EX.NO: 1	VECTORS AND MATRIX REPRESENTATION AND MANIPULATION
DATE:	

AIM:

To write a python program to represent & manipulate vectors and matrices

ALGORITHM:

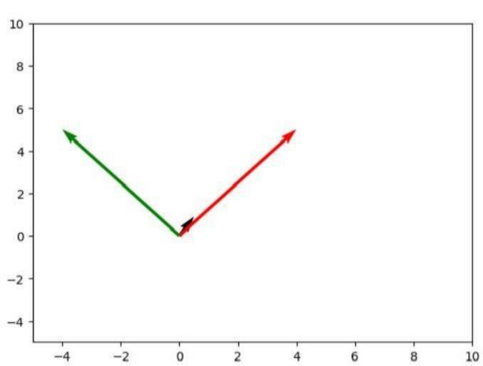
Step 1: Create a python notebook
Step 2: Initialize a vector and perform addition, subtraction and multiplication
Step 3: Plot the vector and visualize the direction of movement
Step 4: Initialize a matrix and perform addition, subtraction and multiplication
Step 5: Calculate the time taken to perform operation using NumPy and normal matrix representation

Vector:

A vector, in programming, is a type of array that is one dimensional. Vectors are a logical element in programming languages that are used for storing a sequence of data elements of the same basic type. Members of a vector are called components.' The major difference between and array and a vector is that the container size of a vector can be easily increased and decreased to complement different data storage types.

PROGRAM:

```
import numpy as np
import matplotlib.pyplot as plt
plt.quiver(0,0,3,4)
plt.quiver(0,0,4,5, scale_units='xy',angles='xy',scale=1, color='r')
plt.quiver(0,0,-4,5, scale_units='xy',angles='xy',scale=1, color='g')
plt.xlim(-5,10)
plt.ylim(-5,10)
plt.show()
```



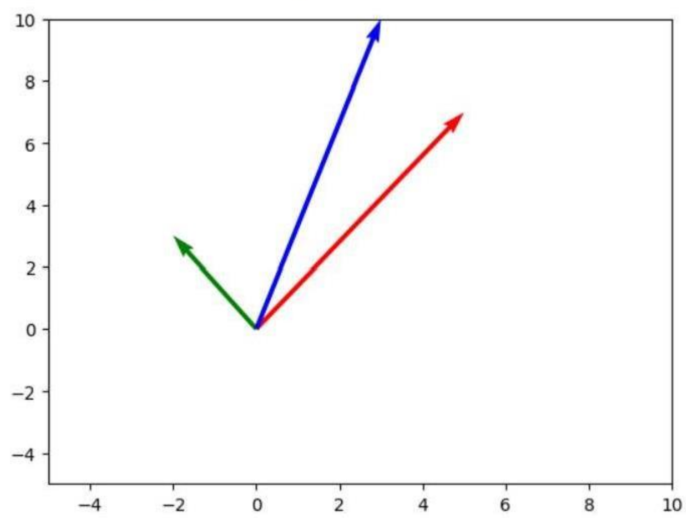
ADDITION, SUBTRACTION AND MULTIPLICATION OF VECTORS

```
def plot_vector(vecs):
    colors = ['r', 'g', 'b', 'y']
    i = 0
for vec in vecs:
        plt.quiver(vec[0], vec[1], vec[2], vec[3], scale_units="xy", angles="xy", scale=1, color=colors[i
%len(colors)])
        i += 1
    plt.xlim(-5,10)
    plt.ylim(-5,10)
    plt.show()
```

 # Vector Addition

```
vecs = [np.asarray([0, 0, 5, 7]), np.asarray([0, 0, -2, 3])]
result_add = vecs[0] + vecs[1]
print("Addition Result:", result_add)
plot_vector([vecs[0], vecs[1], result_add])
```

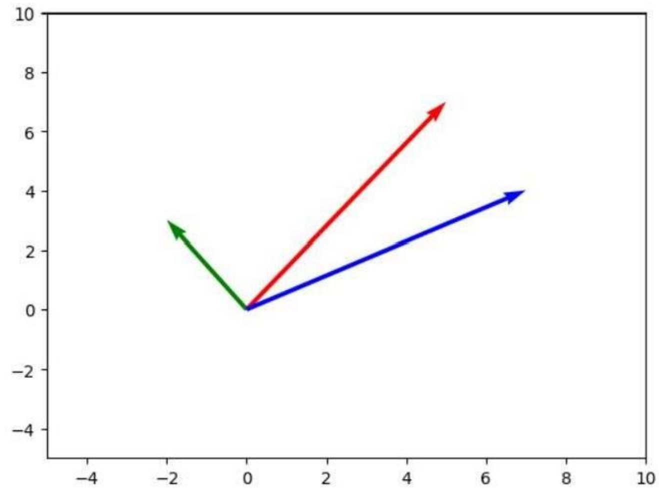
```
Addition Result: [ 0  0  3 10]
```



 # Vector Subtraction

```
result_sub = vecs[0] - vecs[1]
print("Subtraction Result:", result_sub)
plot_vector([vecs[0], vecs[1], result_sub])
```

```
Subtraction Result: [0 0 7 4]
```

