LAB ASSIGNMENT REPORT

On

Deep Learning Using Python Lab (EAI752)

FACULTY OF ENGINEERING AND COMPUTING SCIENCES

B. Tech. CSE (Specialization in AI, ML, DL)

4th Year / 7th Semester

Session: 2022-23

Submitted To:	Submitted By:
Mr. Amit Sharma	Shivam Kumar
Assistant Professor	TCA1959044

S. No.	Program Name	Page No.	Date	Sign	Remark
	Assignment-1				
1.	Write a program to implement and demonstrate FIND-S Algorithm for finding the most specific hypothesis based on any sample dataset.				
2.	Write a program to Perform the following operation in Python. a. Vector operations: Addition, Subtraction, Multiplication, Division, Dot Product, Scalar Product b. Matrix Operations: Addition, Subtraction, Multiplication				
3.	Write a program to demonstrate and implementation of Linear Regression using sample dataset.				
4.	Write a program for the implementation of Logistic Regression using sample dataset.				
5.	Write a program for the implementation of SVM Linear and Radial for understanding the performance parameters of learning algorithms.				
	Assignment-2				
6.	Write a program to demonstrate the working of Decision Tree based ID3 algorithm. Use an appropriate dataset for building a decision tree and apply this knowledge to classify a new sample.				
7.	Write a program to build an ANN by implementing the Back Propagation Algorithm.				
8.	Write a program to build an ANN by implementing the Feed Forward Algorithm.				
9.	Write a program for calculating the Best Association rule based on classification using min support and confidence threshold.				
10.	Write a program to perform basic operations of TensorFlow.				
11.	Write a program to build CNN model for Text processing application.				

Assignment -1

Program 1. Write a program to implement and demonstrate **FIND-S Algorithm** for finding the most specific hypothesis based on any sample dataset.

Solution:

Sample Iris Dataset:

	sepallength	sepalwidth	petallength	petalwidth	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

Python Code:

```
import pandas as pd
import numpy as np
data = pd.read_csv("iris_csv.csv")
print(data.head())
#making an array of all the attributes
d = np.array(data)[:,:-1]
print("n The attributes are: ",d)
#segragating the target that has positive and negative examples
target = np.array(data)[:,-1]
print("n The target is: ",target)
#training function to implement find-s algorithm
def train(c,t):
   for i, val in enumerate(t):
      if val == "Iris-setosa":
         specific_hypothesis = c[i].copy()
```

```
        elif val == "Iris-virginica":
            specific_hypothesis = c[i].copy()
    for i, val in enumerate(c):
        if t[i] == "Iris-setosa" or "Iris-virginica":
            for x in range(len(specific_hypothesis)):
                if val[x] != specific_hypothesis[x]:
                    specific_hypothesis[x] = '?'
                else:
                    pass

    return specific_hypothesis
#obtaining the final hypothesis
print("The final hypothesis is:",train(d,target))
```

OUTPUT:

```
The final hypothesis is: ['?' '?' '?' '?']
```

Program 2. Write a program to Perform the following operation in Python.

a. Vector operations: Addition, Subtraction, Multiplication, Division, Dot Product, Scalar Product
b. Matrix Operations: Addition, Subtraction, Multiplication

Vector Operations:

```
import numpy as np
lst1 = [10,20,30,40,50]
lst2 = [1,2,3,4,5]
#creating vectors from list
```

```
vctr1 = np.array(lst1)

vctr2= np.array(lst2)

print("First vector",vctr1)

print("Second Vector",vctr2)
```

First vector [10 20 30 40 50]
Second Vector [1 2 3 4 5]

```
#vector addition

vctr_add = vctr1+vctr2

print("Addition=",vctr_add)
```

Addition= [11 22 33 44 55]

```
#vector subtraction

vctr_sub = vctr1-vctr2

print("Subtraction=",vctr_sub)
```

Subtraction= [9 18 27 36 45]

```
#vector multiplication

vctr_mul = vctr1*vctr2

print("Multiplication=",vctr_mul)
```

Multiplication= [10 40 90 160 250]

```
#vector division

vctr_div = vctr1/vctr2

print("Division=",vctr_div)
```

Division= [10. 10. 10. 10. 10.]

```
#dot product

vctr_dot = vctr1.dot(vctr2)

print("Dot product=",vctr_dot)
```

Dot product= 550

Matrix Operations:

```
#creating Matrices
x = np.array([[1, 2], [4, 5]])
y = np.array([[7, 8], [9, 10]])
# adding matrices
print("Addition=\n",np.add(x,y))
```

Addition=

[[8 10]

[13 15]]

```
#subtracting matrices
print("Subtraction=\n",np.subtract(x,y))
```

Subtraction=

[[-6 -6]

[-5 -5]]

```
#multiplying matrices
print("Multiplication=\n",np.multiply(x,y))
```

Multiplication=

[[7 16]

[36 50]]

