LAKSHMI NARAIN COLLEGE OF TECHNOLOGY & SCIENCE, BHOPAL

CS-601

MACHINE LEARNING

Vision of the Department

To be recognized for keeping innovation, research and excellence abreast of learning in the field of computer science & engineering to cater the global society.

Mission of the Department

M1: To provide an exceptional learning environment with academic excellence in the field of computer science and engineering.

M2: To facilitate the students for research and innovation in the field of software, hardware and computer applications and nurturing to cater the global society.

M3: To establish professional relationships with industrial and research organisations to enable the students to be updated of the recent technological advancements.

M4: To groom the learners for being the software professionals catering the needs of modern society with ethics, moral values and full of patriotism.

Program Educational Objectives (PEO's)

PEO1: The graduate will have the knowledge and skills of major domains of computer science and engineering in providing solution to real world problems most efficiently.

PEO2: The graduate will be able to create and use the modern tools and procedures followed in the software industry in the relevant domain.

PEO3: The graduate will be following the ethical practices of the software industry and contributing to the society as a responsible citizen.

PEO4: The graduate will have the innovative mindset of learning and implementing the latest developments and research outcomes in the computer hardware and software to keep pace with the fast changing socio economic world.

LAKSHMI NARAIN COLLEGE OF TECHNOLOGY & SCIENCE, BHOPAL

CS-601

MACHINE LEARNING

Course Outcomes

Student will be able to

CO601.1: Apply knowledge of computing and mathematics to machine learning problems, models and algorithms

CS601.2: Analyze a problem and identify the computing requirements for ML model.

CO601.3: Design and implement an algorithm or ML model for accurate outcome.

CO601.4: Evaluate efficiency and related parameters of various ML models.

CO601.5: Differentiate ML models to demonstrate comprehension of the trade-offs involved.

List of Experiments

1. a)Implementation of Python Basic Libraries such as Math, Numpy and Scipy.
   b) Implementation of Python Libraries for ML application such as Pandas and Matplotlib.
2. Extract the data from database using python.
3. The probability that it is Friday and that a student is absent is 3 %. Since there are 5 school days in a week, the probability that it is Friday is 20 %. What is the probability that a student is absent given that today is Friday? Apply Baye's rule in python to get the result.
4. Implement linear regression using python.
5. Build an Artificial Neural Network by implementing the back propagation algorithm and test the same using appropriate data sets.
6. Implement K-nearest neighbor classification using python.
7. Design convolution neural network for image classification in presence of limited data.
8. Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

CS-601

MACHINE LEARNING

9. Classifying data using Support Vector Machines(SVMs) in python.
10. Write a Program to construct Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard heart disease data set. You can use standard heart disease data set. Use python ML Library Classes/API.
11. Assuming a set of documents that need to be classified, Use the Naïve Bayesian classifier model to perform this task. Built-in java /API can be used to write the program calculate the accuracy, precision and recall for your dataset.

CS-601

MACHINE LEARNING

EXPERIMENT -1

Aim: a) Implementation of Python Basic Libraries such as Math, Numpy and Scipy.

b) Implementation of Python Libraries for ML application such as Pandas and Matplotlib.

THEORY

Python

Python is a high-level, general-purpose programming language.

It is used for:

- Web development (server-side),
- Software development,
- Mathematics,
- System scripting.

Libraries

A Python library is a collection of related modules. It contains bundles of code that can be used repeatedly in different programs. It makes Python Programming simpler and convenient for the programmer.

Libraries of Python used for Machine Learning

- Math
- Numpy
- Scipy
- Pandas
- Matplotlib

A) Implementation of Python Basic Libraries such as Math, Numpy and Scipy.

Math

To carry out calculations with real numbers, the Python language contains many additional functions collected in a library (module) called math.To use these functions at the beginning of the program, you need to connect the math library, which is done by the command.

import math

Python provides various operators for performing basic calculations, such as * for multiplication,% for a module, and / for the division. If you are developing a program in Python to perform certain tasks, you need to work with trigonometric functions, as well as complex numbers. Although you cannot use these functions directly, you can access them by turning on the math module math, which gives access to hyperbolic, trigonometric and logarithmic functions for real numbers. To use complex numbers, you can use the math module cmath. When comparing *math* vs *numpy*, a *math* library is more lightweight and can be used for extensive computation as well.

Example: Getting Pi Value

```
>>> import math
>>>math.pi
  3.141592653589793
```

Numpy

Mathematical algorithms implemented in interpreted languages, for example, Python, often work much slower than the same algorithms implemented in compiled languages (for example, Fortran, C, and Java). The NumPy library provides implementations of computational algorithms in the form of functions and operators, optimized for working with multidimensional arrays. As a result, any algorithm that can be expressed as a sequence of operations on arrays (matrices) and implemented using NumPy works as fast as the equivalent code executed in MATLAB. If we compare *numpy* vs *math, we quickly find thatnumpy* has more advantages for computation methods compared to *math*.

Example: **how to work with linear algebra with numpy**.

```
>>> import numpy as np
>>> a = np.array([[1.0, 2.0], [3.0, 4.0]])
>>> print(a)
[[ 1.  2.]
 [ 3.  4.]]
>>> a.transpose()
array([[ 1.,  3.],
       [ 2.,  4.]])
>>> np.linalg.inv(a)
array([[-2. ,  1. ],
       [ 1.5, -0.5]])
>>> u = np.eye(2) # unit 2x2 matrix; "eye" represents "I"
>>> u
array([[ 1.,  0.],
       [ 0.,  1.]])
>>> j = np.array([[0.0, -1.0], [1.0, 0.0]])
>>> j @ j        # matrix product
array([[-1.,  0.],
       [ 0., -1.]])
>>> np.trace(u)  # trace
2.0
>>> y = np.array([[5.], [7.]])
>>> np.linalg.solve(a, y)
```

```
array([[-3.],
       [ 4.]])
>>> np.linalg.eig(j)
(array([ 0.+1.j,  0.-1.j]), array([[ 0.70710678+0.j,0.70710678-0.j],
       [ 0.00000000-0.70710678j,  0.00000000+0.70710678j]]))
```

Scipy

SciPy is a library for the open-source Python programming language, designed to perform scientific and engineering calculations.
SciPy in Python is a collection of mathematical algorithms and functions built as a Numpy extension. It greatly extends the capabilities of an interactive Python session by providing the user with high-level commands and classes for managing and visualizing data. With SciPy, an interactive Python session becomes a data processing and prototyping system competing with systems such as MATLAB, IDL, Octave, R-Lab, and SciLab.