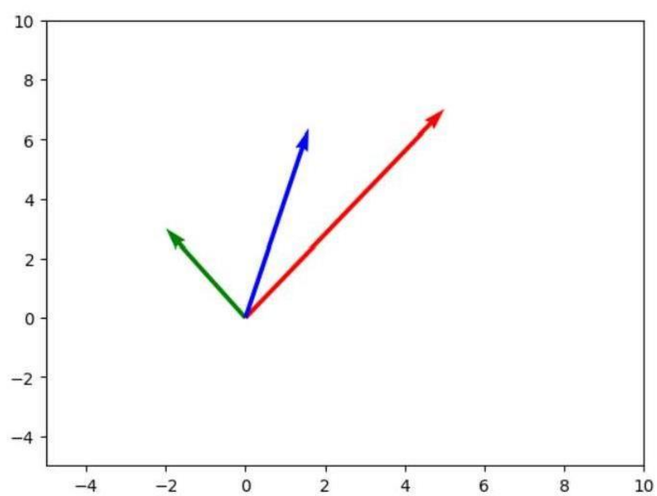


Vector Multiplication (Dot Product)

$$a_b = |\vec{a}|\cos(\theta) = |\vec{a}|\frac{\vec{a}\cdot\vec{b}}{|\vec{a}||\vec{b}|} = \frac{\vec{a}\cdot\vec{b}}{|\vec{b}|}$$

```
vec_a = np.asarray([3, 6])
vec_b = np.asarray([2, 8])
c = np.dot(vec_a, vec_b) / np.linalg.norm(vec_b)
vec_c = (c / np.linalg.norm(vec_b)) * vec_b
print("Multiplication Result (Projection):", vec_c)
plot_vector([vecs[0], vecs[1], [0, 0, vec_c[0], vec_c[1]]])
```

```
Multiplication Result (Projection): [1.58823529 6.35294118]
```



Matrix Addition

```
a = np.array([1, 2, 3], dtype=np.int64)
b = np.array([4, 5, 6])
print("\nMatrix A:\n", a)
```

```
print("Matrix B:\n", b)
print("Regular Matrix Addition A + B:\n", a + b)

print("Addition Using Broadcasting A + 10:\n", a + 10)
```

 # 2D Matrix Addition

```
c = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
d = np.array([[9, 8, 7], [6, 5, 4], [3, 2, 1]])
e = np.array([1, 2, 3])
print("\nMatrix C:\n", c)
print("Matrix D:\n", d)
print("Matrix E:\n", e)
print("Regular Matrix Addition C + D:\n", c + d)
print("Addition Using Broadcasting C + E:\n", c + e)
```

 # Describing a Matrix

```
print("\nMatrix C Size:", c.size)
print("Matrix C Shape:", c.shape)
print("Matrix C Data:", c.data)
print("Matrix C Data Type:", c.dtype)
print("Length of Matrix C (number of rows):", len(c))
```

 # Vectorized Matrix Multiplication

```
print("\nVectorized Matrix Multiplication C.dot(D):\n", c.dot(d))
```

OUTPUT:

```
Matrix A:
 [1 2 3]
Matrix B:
 [4 5 6]
Regular Matrix Addition A + B:
 [5 7 9]
Addition Using Broadcasting A + 10:
 [11 12 13]

Matrix C:
 [[1 2 3]
 [4 5 6]
 [7 8 9]]
Matrix D:
 [[9 8 7]
 [6 5 4]
 [3 2 1]]
Matrix E:
 [1 2 3]
Regular Matrix Addition C + D:
 [[10 10 10]
 [10 10 10]
 [10 10 10]]
Addition Using Broadcasting C + E:
 [[ 2  4  6]
 [ 5  7  9]
 [ 8 10 12]]
```

```
Matrix C Size: 9
Matrix C Shape: (3, 3)
Matrix C Data: <memory at 0x0000020D58089D80>
Matrix C Data Type: int32
Length of Matrix C (number of rows): 3

Vectorized Matrix Multiplication C.dot(D):
 [[ 30  24  18]
 [ 84  69  54]
 [138 114  90]]
```

RESULT:

Thus, the python program to represent & manipulate vectors and matrices was successfully executed.

EX.NO:2	DEEP LEARNING FRAMEWORKS
DATE:	

AIM:

To study and work with different python frameworks and libraries used for implementing Deep Learning libraries.

DESCRIPTION:

TensorFlow

The most popular library for Machine Learning, TensorFlow is the best Python application development tool for advanced solutions. It simplifies building Machine Learning models for beginners and professionals. It has built-in modules for visualization, inspection and model serialization. TensorFlow is backed by the Google brain team, ensuring regular updates. It is useful for natural language processing, deep neural networks, image and speech recognition, and other functions for Deep Learning.

Keras

One of the fastest-growing Deep Learning framework packages, Keras enables using high-level network AP, along with a clean user interface. It enables engineers to combine standalone modules with low restrictions. Keras is highly used in building neural layers, solutions with activation and cost functions, batch normalization, and more. It works on top of TensorFlow, which extends its functionality for ML-based projects.

PyTorch

The primary aim of PyTorch is to speed up the entire process of Python app development for Machine Learning solutions. It has a C++ frontend along with the Python interface. PyTorch enables quick production deployment, providing companies with rapid solutions.PyTorch offers training, building, and deploying small prototypes with ease.

