EXP NO: 07	ARTIFICIAL NEURAL NETWORKS
DATE: 26/04/2022	

AIM:

To apply Artificial Neural networks algorithm to predict the output given a set of inputs and outputs as training example.

ALGORITHM:

Step 1: Start

Step 2: Take the inputs and multiply by random weights.

$$Y = W_iI_i = W_1I_1+W_2I_2+W_3I_3$$

Step 3: Pass the result through a sigmoid formula to calculate the neuron's output. The Sigmoid function is used to normalize the result between 0 and 1

$$1/1 + e^{-y}$$

Step 4: Calculate the error i.e the difference between the actual output and the expected output.

Step 5: Depending on the error, adjust the weights by multiplying the error with the input and again with the gradient of the Sigmoid curve.

Step 6: Repeat the process for few iterations.

Step 7: Stop.

PROGRAM:

```
from joblib.numpy_pickle_utils import xrange
from numpy import *
class NeuralNet(object):
    def __init__(self):
        random.seed(1)
        self.synaptic_weights = 2 * random.random((3, 1)) - 1
    def __sigmoid(self, x):
        return 1 / (1 + exp(-x))
    def __sigmoid_derivative(self, x):
        return x * (1 - x)
```

```
    def train(self, inputs, outputs, training_iterations):
        for iteration in xrange(training_iterations):
            output = self.learn(inputs)
            error = outputs - output
            factor = dot(inputs.T, error * self.__sigmoid_derivative(output))
            self.synaptic_weights += factor
    def learn(self, inputs):
        return self.__sigmoid(dot(inputs, self.synaptic_weights))
if __name__ == "__main__":
    neural_network = NeuralNet()
    inputs = array([[0, 1, 1], [1, 0, 0], [1, 0, 1]])
    outputs = array([[1, 0, 1]]).T
    neural_network.train(inputs, outputs, 10000)
    print(neural_network.learn(array([1, 0, 1])))
```

OUTPUT:

```
[0.9897704]
```

RESULT:

Thus the program for Artificial neural network is implemented successfully.

EX NO: 8	KNN - Algorithm
DATE: 09-05-2022	

AIM:

To Apply KNN algorithm to find the nearest of the boundary line.

ALGORITHM:

Step 1: start

Step 2: Store the training samples in an array of data points arr[]. This means each element of this array represents a tuple (x, y).

Step 3: for i=0 to m:
Calculate Euclidean distance d(arr[i], p).

Step 4: Make set S of K smallest distances obtained. Each of these distances correspond to an already classified data point.

Step 5: Return the majority label among S.

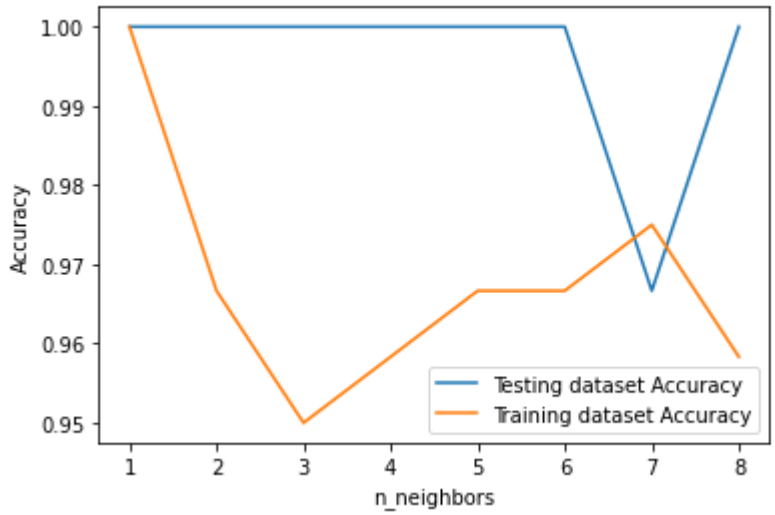
Step 6: stop

PROGRAM:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
import numpy as np
import matplotlib.pyplot as plt

irisData = load_iris()

# Create feature and target arrays
X = irisData.data
y = irisData.target

# Split into training and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0
.2, random_state=42)

neighbors = np.arange(1, 9)
train_accuracy = np.empty(len(neighbors))
test_accuracy = np.empty(len(neighbors))