Example: how to calculate effectively on scipy

```
>>> p = poly1d([3,4,5])
>>> print(p)
   2
3 x + 4 x + 5
>>> print(p*p)
   4      3      2
```

```
9 x + 24 x + 46 x + 40 x + 25
>>> print(p.integ(k=6))
   3     2
1 x + 2 x + 5 x + 6
>>> print(p.deriv())
6 x + 4
>>> p([4, 5])
array
```

B) Implementation of Python Libraries for ML application such as Pandas and Matplotlib.

PANDAS: Pandas is a Python library used for working with data sets. It has functions for analyzing, cleaning, exploring, and manipulating data.Pandas allows us to analyze big data and make conclusions based on statistical theories.It can clean messy data sets, and make them readable and relevant.

Relevant data is very important in data science.

Example: Creating a Series by passing a list of values, letting pandas create a default integer index:

```
In [3]: s = pd.Series([1, 3, 5, np.nan, 6, 8])

In [4]: s
Out[4]:
0    1.0
1    3.0
2    5.0
3    NaN
4    6.0
5    8.0
dtype: float64
```

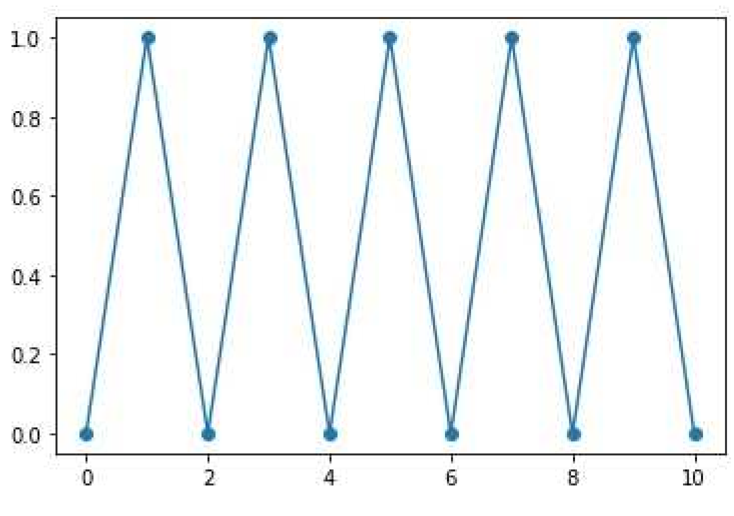
Matplotlib

Matplotlib is a Python library that helps in visualizing and analyzing the data and helps in better understanding of the data with the help of graphical, pictorial visualizations that can be simulated using the matplotlib library. Matplotlib is a comprehensive library for static, animated and interactive visualizations.

Example: simple plot

```
import matplotlib.pyplot as plt

x = [0,1,2,3,4,5,6,7,8,9,10]
y = [0,1,0,1,0,1,0,1,0,1,0]
plt.plot(x, y, marker="o")
plt.show()
```



EXPERIMENT -2

Aim: Extract the data from database using python

ALGORITHM:

Step 1: Connect to MySQL from Python

Step 2: Define a SQL SELECT Query

Step 3: Get Cursor Object from Connection

Step 4: Execute the SELECT query using execute() method

Step 5: Extract all rows from a result

Step 6: Iterate each row

Step 7: Close the cursor object and database connection object

Step 8: End.

PROCEDURE

CREATING A DATABASE IN MYSQL AS FOLLOWS:

```
CREATE DATABASE myDB;

SHOW DATABASES;

USE myDB

CREATE TABLE MyGuests (id INT, name VARCHAR(20), email VARCHAR(20));

SHOW TABLES;

INSERT INTO MyGuests (id,name,email) VALUES(1,"sairam","xyz@abc.com");

…

SELECT * FROM authors;
```

We need to install mysql-connector to connect Python with MySQL. You can use the below command to

install this in your system.

```
pip install mysql-connector-python-rf
```

PYTHON SOURCE CODE:

```
import mysql.connector

mydb = mysql.connector.connect(

host="localhost",

user="root",

password="",

database="myDB"
```

)

15

Machine Learning Lab

```
mycursor = mydb.cursor()

mycursor.execute("SELECT * FROM MyGuests")

myresult = mycursor.fetchall()

for x in myresult:

print(x)
```

EXPERIMENT -3

Aim: The probability that it is Friday and that a student is absent is 3 %. Since there are 5 school days in a week, the probability that it is Friday is 20 %. What is the probability that a student is absent given that today is Friday? Apply Baye’s rule in python to get the result**.**

ALGORITHM:

```
Step 1: Calculate probability for each word in a text and filter the words which have a probability less than threshold probability. Words with probability less than threshold probability are irrelevant.
Step 2: Then for each word in the dictionary, create a probability of that word being in insincere questions and its probability insincere questions. Then finding the conditional probability to use in naive Bayes classifier.
Step 3: Prediction using conditional probabilities.
Step 4: End
```

PROGRAM:

```
PFIA=float(input(“Enter probability that it is Friday and that a student is absent=”))
PF=float(input(“ probability that it is Friday=”))
PABF=PFIA / PF
print(“probability that a student is absent given that today is Friday using conditional probabilities=”,PABF)
```

OUTPUT:

```
Enter probability that it is Friday and that a student is absent= 0.03

Probability that it is Friday= 0.2

 Probability that a student is absent given that today is Friday using conditional
probabilities= 0.15
```

EXPERIMENT -4

Aim: Implement linear regression using python.