Scikit-Learn

One of the top Python libraries for Machine Learning, Scikit Learn integrates swiftly with NumPy and Pandas. The main purpose of Scikit Learn is to focus only on data modeling. It is the fundamental library that engineers use to build end-to-end Machine Learning applications. There are also some excellent data pre-processing tools in the library.

Theano

Built on NumPy, Theano is a dynamic Machine Learning framework with a powerful interface, similar to the NumPy library. Theano helps to build efficient Machine Learning algorithms. It offers faster and stable monitoring of the most complicated variables.

Net

Known as one of the most popular Deep Learning frameworks for neural network development, MXNet is a flexible framework as it supports multiple programming languages, including Python, Java, C++, Scala, Go, R, and more. MXNet is one of the best Python frameworks for Deep learning as it is portable and scales to multiple GPU ports. It also offers faster context switching and optimized computation for different functions.

Pandas

Another of the highly known Python Machine Learning libraries in Python. Engineers use the library for data manipulation and analysis. It works amazingly well with structured data for Machine Learning algorithms. It offers great features to deploy ML and DL-based applications. Pandas assists with data reshaping, dataset joining, data filtration, alignment and easily handles missing data as well. It also provides a 2-D representation of data to make things convenient for python developers.

NumPy

An emerging package and one of the most useful frameworks for Machine Learning engineers, NumPy enables developers to process large amounts of multidimensional arrays. It is also useful in Fourier transforms, linear algebra, and other mathematical functions.

NumPy offers developers the capability to add speedy computations in the solution. Complicated functions can be easily executed – all thanks to NumPy's power for scientific and numerical computing.

NLTK

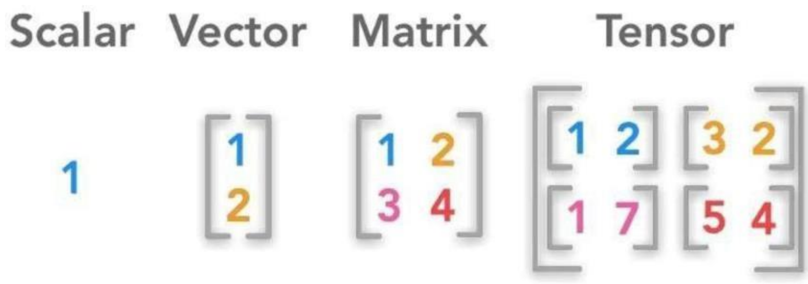
Also known as the Natural Language ToolKit, NLTK is used by a Python web development company to integrate Natural Language Processing. The tool is useful for Deep Learning solutions that require high amounts of text and speech processing. NLTK works well with FrameNet, WordNet, and Word2Vec for proper language processing.

Spark ML

The Spark ML framework simplifies matrix multiplication for Machine Learning. It divides the matrix into slices and runs the calculation on different servers. It requires a distributed architecture, ensuring that the computer doesn't run out of memory while performing valuable operations. Engineers that use Spark for Big Data and Data Analytics may find it easy to work with Spark ML.

Tensors

A tensor is an array that represents the types of data in the TensorFlow Python deep-learning library. A tensor, as compared to a one-dimensional vector or array or a two-dimensional matrix, can have n dimensions. The values in a tensor contain identical data types with a specified shape. Dimensionality is represented by the shape. A vector, for example, is a one-dimensional tensor, a matrix is a two-dimensional tensor, and a scalar is a zero-dimensional tensor.



Example:

```
# importing tensorflow
import tensorflow as tf

# creating nodes in computation graph
node1 = tf.constant(3, dtype=tf.int32)
node2 = tf.constant(5, dtype=tf.int32)
node3 = tf.add(node1, node2)

# create tensorflow session object
sess = tf.Session()

# evaluating node3 and printing the result
print("Sum of node1 and node2 is:",sess.run(node3))

# closing the session
sess.close()
```

KERAS

The Keras Workflow Model

- Define the training data—the input tensor and the target tensor
- Build a model or a set of Keras layers, which leads to the target tensor
- Structure a learning process by adding metrics, choosing a loss function, and defining the optimizer
- Use the fit() method to work through the training data and teach the model

Example:

```
from keras import models
from keras import layers
model = models.Sequential()
model.add(layers.Dense(32, activation='relu', input_shape=(784,)))
model.add(layers.Dense(10, activation='softmax'))
```

```
input_tensor = layers.Input(shape=(784,))
x = layers.Dense(32, activation='relu')(input_tensor)
output_tensor = layers.Dense(10, activation='softmax')(x)
model = models.Model(inputs=input_tensor, outputs=output_tensor)
from keras import optimizers

model.compile(optimizer=optimizers.RMSprop(lr=0.001),
loss='mse',
metrics=['accuracy'])
model.fit(input_tensor, target_tensor, batch_size=128, epochs=10)
```

RESULT:

Thus, the study on different framework and libraries used for deep learning was completed.

EX.NO:3a	SIMPLE NEURAL NETWORK FORMATION
DATE:	

AIM:

To implement simple neural network for AND & OR gate using McCulloch-Pitts neuron.

