stop the program.

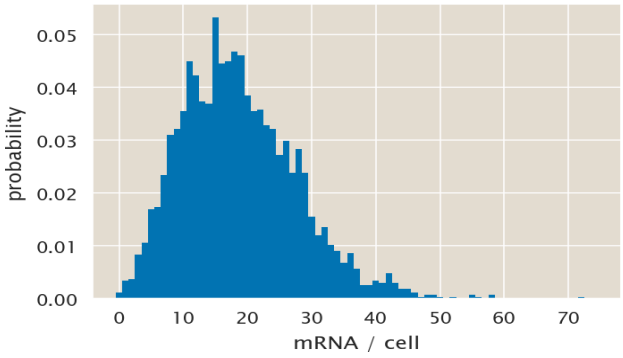
PROGRAM:

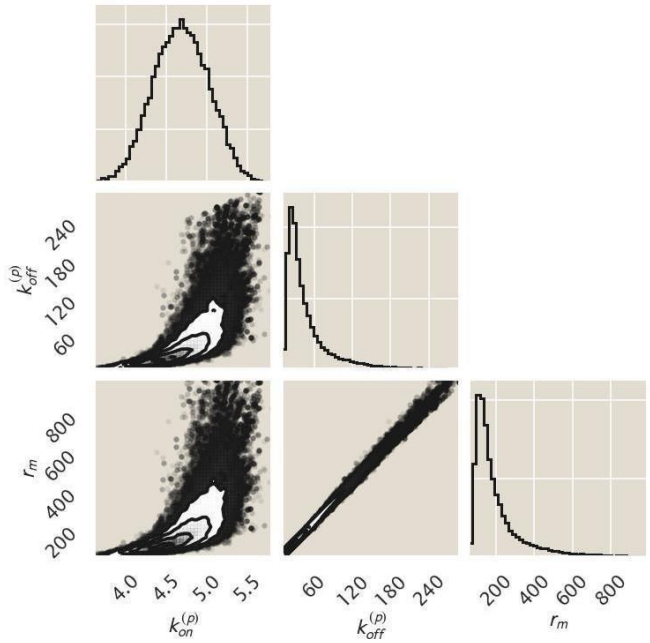
```
!pip install pymc3
#get_ipython().system('pip install pymc3')
import numpy as np
import pymc3 as pm
from pymc3 import *
import matplotlib.pyplot as plt
# Generate synthetic gene expression data
np.random.seed(42)
n_samples = 100 # Number of samples
n_genes = 500 # Number of genes
n_conditions = 2  # Number of conditions
true_expression = np.random.normal(loc=0, scale=1, size=(n_samples, n_genes))
condition_effects = np.array([-1, 1]) # True condition effects
true_condition = np.random.randint(0, 2, size=n_samples)
observed_expression = true_expression + condition_effects[true_condition][:, np.newaxis]
```

```
# Bayesian model
with pm.Model() as gene_model:

    # Priors for parameters
    condition_effect = pm.Normal('condition_effect', mu=0, sd=1, shape=n_conditions)
    gene_expression = pm.Normal('gene_expression', mu=condition_effect[true_condition
    [:, np.newaxis]], sd=1, shape=(n_samples, n_genes), observed=observed_expression)

    # Sampling
    trace = pm.sample(1000, tune=1000)

# Posterior analysis
pm.summary(trace)
pm.traceplot(trace)

# Plotting
plt.figure(figsize=(12, 6))
plt.hist(trace['condition_effect'][:, 0], bins=30, density=True, alpha=0.5, color='red',
label='Condition 0')
plt.hist(trace['condition_effect'][:, 1], bins=30, density=True, alpha=0.5, color='blue',
label='Condition 1')
plt.xlabel('Condition Effect')
plt.ylabel('Density')
plt.title('Posterior Distribution of Condition Effects')
plt.legend()
plt.show()
```

Output:

	area_cells	date	experiment	mRNA_cell	num_intens_totals	spots_totals
0	402	20111220	UV5	27	4.544086	21
1	288	20111220	UV5	19	3.196886	14
2	358	20111220	UV5	25	4.249250	19
3	310	20111220	UV5	30	5.075867	22
4	300	20111220	UV5	31	5.361156	24





RESULT:

Thus, the program to perform the Bayesian inference for gene expression analysis has been executed successfully.