THEORY

Linear regression algorithm shows a linear relationship between a dependent (y) and one or more independent (y) variables, hence called as linear regression. Since linear regression shows the linear relationship, which means it finds how the value of the dependent variable is changing according to the value of the independent variable.

ALGORITHM:

Step 1: Create Database for Linear Regression

Step 2: Finding Hypothesis of Linear Regression

Step 3: Training a Linear Regression model

Step 4: Evaluating the model

Step 5: Scikit-learn implementation

Step 6: End

PROGRAM:

```
# Importing Necessary Libraries

 import numpy as np

import matplotlib.pyplot as plt

from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

 # generate random data-set

np.random.seed(0)

x = np.random.rand(100, 1) #Generate a 2-D array with 100 rows, each row containing 1 random
numbers:

y = 2 + 3 * x + np.random.rand(100, 1)

regression_model = LinearRegression() # Model initialization

 regression_model.fit(x, y) # Fit the data(train the model)

y_predicted = regression_model.predict(x) # Predict

# model evaluation

rmse = mean_squared_error(y, y_predicted)

r2 = r2_score(y, y_predicted)

# printing values

print('Slope:' ,regression_model.coef_)

print('Intercept:', regression_model.intercept_)

print('Root mean squared error: ', rmse)

print('R2 score: ', r2)

 # plotting values # data points

plt.scatter(x, y, s=10)

plt.xlabel('x-Values from 0-1')

plt.ylabel('y-values from 2-5')

 # predicted values
```

```
plt.plot(x, y_predicted, color='r')

plt.show() )
```

EXPERIMENT -5

Aim: Build an Artificial Neural Network by implementing the back propagation algorithm and test the same using appropriate data sets.

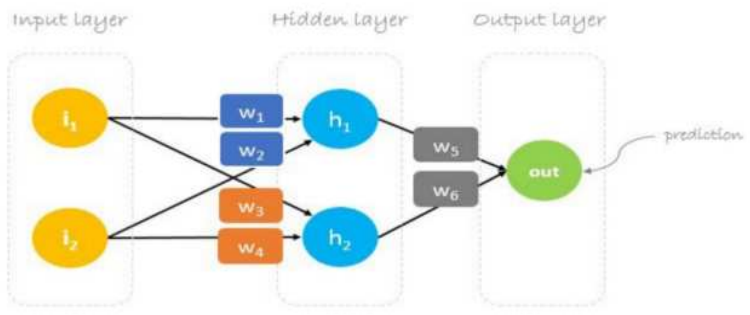
THEORY

Back-propagation is the essence of neural net training. It is the method of fine-tuning the weights of a neural net based on the error rate obtained in the previous epoch (i.e., iteration). Proper tuning of the weights allows you to reduce error rates and to make the model reliable by increasing its generalization. Back-propagation is a short form for “backward propagation of errors.” It is a standard method of training artificial neural networks. This method helps to calculate the gradient of a loss function with respects to all the weights in the network.

ALGORITHM

Back propagation Algorithm:

```
1.Load data set
2. Assign all network inputs and output
3.Initialize all weights with small random numbers, typically between -1 and 1
repeat
for every pattern in the training set
Present the pattern to the network
// Propagated the input forward through the network:
for each layer in the network
for every node in the layer
1. Calculate the weight sum of the inputs to the node
2. Add the threshold to the sum
3. Calculate the activation for the node
end
end
// Propagate the errors backward through the network
for every node in the output layer calculate the error signal end
for all hidden layers
for every node in the layer
1. Calculate the node's signal error
2. Update each node's weight in the network
end
end
// Calculate Global Error
Calculate the Error
Function
```



- Input layer with two inputs neurons
- One hidden layer with two neurons
- Output layer with a single neuron

SOURCE CODE (IF INCLUDED)

```
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) # maximum of X array longitudinally y = y/100

#Sigmoid

Function def

sigmoid (x):

return (1/(1 + np.exp(-x)))

#Derivative of Sigmoid Function

def derivatives_sigmoid(x):

return x * (1 - x)

#Variable initialization

epoch=7000                        #Setting training iterations

lr=0.1                            #Setting learning rate

inputlayer_neurons = 2            #number of features in data set
```

hiddenlayer_neurons = 3 #number of hidden layers neurons

output_neurons = 1 #number of neurons at output layer

#weight and bias initialization 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))

# draws a random range of numbers uniformly of dim

x*y #Forward Propagation

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)

#Backpropagation

EO = y-output

outgrad = derivatives_

sigmoid(output) d_output = EO*

outgrad

EH = d_output.dot(wout.T)

hiddengrad = derivatives_sigmoid(hlayer_act)

#how much hidden layer wts contributed to

```
error

d_hiddenlayer = EH * hiddengrad

wout += hlayer_act.T.dot(d_output)

*lr

# dotproduct of nextlayererror and currentlayerop

bout += np.sum(d_output, axis=0,keepdims=True) *lr

wh += X.T.dot(d_hiddenlayer) *lr

#bh += np.sum(d_hiddenlayer, axis=0,keepdims=True) *lr

print("Input: \n" + str(X)
```

Output:

```
Input:

[[ 0.66666667 1. ]

[ 0.33333333 0.55555556]

[ 1. 0.66666667]]

Actual

Output: [[

0.92]

[ 0.86]

[ 0.89]]

Predicted

Output: [[

0.89559591]

[ 0.88142069]

 [ 0.8928407 ]
```

CS-601

MACHINE LEARNING

EXPERIMENT -6

Aim: Implement k-nearest neighbor classification using python

THEORY

KNN is a simple, supervised machine learning (ML) algorithm that can be used for classification or regression tasks - and is also frequently used in missing value imputation. It is based on the idea that the observations closest to a given data point are the most "similar" observations in a data set, and we can therefore classify unforeseen points based on the values of the closest existing points. By choosing *K*, the user can select the number of nearby observations to use in the algorithm.