ALGORITHM:

Step 1: Create a python notebookStep
2: Initialize the input vector Step 3:
Initialize the threshold value
Step 4: Calculate weighted sum of neuron in the output layer
Step 5: Find the output based on the relationship between weighted sum and threshold value

CODE:

AND :

```
import numpy as np
def nparray(a,b):
a=np.array([0,0,1,1])
  b=np.array([0,1,0,1])
  sum=np.add(a,b)
  threshold=2
  for i in sum:
      if i>=threshold:
          print(1)
      else:
          print(0)
a=[0,0,1,1]
b=[0,1,0,1]
print("\nAND")
nparray(a,b)
```

Output:

```
AND
0
0
0
1
```

OR:

```
def nparray(a,b):
  a=np.array([0,0,1,1])
  b=np.array([0,1,0,1])
  sum=np.add(a,b)
  threshold=1
  for i in sum:
      if i>=threshold:
          print(1)
      else:
          print(0)
a=[0,0,1,1]
b=[0,1,0,1]
print("\nOR")
nparray(a,b)
```

Output:

```
OR
0
1
1
1
```

RESULT:

Thus, implementation of simple neural network for AND & OR gate using McCulloch-Pitts neuron was done successfully.

EX.NO:3b	SIMPLE NEURAL NETWORK FORMATION FOR BREAST CANCER DATASET
DATE:	

AIM:

To implement simple neural network over breast cancer dataset using McCulloch-Pitts neuron.

ALGORITHM:

Step 1: Create a python notebook
Step 2: Initialize the input vector Step
3: Initialize the threshold value
Step 4: Calculate weighted sum of neuron in the output layer
Step 5: Find the output based on the relationship between weighted sum and threshold value

CODE:

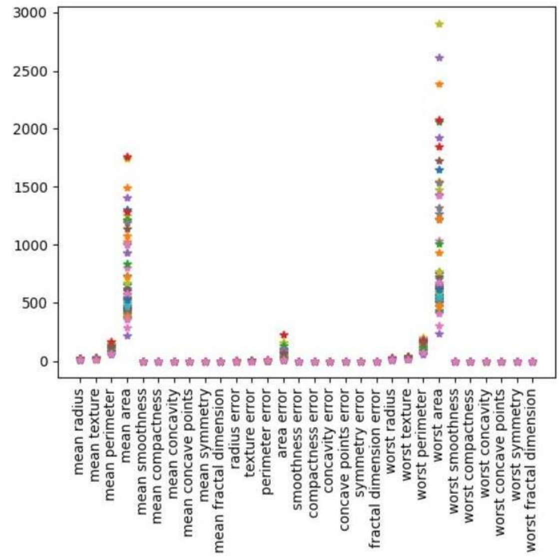
```
import sklearn.datasets
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Load the Breast Cancer dataset
breast_cancer = sklearn.datasets.load_breast_cancer()
X = breast_cancer.data
Y = breast_cancer.target
print(X.shape, Y.shape)

# Create a DataFrame with the features and the target
data = pd.DataFrame(breast_cancer.data, columns=breast_cancer.feature_names)
data['class'] = breast_cancer.target
print(data.head())
print(data.describe())
print(data['class'].value_counts())
print(breast_cancer.target_names)
print(data.groupby('class').mean())

# Train-test split
X = data.drop('class', axis=1)
Y = data['class']
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.1, stratify=Y, random_state=1)
print(Y.shape, Y_train.shape, Y_test.shape)
print(Y.mean(), Y_train.mean(), Y_test.mean())
```

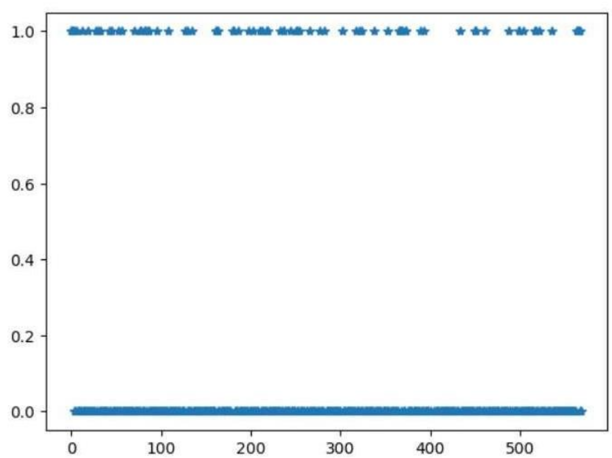
```
print(X_train.mean(), X_test.mean(), X.mean())

# Binarisation of input
plt.plot(X_test.T, '*')
plt.xticks(rotation='vertical')
plt.show()

# Binarising a specific feature: 'mean area'
X_binarised_3_train = X_train['mean area'].map(lambda x: 0 if x < 1000 else 1)
plt.plot(X_binarised_3_train, '*')
plt.show()

# Binarise the entire dataset
X_binarised_train = X_train.apply(pd.cut, bins=2, labels=[1, 0])
plt.plot(X_binarised_train.T, '*')
plt.xticks(rotation='vertical')
plt.show()

X_binarised_test = X_test.apply(pd.cut, bins=2, labels=[1, 0])

# Convert to numpy arrays
X_binarised_test = X_binarised_test.values
X_binarised_train = X_binarised_train.values
print(type(X_binarised_test), type(X_binarised_train))

# MP Neuron Class
class MPNeuron:
    def __init__(self):
        self.b = None

    def model(self, x):
        return sum(x) >= self.b

    def predict(self, X):
        Y = []
        for x in X:
            result = self.model(x)
            Y.append(result)
        return np.array(Y)

    def fit(self, X, Y):
        accuracy = {}
        for b in range(X.shape[1] + 1):
            self.b = b
            Y_pred = self.predict(X)
            accuracy[b] = accuracy_score(Y_pred, Y)
        best_b = max(accuracy, key=accuracy.get)
        self.b = best_b
        print('Optimal value of b is', best_b)
```