CONTENT BEYOND SYLLABUS

Ex. No: 11	IMPLEMENTATION OF A SIMPLE GAN

AIM:

To perform the program to generate fake images by the implementation of a simple GAN.

ALGORITHM:

Step-1: Define the generator and discriminator networks using PyTorch's nn.Module class.

Step-2: Initialize the generator and discriminator networks.

Step-3: Define the loss function for the GAN. In this case, binary cross-entropy loss is used.

Step-4: Define the optimizers for the generator and discriminator networks.

Step-5: Generate some fake images using the trained generator (fake_images = G(torch.Tensor(np.random.normal(0, 1, (10, 100)))).detach().numpy()).

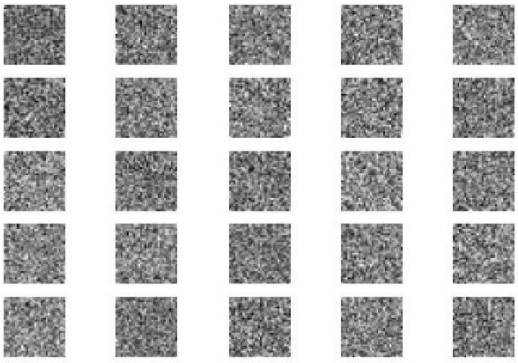
PROGRAM:

```
import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np

# Define the generator network
class Generator(nn.Module):
    def__init_(self):
        super(Generator, self)._init ()
        self.fc1 = nn.Linear(100, 128)
        self.fc2 = nn.Linear(128, 256)
        self.fc3 = nn.Linear(256, 784)
        self.relu = nn.ReLU()
        self.sigmoid = nn.Sigmoid()

    def forward(self, z):
        x = self.relu(self.fc1(z))
        x = self.relu(self.fc2(x))
        x = self.sigmoid(self.fc3(x))
        return x

# Define the discriminator network
class Discriminator(nn.Module):
    def__init_(self):
        super(Discriminator, self)._init_()
        self.fc1 = nn.Linear(784, 256)
        self.fc2 = nn.Linear(256, 128)
        self.fc3 = nn.Linear(128, 1)
```

```
        self.relu = nn.ReLU()
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.relu(self.fc1(x))
        x = self.relu(self.fc2(x))
        x = self.sigmoid(self.fc3(x))
        return x

# Initialize the generator and discriminator
G = Generator()
D = Discriminator()

# Define the loss function and optimizer
criterion = nn.BCELoss()
optimizer_G = optim.Adam(G.parameters(), lr=0.0002)
optimizer_D = optim.Adam(D.parameters(), lr=0.0002)

# Train the GAN
for epoch in range(200):
    for i in range(60000):
        # Train the discriminator with real data
        D.zero_grad()
        real_data = torch.Tensor(np.random.normal(0, 1, (128, 784)))
        real_target = torch.ones((128, 1))
        real_output = D(real_data)
        loss_real = criterion(real_output, real_target)
        loss_real.backward()

        # Train the discriminator with fake data
        fake_data = G(torch.Tensor(np.random.normal(0, 1, (128, 100))))
        fake_target = torch.zeros((128, 1))
        fake_output =  D(fake_data.detach())
        loss_fake = criterion(fake_output, fake_target)
        loss_fake.backward()

        # Update the discriminator
        optimizer_D.step()

        # Train the generator
        G.zero_grad()
        fake_target = torch.ones((128, 1))
        fake_output = D(fake_data)
        loss_G = criterion(fake_output, fake_target)
        loss_G.backward()

        # Update the generator
        optimizer_G.step()

    # Print the losses for each epoch
    print('Epoch %d: D_loss=%.4f, G_loss=%.4f' % (epoch+1, (loss_real+loss_fake).item(),
    loss_G.item()))
```

```
# Generate some fake images using the trained generator
fake_images = G(torch.Tensor(np.random.normal(0, 1, (10)))
```

Output:



RESULT:

Thus, the program to generate fake images by the implementation of a simple GAN has been executed successfully.