ALGORITHM:

Step 1: Load the data
Step 2: Initialize the value of k
Step 3: For getting the predicted class, iterate from 1 to total number of training data points

i) Calculate the distance between test data and each row of training data. Here we will use Euclidean
distance as our distance metric since it's the most popular method. The other metrics that can be
used are Chebyshev, cosine, etc.
ii) Sort the calculated distances in ascending order based on distance values 3. Get top k rows from the
sorted array
iii) Get the most frequent class of these rows i.e. Get the labels of the selected K entries
iv) Return the predicted class

- If regression, return the mean of the K labels
- If classification, returnthe mode of the K labels
- If regression, return the mean of the K labels
- If classification, return the mode of the K labels

Step 4: End.

PROGRAM

```
import numpy as np

from sklearn import datasets
```

```
iris = datasets.load_iris()

data = iris.data

labels = iris.target

for i in [0, 79, 99, 101]:

print(f"index: {i:3}, features: {data[i]}, label: {labels[i]}")

np.random.seed(42)

indices = np.random.permutation(len(data))

n_training_samples = 12

learn_data = data[indices[:-n_training_samples]]

learn_labels = labels[indices[:-n_training_samples]]

test_data = data[indices[-n_training_samples:]]

test_labels = labels[indices[-n_training_samples:]]

print("The first samples of our learn set:")

print(f"{'index':7s}{'data':20s}{'label':3s}")

for i in range(5):

print(f"{i:4d} {learn_data[i]} {learn_labels[i]:3}")

print("The first samples of our test set:")

print(f"{'index':7s}{'data':20s}{'label':3s}")

for i in range(5):

print(f"{i:4d} {learn_data[i]} {learn_labels[i]:3}")
```

LAKSHMI NARAIN COLLEGE OF TECHNOLOGY & SCIENCE, BHOPAL

CS-601

MACHINE LEARNING

EXPERIMENT -7

Aim: Design convolution neural network for image classification in presence of limited data.

THEORY

Convolutional Neural Networks come under the sub domain of Machine Learning which is Deep Learning. Algorithms under Deep Learning process information the same way the human brain does, but obviously on a very small scale, since our brain is too complex (our brain has around 86 billion neurons).
Image classification involves the extraction of features from the image to observe some patterns in the dataset. Using an ANN for the purpose of image classification would end up being very costly in terms of computation since the trainable parameters become extremely large.

ALGORITHM

Step 1: Choose a Dataset

Step 2: Prepare Dataset for Training

Step 3: Create Training Data

Step 4: Shuffle the Dataset

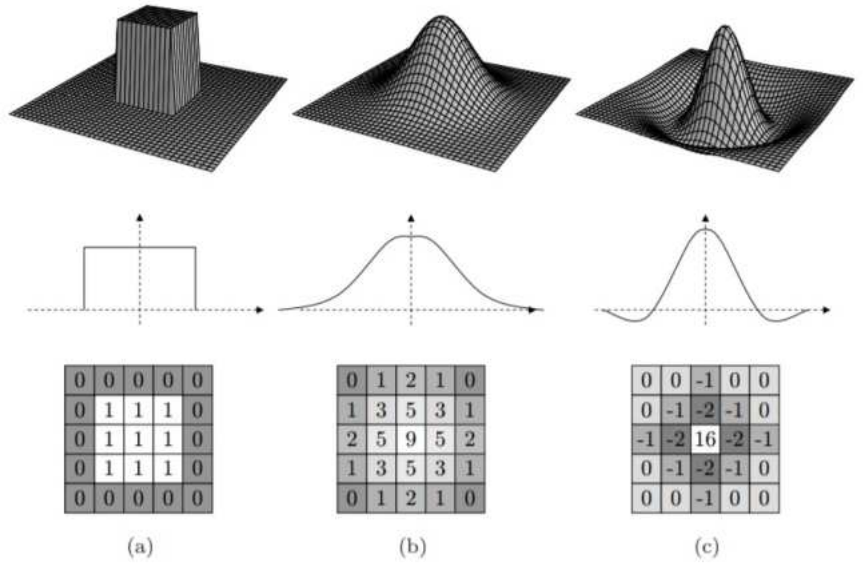
Step 5: Assigning Labels and Features

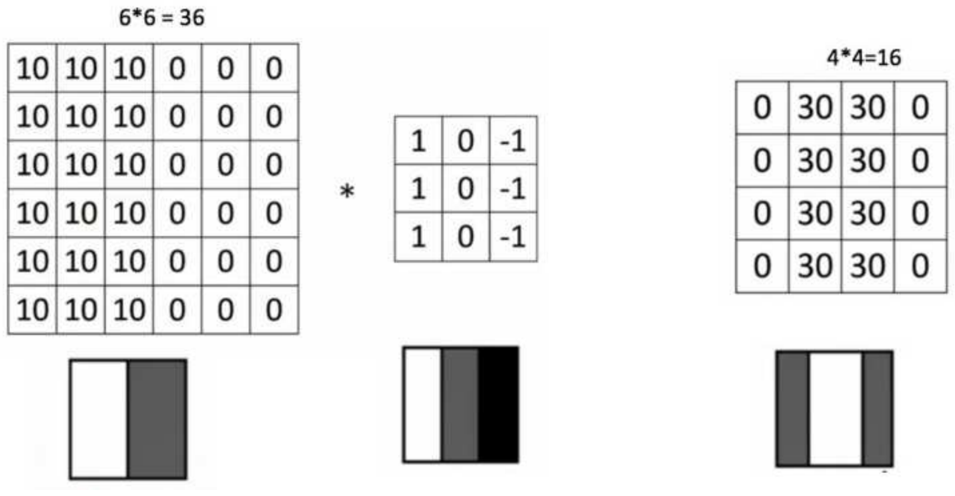
Step 6: Normalizing X and converting labels to categorical data