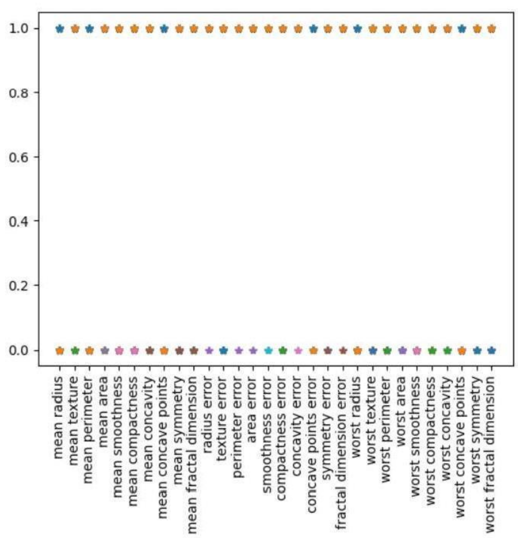
```
print('Highest accuracy is', accuracy[best_b])

# Instantiate and train the MP Neuron model
mp_neuron = MPNeuron()
mp_neuron.fit(X_binarised_train, Y_train)

# Predictions on the test set
Y_test_pred = mp_neuron.predict(X_binarised_test)
test_accuracy = accuracy_score(Y_test_pred, Y_test)
print('Test set accuracy:', test_accuracy)
```

OUTPUT:

```
(569, 30) (569,)
   mean radius  mean texture  mean perimeter  mean area  mean smoothness  \
0        17.99         10.38          122.80     1001.0          0.11840
1        20.57         17.77          132.90     1326.0          0.08474
2        19.69         21.25          130.00     1203.0          0.10960
3        11.42         20.38           77.58      386.1          0.14250
4        20.29         14.34          135.10     1297.0          0.10030

   mean compactness  mean concavity  mean concave points  mean symmetry  \
0           0.27760          0.3001              0.14710         0.2419
1           0.07864          0.0869              0.07017         0.1812
2           0.15990          0.1974              0.12790         0.2069
3           0.28390          0.2414              0.10520         0.2597
4           0.13280          0.1980              0.10430         0.1809


[2 rows x 30 columns]
(569,) (512,) (57,)
0.6274165202108963 0.626953125 0.631578947368421
mean radius                 14.058656
mean texture                19.309668
mean perimeter              91.530488
mean area                  648.097266
mean smoothness              0.096568
mean compactness             0.105144
mean concavity               0.089342
mean concave points          0.048892
mean symmetry                0.181961
mean fractal dimension       0.062979
radius error                 0.403659
```


3000
2500
2000
1500
1000
500
0
mean radius
mean texture
mean perimeter
mean area
mean smoothness
mean compactness
mean concavity
mean concave points
mean symmetry
mean fractal dimension
radius error
texture error
perimeter error
area error
smoothness error
compactness error
concavity error
concave points error
symmetry error
fractal dimension error
worst radius
worst texture
worst perimeter
worst area
worst smoothness
worst compactness
worst concavity
worst concave points
worst symmetry
worst fractal dimension


1.0
0.8
0.6
0.4
0.2
0.0
0
100
200
300
400
500


1.0
0.8
0.6
0.4
0.2
0.0
mean radius
mean texture
mean perimeter
mean area
mean smoothness
mean compactness
mean concavity
mean concave points
mean symmetry
mean fractal dimension
radius error
texture error
perimeter error
area error
smoothness error
compactness error
concavity error
concave points error
symmetry error
fractal dimension error
worst radius
worst texture
worst perimeter
worst area
worst smoothness
worst compactness
worst concavity
worst concave points
worst symmetry
worst fractal dimension

```
<class 'numpy.ndarray'> <class 'numpy.ndarray'>
Optimal value of b is 28
Highest accuracy is 0.849609375
Test set accuracy: 0.7894736842105263
```

Result:

Thus, the implementation of simple neural network over breast cancer dataset using McCulloch-Pitts neuron was executed successfully.

EX.NO:4	PERCEPTRON FOR BINARY CLASSIFICATION
DATE:	

AIM:

To implement simple neural network over Iris dataset using perceptron learning algorithm.