# Loop over K values
for i, k in enumerate(neighbors):
```

```
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)

    # Compute training and test data accuracy
    train_accuracy[i] = knn.score(X_train, y_train)
    test_accuracy[i] = knn.score(X_test, y_test)

# Generate plot
plt.plot(neighbors, test_accuracy, label = 'Testing dataset Accuracy')
plt.plot(neighbors, train_accuracy, label = 'Training dataset Accuracy)
plt.legend()
plt.xlabel('n_neighbors')
plt.ylabel('Accuracy')
plt.show()
```

OUTPUT:



RESULT:

Thus, the program to find the nearest of the boundary line using KNN algorithm is successfully implemented.

EXP NO: 09	BAYESIAN CLASSIFIER
DATE:	

AIM:

To perform Bayesian classifier algorithm and print the probabilities.

ALGORITHM:

Step 1: Start

Step 2: Split the dataset by class values.

Step 3: Calculate the standard deviation for a list of numbers.

Step 4: Calculate the probability for each class value.

Step 5: Display the probability.

Step 6: Stop.

PROGRAM:

```
from math import sqrt
from math import pi
from math import exp
def separate_by_class(dataset):
  separated = dict()
  for i in range(len(dataset)):
    vector = dataset[i]
    class_value = vector[-1]
    if (class_value not in separated):
      separated[class_value] = list()
    separated[class_value].append(vector)
  return separated
def mean(numbers):
  return sum(numbers)/float(len(numbers))
def stdev(numbers):
  avg = mean(numbers)
  variance = sum([(x-avg)**2 for x in numbers]) / float(len(numbers)-1)
  return sqrt(variance)
def summarize_dataset(dataset):
  summaries = [(mean(column), stdev(column), len(column)) for column in zip(*dataset)]
  del(summaries[-1])
  return summaries
```

```
def summarize_by_class(dataset):
  separated = separate_by_class(dataset)
  summaries = dict()
  for class_value, rows in separated.items():
    summaries[class_value] = summarize_dataset(rows)
  return summaries
def calculate_probability(x, mean, stdev):
  exponent = exp(-((x-mean)**2 / (2 * stdev**2 )))
  return (1 / (sqrt(2 * pi) * stdev)) * exponent
```

```
def calculate_class_probabilities(summaries, row):
  total_rows = sum([summaries[label][0][2] for label in summaries])
  probabilities = dict()
  for class_value, class_summaries in summaries.items():
    probabilities[class_value] = summaries[class_value][0][2]/float(total_rows)
    for i in range(len(class_summaries)):
      mean, stdev, _ = class_summaries[i]
      probabilities[class_value] *= calculate_probability(row[i], mean, stdev)
  return probabilities
dataset = [[3.393533211,2.331273381,0],
  [3.110073483,1.781539638,0],
  [1.343808831,3.368360954,0],
  [3.582294042,4.67917911,0],
  [2.280362439,2.866990263,0],
  [7.423436942,4.696522875,1],
  [5.745051997,3.533989803,1],
  [9.172168622,2.511101045,1],
  [7.792783481,3.424088941,1],
  [7.939820817,0.791637231,1]]
summaries = summarize_by_class(dataset)
probabilities = calculate_class_probabilities(summaries, dataset[0])
print(probabilities)
```

OUTPUT:

```
{0: 0.05032427673372076, 1: 0.00011557718379945765}
```

RESULT:

Hence Bayesian classifier algorithm is implemented and the probabilities have been displayed.

EX NO: 10	k-Means Algorithm
DATE: 09-05-2022	

AIM:

To Apply K-Means algorithm to find the partition the dataset into K pre-defined distinct non-overlapping subgroups (clusters) where each data point belongs to only one group

ALGORITHM:

Step 1: start

Step 2: Specify number of clusters *K*.

Step 3: Initialize centroids by first shuffling the dataset and then randomly selecting *K* data points for the centroids without replacement.

Step 4: Keep iterating until there is no change to the centroids. i.e assignment of data points to clusters isn't changing.

Step 5: Compute the sum of the squared distance between data points and all centroids.

Step 6: Assign each data point to the closest cluster (centroid).

Step 7: Compute the centroids for the clusters by taking the average of the all data points that belong to each cluster.

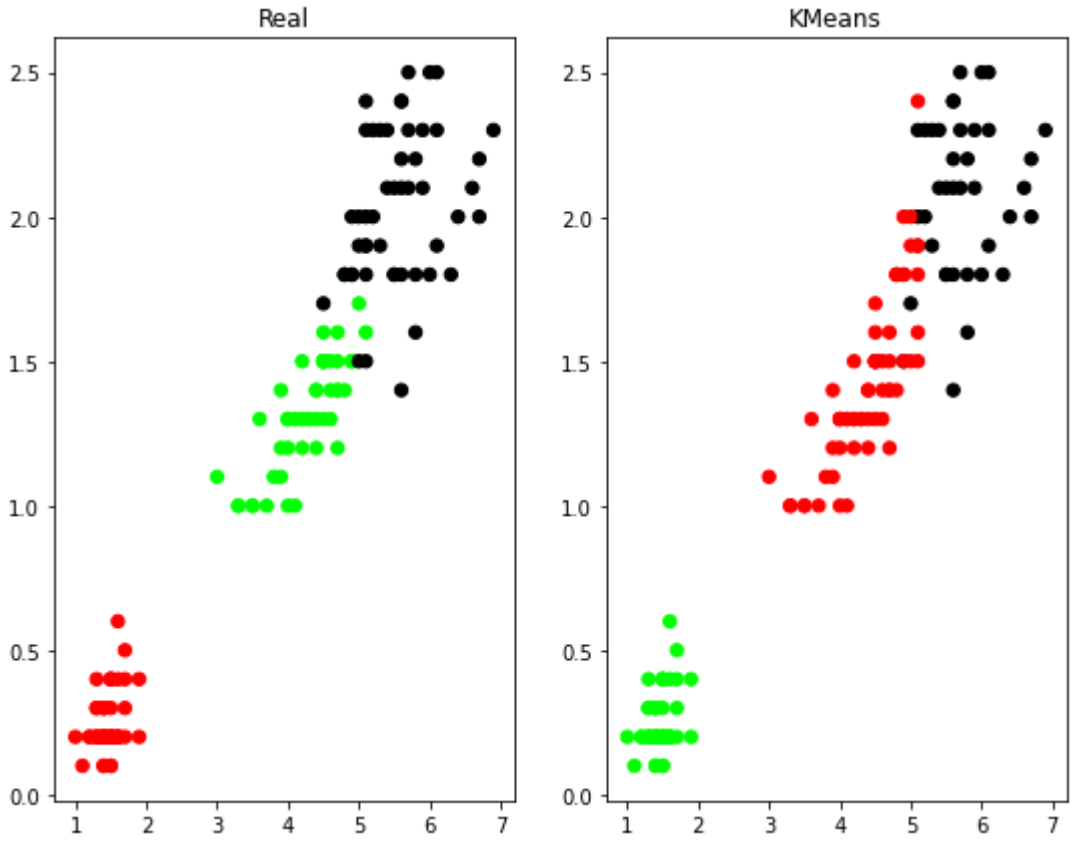
Step 8: stop

PROGRAM:

```
from sklearn.cluster import KMeans
from sklearn import preprocessing
from sklearn.mixture import GaussianMixture
from sklearn.datasets import load_iris
import sklearn.metrics as sm
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
dataset=load_iris()
X=pd.DataFrame(dataset.data)
X.columns=['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width'
]
y=pd.DataFrame(dataset.target)
y.columns=['Targets']
```

```
plt.figure(figsize=(14,7))
colormap=np.array(['red','lime','black'])
plt.subplot(1,3,1)
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y.Targets],s=40)
plt.title('Real')
plt.subplot(1,3,2)
model=KMeans(n_clusters=3)
model.fit(X)
predY=np.choose(model.labels_,[0,1,2]).astype(np.int64)
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[predY],s=40)
plt.title('KMeans')
```

OUTPUT:



RESULT:

Thus, successfully partition the dataset into K pre-defined distinct non-overlapping subgroups (clusters) where each data point belongs to only one group is achived.