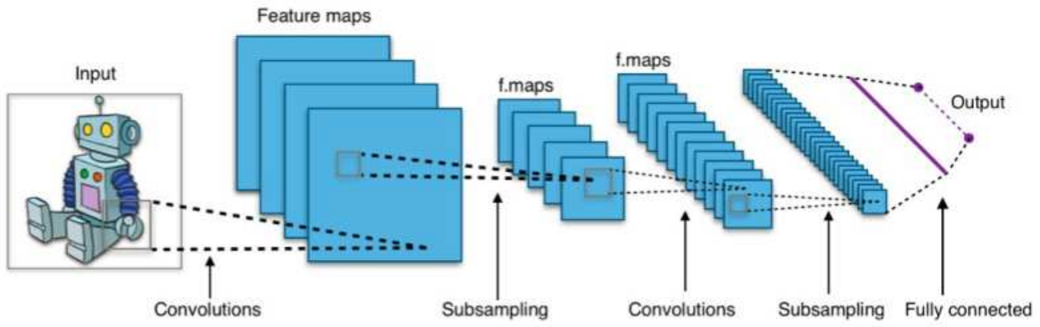
Step 7: Split X and Y for use in CNN

Step 8: Define, compile and train the CNN Model

Step 9: Accuracy and Score of model


0 0 0 0 0
0 1 1 1 0
0 1 1 1 0
0 1 1 1 0
0 0 0 0 0
0 1 2 1 0
1 3 5 3 1
2 5 9 5 2
1 3 5 3 1
0 1 2 1 0
0 0 -1 0 0
0 -1 -2 -1 0
-1 -2 16 -2 -1
0 -1 -2 -1 0
0 0 -1 0 0
(a)
(b)
(c)


6*6 = 36
10 10 10 0 0 0
10 10 10 0 0 0
10 10 10 0 0 0
10 10 10 0 0 0
10 10 10 0 0 0
10 10 10 0 0 0
*
1 0 -1
1 0 -1
1 0 -1
4*4=16
0 30 30 0
0 30 30 0
0 30 30 0
0 30 30 0



PROGRAM

Here are all the libraries that we would require and the code for importing them.

```
from keras.models import Sequential

import tensorflow as tf

import tensorflow_datasets as tfds

tf.enable_eager_execution()

from keras.layers.core import Dense, Activation, Dropout, Flatten

from keras.layers.convolutional import Convolution2D, MaxPooling2D

from keras.optimizers import SGD, RMSprop, adam

from keras.utils import np_utils

from sklearn.tree import DecisionTreeClassifier # Import Decision Tree Classifier

from sklearn import metricsfrom sklearn.utils import shuffle

from sklearn.model_selection import train_test_splitimport matplotlib.image as mpimg

import matplotlib.pyplot as plt

import numpy as np

import os

import cv2

import randomfrom numpy import *
```

```
from PIL import Image

mport theano
```

Resizing images into 200 X 200

```
path_test = "/content/drive/My Drive/semester 5 - ai ml/datasetHomeAssign/TRAIN"
CATEGORIES = ["EOSINOPHIL", "LYMPHOCYTE", "MONOCYTE", "NEUTROPHIL"]
print(img_array.shape)IMG_SIZE =200
new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))
```

Training is an array that will contain image pixel values and the index at which the image in the CATEGORIES list.

```
training = []def createTrainingData():
 for category in CATEGORIES:
 path = os.path.join(path_test, category)
 class_num = CATEGORIES.index(category)
  for img in os.listdir(path):
  img_array = cv2.imread(os.path.join(path,img))
  new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))

 training.append([new_array, class_num])createTrainingData()

random.shuffle(training)
```

This shape of both the lists will be used in Classification using the NEURAL NETWORKS.

```
X =[]
y =[]for features, label in training:
  X.append(features)
  y.append(label)

X = np.array(X).reshape(-1, IMG_SIZE, IMG_SIZE, 3)

X = X.astype('float32')
X /= 255
```

```
from keras.utils import np_utils
Y = np_utils.to_categorical(y, 4)
print(Y[100])



score = model.evaluate(X_test, y_test, verbose = 0 )
print("Test Score: ", score[0])
print("Test accuracy: ", score[1])
```

EXPERIMENT -8

Aim: **Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.**

THEORY

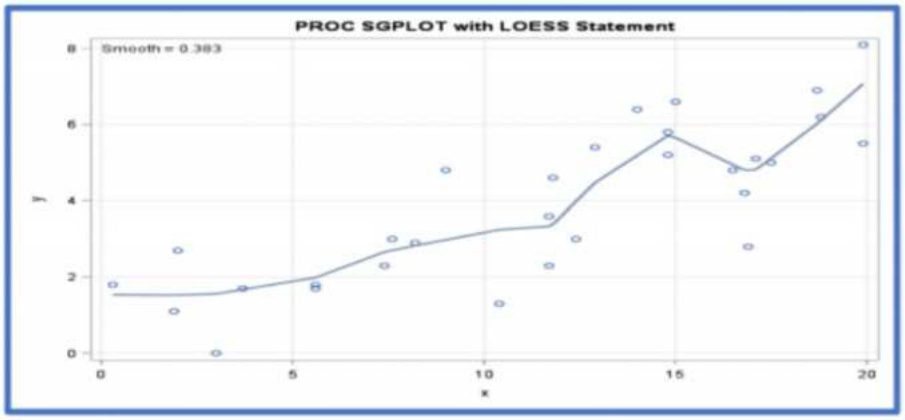
Locally Weighted Regression Algorithm

Regression:

- Regression is a technique from statistics that is used to predict values of a desiredtarget quantity when the target quantity is continuous.
- In regression, we seek to identify (or estimate) a continuous variable y associated witha given input vector x.
- y is called the dependent variable.
- x is called the independent variable.