ALGORITHM:

Step 1: Create a python notebook
Step 2: Initialize the input vector and weight matrix
Step 3: Calculate weighted sum of neuron in the output layer
Step 4: Apply activation function over the weighted sum
Step 5: Calculate new weight value based on perceptron learning rule

$$y = 1, \text{if} \sum_i w_i x_i >= b$$

$$y = 0, \text{otherwise}$$

Step 6: Repeat step 3 to 5 until network converges

CODE:

```
import sklearn

import sklearn.datasets

import numpy as np

import pandas as pd

# Load the Iris dataset

iris = sklearn.datasets.load_iris()

X = iris.data

Y = iris.target

print(X.shape, Y.shape)

# Create a DataFrame from the Iris dataset

data = pd.DataFrame(iris.data, columns=iris.feature_names)

data['class'] = iris.target

print(data.head())
```

print(data.describe())

print(data['class'].value_counts())

print(iris.target_names)

print(data.groupby('class').mean())

```
(150, 4) (150,)
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2

   class
0      0
1      0
2      0
3      0
4      0
       sepal length (cm)  sepal width (cm)  petal length (cm)  \
count         150.000000        150.000000         150.000000
mean            5.843333          3.057333           3.758000
std             0.828066          0.435866           1.765298
min             4.300000          2.000000           1.000000
25%             5.100000          2.800000           1.600000
50%             5.800000          3.000000           4.350000
75%             6.400000          3.300000           5.100000
max             7.900000          4.400000           6.900000
```

#Train test split

from sklearn.model_selection import train_test_split

X = data.drop('class', axis=1)

Y = data['class']

type(X)

X_train, X_test, Y_train, Y_test = train_test_split(X, Y)

print(Y.shape, Y_train.shape, Y_test.shape)

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.1,

stratify = Y,

random_state=1)

print(X_train.mean(), X_test.mean(), X.mean())

```
(569,) (426,) (143,)
mean radius                  14.058656
mean texture                 19.309668
mean perimeter               91.530488
mean area                   648.097266
mean smoothness               0.096568
mean compactness              0.105144
mean concavity                0.089342
mean concave points           0.048892
mean symmetry                 0.181961
mean fractal dimension        0.062979
radius error                  0.403659
texture error                 1.206856
perimeter error               2.861173
area error                   39.935506
smoothness error              0.007067
compactness error             0.025681
concavity error               0.032328
concave points error          0.011963
symmetry error                0.020584
fractal dimension error       0.003815
```

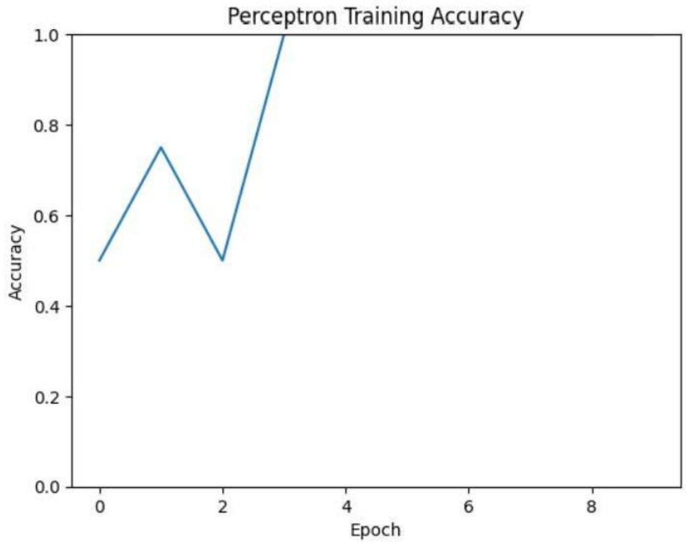
```
class Perceptron:
    def___init_(self):
        self.w = None
        self.b = None

    def model(self, x):
        # Predicts 1 if the dot product of weights and x is >= bias, else 0
        return 1 if np.dot(self.w, x) >= self.b else 0

    def predict(self, X):
        # Predicts outputs for a batch of inputs
        Y = [self.model(x) for x in X]
        return np.array(Y)

    def fit(self, X, Y, epochs=1):
        self.w = np.ones(X.shape[1])
        self.b = 0
        accuracy = {}
        max_accuracy = 0
```

```
wt_matrix = []
for epoch in range(epochs):
    for x, y in zip(X, Y):
        y_pred = self.model(x)
        if y == 1 and y_pred == 0:
            self.w += x
            self.b -= 1
        elif y == 0 and y_pred == 1:
            self.w -= x
            self.b += 1
    wt_matrix.append(self.w.copy())
    acc = accuracy_score(self.predict(X), Y)
    accuracy[epoch] = acc
    if acc > max_accuracy:
        max_accuracy = acc
        chkptw = self.w.copy()
        chkptb = self.b
  # Restore the best weights and bias
  self.w = chkptw
  self.b = chkptb
  print('Max accuracy:', max_accuracy)
  # Plotting accuracy over epochs
  plt.plot(list(accuracy.values()))
  plt.ylim([0, 1])
  plt.xlabel('Epoch')
  plt.ylabel('Accuracy')
  plt.title('Perceptron Training Accuracy')
  plt.show()
  return np.array(wt_matrix)
```

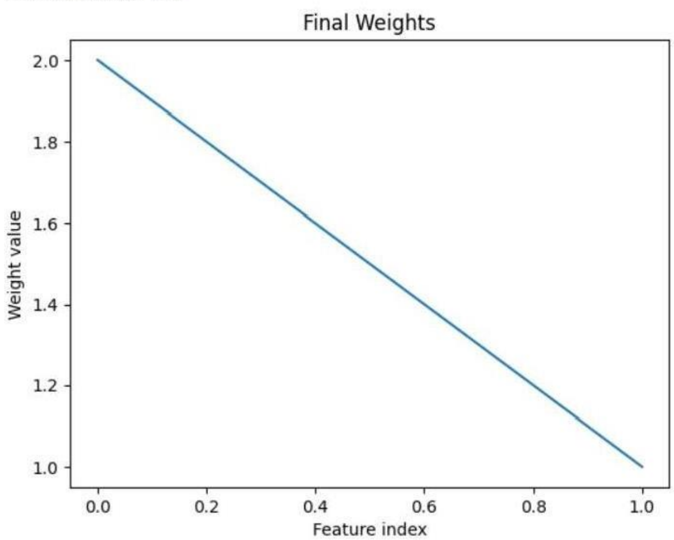
```
if__name__== "_main_":
    # Sample data
    X_train = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
    Y_train = np.array([0, 0, 0, 1])
    X_test = np.array([[0, 1], [1, 1]])
    Y_test = np.array([0, 1])
    # Initialize and fit the Perceptron
    perceptron = Perceptron()
    wt_matrix = perceptron.fit(X_train, Y_train, epochs=10)
    # Predict and evaluate accuracy on the test set
    Y_pred_test = perceptron.predict(X_test)
    print('Test accuracy:', accuracy_score(Y_pred_test, Y_test))
    # Plotting the final weights
    plt.plot(wt_matrix[-1, :])
    plt.xlabel('Feature index')
    plt.ylabel('Weight value')
    plt.title('Final Weights')
    plt.show()
```

OUTPUT:

Max accuracy: 1.0



Test accuracy: 1.0



RESULT:

Thus, the implementation of simple neural network over breast cancer dataset using perceptron learning algorithm was successfully completed.

EX.NO:5a	FEED FORWARD DNN-BINARY CLASSIFICATION
DATE:	

AIM:

To write a python program to implement feed forward deep neural network for binary classification

ALGORITHM:

Step 1: Create a python notebook. Step
2: Load the dataset
Step 3: Split the input and class labels in the dataset into training and testing data.
Step 4: Construct the neural network (Specify the no. of. neurons & activation function).
Step 5: Compile the model with binary loss function and optimizer.
Step 6: Train the model for 150 epochs using backpropagation algorithm and observe the accuracy.

PROGRAM:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Load the dataset
dataset = pd.read_csv(r"F:\heart.csv")

# Display the first few rows of the dataset
print(dataset.head())

# Split the dataset into training and testing sets
train, test = train_test_split(dataset, test_size=0.25, random_state=0, stratify=dataset['target'])

# Separate features and labels
train_X = train.iloc[:, :-1] # All columns except the last one
test_X = test.iloc[:, :-1]
train_Y = train['target']
test_Y = test['target']
```

```
# Display the first few rows of the training data
print(train_X.head())
print(train_Y.head())

# Define the model
model = Sequential()
model.add(Dense(12, input_shape=(train_X.shape[1],), activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

# Compile the model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Display the model summary
model.summary()

# Train the model
model.fit(train_X, train_Y, epochs=150, batch_size=10)

# Evaluate the model on the test data
_, accuracy = model.evaluate(test_X, test_Y)
print('Accuracy: %.2f' % (accuracy * 100))
```

OUTPUT:

```
   Area  Perimeter  Compactness  Kernel.Length  Kernel.Width  Asymmetry.Coeff  Kernel.Groove  Type
0  15.26      14.84       0.8710          5.763         3.312            2.221          5.220     1
1  14.88      14.57       0.8811          5.554         3.333            1.018          4.956     1
2  14.29      14.09       0.9050          5.291         3.337            2.699          4.825     1
3  13.84      13.94       0.8955          5.324         3.379            2.259          4.805     1
4  16.14      14.99       0.9034          5.658         3.562            1.355          5.175     1
     Area  Perimeter  Compactness  Kernel.Length  Kernel.Width  Asymmetry.Coeff  Kernel.Groove
94   18.17      16.26       0.8637          6.271         3.512            2.853          6.273
95   18.72      16.34       0.8810          6.219         3.684            2.188          6.097
178  10.91      12.80       0.8372          5.088         2.675            4.179          4.956
90   18.36      16.52       0.8452          6.666         3.485            4.933          6.448
126  18.94      16.32       0.8942          6.144         3.825            2.908          5.949
94     2
95     2
178    3
90     2
126    2
```

```
 Total params: 209 (836.00 B)
 Trainable params: 209 (836.00 B)
 Non-trainable params: 0 (0.00 B)
Epoch 1/150
15/15 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.2746 - loss: -2.0328
Epoch 2/150
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.3477 - loss: -3.8251
Epoch 3/150
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.3595 - loss: -5.4914
Epoch 4/150
```