Loess/Lowess Regression:

Loess regression is a nonparametric technique that uses local weighted regression to fit a smooth curve through points in a scatter plot.



Lowess Algorithm:

- Locally weighted regression is a very powerful nonparametric model used in statistical learning.
- Given a dataset X, y, we attempt to find a model parameter β(x) that minimizes residual sum of weighted squared errors.
- The weights are given by a kernel function (k or w) which can be chosen arbitrarily.

Algorithm

1. Read the Given data Sample to X and the curve (linear or non linear) to Y
2. Set the value for Smoothening parameter or free parameter say τ
3. Set the bias /Point of interest set x0 which is a subset of X
4. Determine the weight matrix using:

$$w(x, x_o) = e^{-\frac{(x-x_o)^2}{2\tau^2}}$$

5. Determine the value of model term parameter β using :

$$\hat{\beta}(x_o) = (X^T W X)^{-1} X^T W y$$

6. Prediction = x0*β:

Program

```
import numpy as np

from bokeh.plotting import figure, show, output_notebook

from bokeh.layouts import gridplot

from bokeh.io import push_notebook

def local_regression(x0, X, Y, tau):# add bias term

x0 = np.r_[1, x0] # Add one to avoid the loss in

information

X = np.c_[np.ones(len(X)), X]

# fit model: normal equations with kernel

xw = X.T * radial_kernel(x0, X, tau) # XTranspose * W

beta = np.linalg.pinv(xw @ X) @ xw @ Y #@ Matrix

Multiplication or Dot Product

# predict value

return x0 @ beta # @ Matrix Multiplication or Dot Product

for prediction

def radial_kernel(x0, X, tau):

return np.exp(np.sum((X - x0) ** 2, axis=1) / (-2 * tau *

tau))

# Weight or Radial Kernal Bias Function

n = 1000

# generate dataset
```

```
X = np.linspace(-3, 3, num=n)
print("The Data Set ( 10 Samples) X :\n",X[1:10])
Y = np.log(np.abs(X ** 2 - 1) + .5)
print("The Fitting Curve Data Set (10 Samples) Y
:\n",Y[1:10])
# jitter X
X += np.random.normal(scale=.1, size=n)
print("Normalised (10 Samples) X :\n",X[1:10])
domain = np.linspace(-3, 3, num=300)
print(" Xo Domain Space(10 Samples) :\n",domain[1:10])
def plot_lwr(tau):
# prediction through regression
prediction = [local_regression(x0, X, Y, tau) for x0 in
domain]
plot = figure(plot_width=400, plot_height=400)
plot.title.text='tau=%g' % tau
plot.scatter(X, Y, alpha=.3)
plot.line(domain, prediction, line_width=2, color='red')
return plot
show(gridplot([
[plot_lwr(10.), plot_lwr(1.)],
[plot_lwr(0.1), plot_lwr(0.01)]]))
```

CS-601

MACHINE LEARNING

EXPERIMENT -9

Aim: Classifying data using Support Vector Machines (SVMs) in python. THEORY

In machine learning, support vector machines (SVMs, also support vector networks) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane.

An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. In addition to performing linear classification, SVMs can efficiently perform a non-linear classification, implicitly mapping their inputs into high-dimensional feature spaces.

ALGORITHM

1. Import required libraries
2. Load the image and convert it to a dataframe.
3. separate input features and targets.
4. Split train and test value.
5. Build and train the model
6. Model evaluation.
7. Prediction

PROGRAM

```
import pandas as pd

import os

from skimage.transform import resize

from skimage.io import imread
```

```
import numpy as np

import matplotlib.pyplot as plt

from sklearn import svm

from sklearn.model_selection import GridSearchCV

Categories=['cats','dogs']

flat_data_arr=[] #input array

target_arr=[] #output array

datadir='IMAGES/'

#path which contains all the categories of images

for i in Categories:


    print(f'loading... category : {i}')

    path=os.path.join(datadir,i)

    for img in os.listdir(path):

        img_array=imread(os.path.join(path,img))

        img_resized=resize(img_array,(150,150,3))

        flat_data_arr.append(img_resized.flatten())

        target_arr.append(Categories.index(i))

    print(f'loaded category:{i} successfully')

flat_data=np.array(flat_data_arr)

target=np.array(target_arr)
#dataframe

df=pd.DataFrame(flat_data)

df['Target']=target
```

```
df.shape
#input data

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

#output data

y=df.iloc[:,-1]
# Splitting the data into training and testing sets

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.20,


        random_state=77 , stratify=y)

# Defining the parameters grid for GridSearchCV

param_grid={'C':[0.1,1,10,100],

                'gamma':[0.0001,0.001,0.1,1],

                'kernel':['rbf','poly']}

# Creating a support vector classifier

svc=svm.SVC(probability=True)

# Creating a model using GridSearchCV with the parameters grid

    model=GridSearchCV(svc,param_grid)
# Training the model using the training data

model.fit(x_train,y_train)

# Testing the model using the testing data

y_pred = model.predict(x_test)

# Calculating the accuracy of the model

accuracy = accuracy_score(y_pred, y_test)

# Print the accuracy of the model

print(f"The model is {accuracy*100}% accurate")
print(classification_report(y_test, y_pred, target_names=['cat', 'dog']))

path='dataset/test_set/dogs/dog.4001.jpg'

img=imread(path)

plt.imshow(img)

plt.show()

img_resize=resize(img,(150,150,3))

l=[img_resize.flatten()]

probability=model.predict_proba(l)

for ind,val in enumerate(Categories):

    print(f'{val} = {probability[0][ind]*100}%')

print("The predicted image is : "+Categories[model.predict(l)[0]])
```

EXPERIMENT -10

Aim: Write a Program to construct Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard heart disease data set. You can use standard heart disease data set. Use python ML Library Classes/API**.**

THEORY

A Bayesian network, Bayes network, belief network, decision network, Bayes(ian) model or probabilistic directed acyclic graphical model is a probabilistic graphical model (a type of statistical model) that represents a set of variables and their conditional dependencies via a directed acyclic graph (DAG). Bayesian networks are ideal for taking an event that occurred and predicting the likelihood that any one of several possible known causes was the contributing factor. For example, a Bayesian network could represent the probabilistic relationships between diseases and symptoms. Given symptoms, the network can be used to compute the probabilities of the presence of various diseases. Efficient algorithms can perform inference and learning in

Bayesian networks. Bayesian networks that model sequences of variables (e.g. speech signals or protein sequences) are called dynamic Bayesian networks. Generalizations of Bayesian networks that can represent and solve decision problems under uncertainty are called influence diagrams.

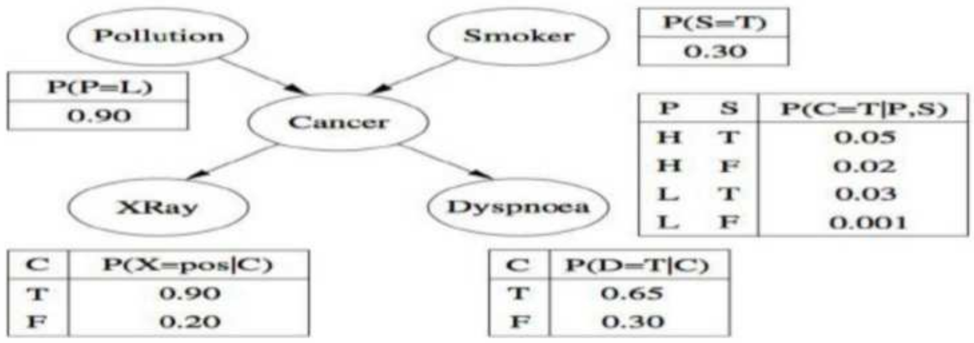


Fig. A BN for the lung cancer

Attribute Information:

-- Only 14 used

-- 1. #3 (age)

-- 2. #4 (sex)

-- 3. #9 (cp)

-- 4. #10 (trestbps)

-- 5. #12 (chol)

-- 6. #16 (fbs)

-- 7. #19 (restecg)

-- 8. #32 (thalach)

-- 9. #38 (exang)

-- 10. #40 (oldpeak)

-- 11. #41 (slope)

-- 12. #44 (ca)

-- 13. #51 (thal)

-- 14. #58 (num)

SOURCE CODE

```
import numpy as np

from urllib.request import urlopen

import urllib

import pandas as pd

from pgmpy.inference import VariableElimination

from pgmpy.models import BayesianModel

from pgmpy.estimators import MaximumLikelihoodEstimator, BayesianEstimator

names = [‘age’, ‘sex’, ‘cp’, ‘trestbps’, ‘chol’, ‘fbs’, ‘restecg’, ‘thalach’, ‘exang’, ‘oldpeak’,
‘slope’, ‘ca’,

‘thal’, ‘heartdisease’]

heartDisease = pd.read_csv(‘heart.csv’, names = names)

heartDisease = heartDisease.replace(‘?’, np.nan)

model = BayesianModel([(‘age’, ‘trestbps’), (‘age’, ‘fbs’), (‘sex’, ‘trestbps’), (‘exang’,

‘trestbps’),(‘trestbps’,’heartdisease’),(‘fbs’,’heartdisease’),(‘heartdisease’,’restecg’),
(‘heartdisease’,’thalach’),

(‘heartdisease’,’chol’)])

model.fit(heartDisease, estimator=MaximumLikelihoodEstimator)

from pgmpy.inference import VariableElimination
```

```
HeartDisease_infer = VariableElimination(model)

q = HeartDisease_infer.query(variables=[‘heartdisease’], evidence={‘age’: 37, ‘sex’ :0})

print(q[‘heartdisease’])
```

CS-601

MACHINE LEARNING

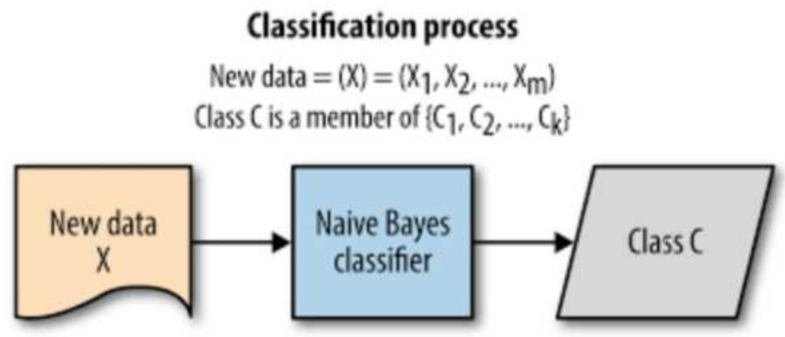
EXPERIMENT -11

1. **Aim: Assuming a set of documents that need to be classified, use the naïve Bayesian Classifier model to perform this task. Built-in Java classes/API can be used to write the program. Calculate the accuracy, precision, and recall for your data set.**

THEORY

$$\underset{\text{Posterior Probability}}{P(c \mid x)} = \frac{\overset{\text{Likelihood}}{P(x \mid c)}\,\overset{\text{Class Prior Probability}}{P(c)}}{\underset{\text{Predictor Prior Probability}}{P(x)}}$$

$$P(c \mid X) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$


Classification process
New data = (X) = (X1, X2, ..., Xm)
Class C is a member of {C1, C2, ..., Ck}
New data X
Naive Bayes classifier
Class C

The dataset is divided into two parts, namely, feature matrix and the response vector.