```
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.2855 - loss: -30328.6348
Epoch 149/150
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.3026 - loss: -31483.1523
Epoch 150/150
15/15 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.3267 - loss: -30272.4785
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.3204 - loss: -31510.6914
Accuracy: 34.00
```

RESULT:

Thus, the python program to implement feed forward deep neural network for binary classification is executed successfully.

EX.NO:5b	FEED FORWARD DNN- MULTICLASS CLASSIFICATION
DATE:	

AIM:

To write a python program to implement feed forward deep neural network for multi class classification

ALGORITHM:

Step 1: Create a python notebook. Step 2: Load the dataset

Step 3: Split the input and class labels in the dataset into training and testing data. Step 4: Convert output data into one-hot encoded representation

Step 5: Construct the neural network (Specify the no. of. neurons & activation function). Step 6: Compile the model with binary loss function and optimizer.

Step 7: Train the model for 150 epochs using backpropagation algorithm and observe the accuracy.

PROGRAM:

```
import numpy as np

import tensorflow as tf

from tensorflow.keras.datasets  import cifar10

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Flatten

from tensorflow.keras.utils import to_categorical

import matplotlib.pyplot as plt

# Load the CIFAR-10 dataset

(x_train, y_train), (x_valid, y_valid) = cifar10.load_data()

# Inspect data shapes and types

print(x_train.shape)  # (50000, 32, 32, 3)
```

```
print(x_valid.shape) # (10000, 32, 32, 3)
# Normalize the data
x_train = x_train / 255.0
x_valid = x_valid / 255.0

# Convert labels to one-hot encoding
num_categories = 10
y_train = to_categorical(y_train, num_categories)
y_valid = to_categorical(y_valid, num_categories)

# Define the model
model = Sequential([
    Flatten(input_shape=(32, 32, 3)), # Flatten the input images
   Dense(units=1024, activation='relu'),
    Dense(units=512, activation='relu'),
    Dense(units=256, activation='relu'),
    Dense(units=num_categories, activation='softmax')
])
# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
# Display the model summary
model.summary()
# Train the model
history = model.fit(
    x_train, y_train,
    epochs=10,  # Increased number of epochs
    batch_size=64, # Adjusted batch size
    validation_data=(x_valid, y_valid))
```

Evaluate the model on the test data

loss, accuracy = model.evaluate(x_valid, y_valid)

print('Accuracy: %.2f%%' % (accuracy * 100))

Plot training and validation accuracy

plt.plot(history.history['accuracy'], label='Train Accuracy')

plt.plot(history.history['val_accuracy'], label='Validation Accuracy')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend()

plt.show()

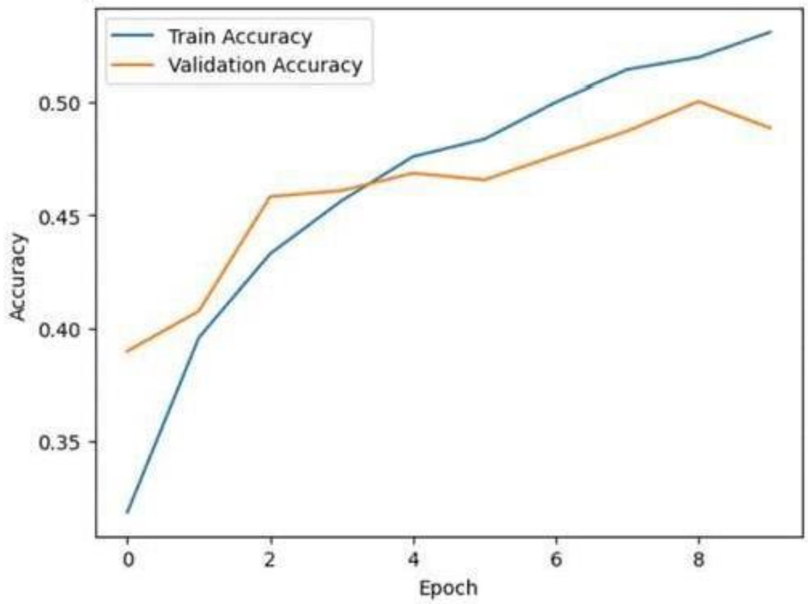
OUTPUT:

```
(50000, 32, 32, 3)
(10000, 32, 32, 3)
Model: "sequential_3"
```

Layer (type)	Output Shape	Param #
flatten_2 (Flatten)	(None, 3072)	0
dense_11 (Dense)	(None, 1024)	3,146,752
dense_12 (Dense)	(None, 512)	524,800
dense_13 (Dense)	(None, 256)	131,328

```
Total params: 3,805,450 (14.52 MB)
Trainable params: 3,805,450 (14.52 MB)
Non-trainable params: 0 (0.00 B)
Epoch 1/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 30s 36ms/step - accuracy: 0.2656 - loss: 2.0485 - val_accuracy: 0.3898 - val_loss: 1.7020
Epoch 2/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 29s 37ms/step - accuracy: 0.3857 - loss: 1.7043 - val_accuracy: 0.4075 - val_loss: 1.6516
Epoch 3/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 28s 35ms/step - accuracy: 0.4268 - loss: 1.6008 - val_accuracy: 0.4582 - val_loss: 1.5331
```

Accuracy: 48.86%



RESULT:

Thus, the python program to implement feed forward deep neural network for multi class classification is executed successfully.