- Feature matrix contains all the vectors (rows) of dataset in which each vector consists of the value of dependent features. In above dataset, features are ‘Outlook’, ‘Temperature’, ‘Humidity’ and ‘Windy’.
- Response vector contains the value of class variable (prediction or output) for each row of feature matrix. In above dataset, the class variable name is ‘Play golf’.

Types of Naive Bayes Algorithm

Gaussian Naive Bayes

When attribute values are continuous, an assumption is made that the values associated with each class are distributed according to Gaussian i.e., Normal Distribution. If in our data, an attribute say “x” contains continuous data. We first segment the data by the class and then compute mean μy & Variance σ 2y of each class.

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} exp\left(-\frac{(x_i-\mu_y)^2}{2\sigma_y^2}\right)$$

MultiNomial Naive Bayes MultiNomial Naive Bayes is preferred to use on data that is multinomially distributed. It is one of the standard classic algorithms. Which is used in text categorization (classification). Each event in text classification represents the occurrence of a word in a document.

$$p(\mathbf{x} \mid C_k) = \frac{(\sum_i x_i)!}{\prod_i x_i!} \prod_i {p_{ki}}^{x_i}$$

$$\hat{P}(x_i \mid \omega_j) = \frac{\sum tf(x_i, d \in \omega_j) + \alpha}{\sum N_{d\in\omega j} + \alpha \cdot V}$$

Bernoulli Naive Bayes

Bernoulli Naive Bayes is used on the data that is distributed according to multivariate Bernoulli distributions .i.e., multiple features can be there, but each one is assumed to be a binary-valued (Bernoulli, Boolean) variable. So, it requires features to be binary valued.

$$P(C \mid X_1, \dots, X_n) = \prod_{i=1}^{n} [X_i P(t_i \mid C) + (1 - X_i)(1 - P(t_i \mid C)]$$

$X_i = 1$ if t_i is present
0 otherwise

Absence of terms

ALGORITHM

1. Given training data set le) which consists of documents belonging to diffcrcnt class say class A and B.
2. Calculate the prior probability of class A=number of objects of class A / total number of objects Calculate the prior probability of class B=number of objects: of class B / total number of objects.
3. Find ni, the total number of word frequency of each class.. na= the total number of word frequency of class A. nb= the total number of word frequency of class B.

4. Find conditional probability of keyword occurrence given ai class.
P(wordl f class A) = wordcount / ni(A)
P(wordl / class B) =wordcount / ni(B)
P(word2 / class A) = wordcount / ni(A)
P(word2 / class B) =wordcount / ni(B)
P(wordn / class B) =wordcount / ni(B)
5. Avoid zero frequency problems by applying uniform distribution.
6. Classify a new document C based on the probability P(C /W). a) Find P(A / W) a P(A) P(wordl/ class A) a's P(word2/' class A) P(wordn / class A). b) Find P(B /W) = P(13) P(word I / class B) P(word2/1 class B) P(wordn class B).
7. Assign document to class that has higher probability.

SOURCE CODE

```
import pandas as pd

msg=pd.read_csv('naivetext1.csv',names=['message','label'])

print('The dimensions of the dataset',msg.shape)

msg['labelnum']=msg.label.map({'pos':1,'neg':0})

X=msg.message

y=msg.labelnum

print(X)

print(y)

from sklearn.model_selection import train_test_split

xtrain,xtest,ytrain,ytest=train_test_split(X,y)

print(xtest.shape)

print(xtrain.shape)

print(ytest.shape)

print(ytrain.shape)

from sklearn.feature_extraction.text import CountVectorizer
```

```
count_vect = CountVectorizer()

xtrain_dtm = count_vect.fit_transform(xtrain)

xtest_dtm=count_vect.transform(xtest)

from sklearn.naive_bayes import MultinomialNB

clf = MultinomialNB().fit(xtrain_dtm,ytrain)

predicted = clf.predict(xtest_dtm)

from sklearn import metrics

print('Accuracy metrics')

print('Accuracy of the classifer is',metrics.accuracy_score(ytest,predicted))

print('Confusion matrix')

print(metrics.confusion_matrix(ytest,predicted))

print('Recall and Precison ')

print(metrics.recall_score(ytest,predicted))

print(metrics.precision_score(ytest,predicted))
```

OUTPUT

```
The dimensions of the dataset (18, 2)
0 I love this sandwich
1 This is an amazing place
2 I feel very good about these beers
3 This is my best work
4 What an awesome view
5 I do not like this restaurant
6 I am tired of this stuff
7 I can't deal with this
8 He is my sworn enemy
9 My boss is horrible
10 This is an awesome place
11 I do not like the taste of this juice
12 I love to dance
```

```
13 I am sick and tired of this place
14 What a great holiday
15 That is a bad locality to stay
16 We will have good fun tomorrow
17 I went to my enemy’s house today
Name: message, dtype: object
0 1
1 1
2 1
3 1
4 1
5 0
6 0
7 0
8 0
9 0
10 1
11 0
12 1
13 0
14 1
15 0
16 1
17 0
Name: labelnum, dtype: int64
(5,)
(13,)
(5,)
(13,)
Accuracy metrics
Accuracy of the classifer is 0.8
Confusion matrix
[[3 1] [0 1]]
Recall and Precison 1.0 0.5
```

LAKSHMI NARAIN COLLEGE OF TECHNOLOGY & SCIENCE, BHOPAL

CS-601

MACHINE LEARNING