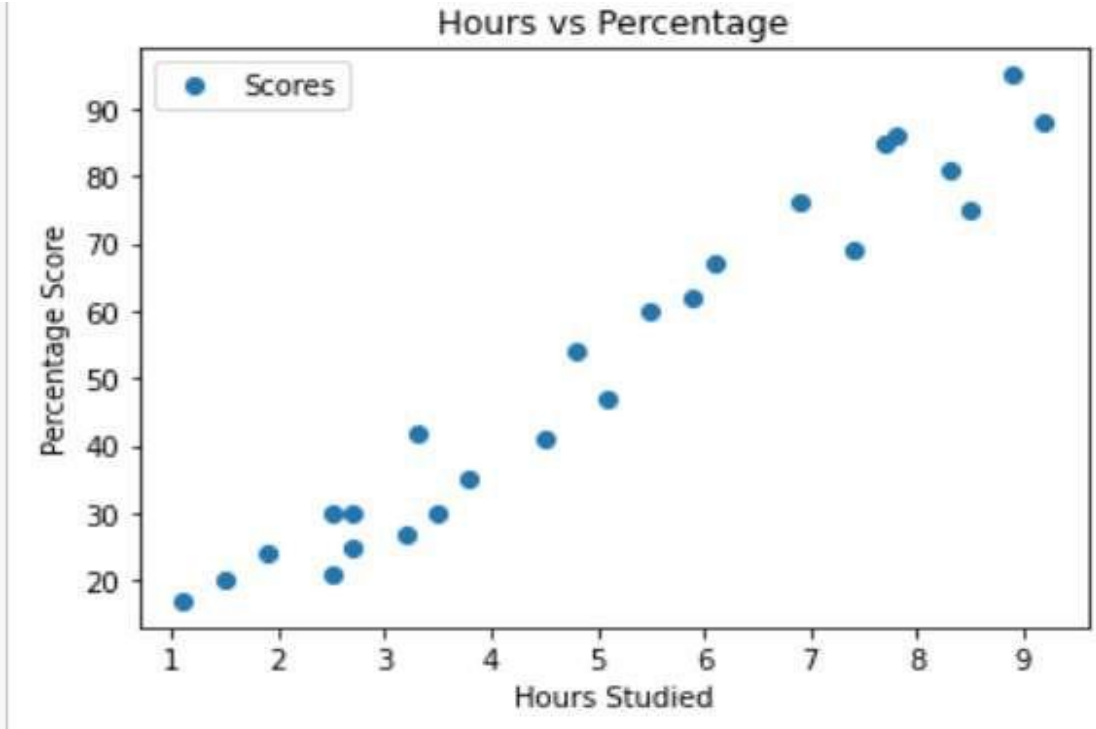
Program 3.Write a program to demonstrate and implementation of Linear Regression using sample dataset.

Solution: Sample Dataset:

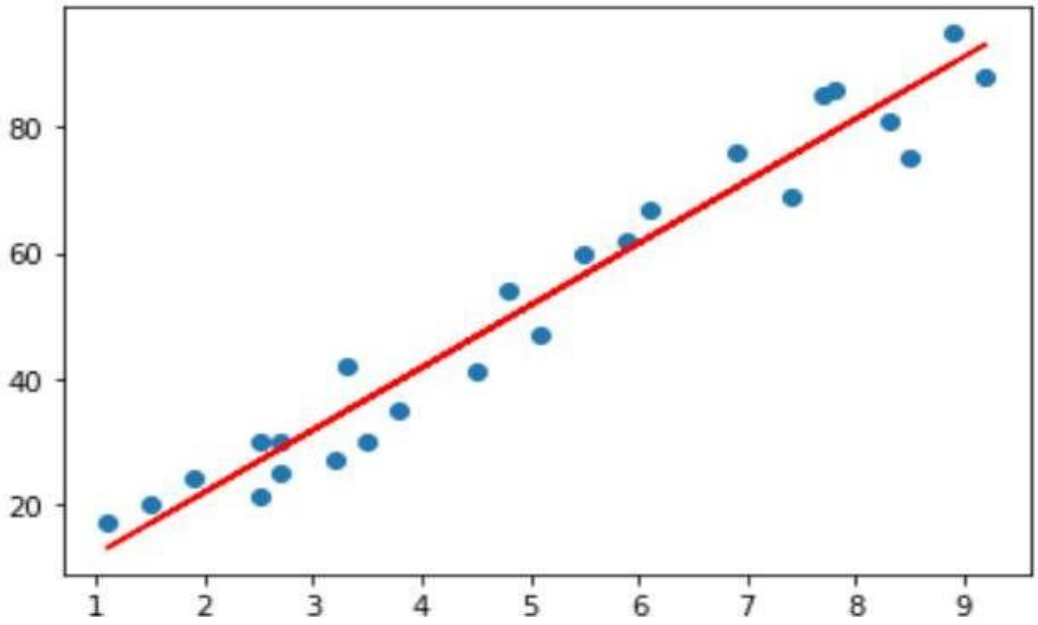
	Hours	Scores
0	2.5	21
1	5.1	47
2	3.2	27
3	8.5	75
4	3.5	30
5	1.5	20

Python Code:

```
# importing libraries
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
data = pd.read_csv("student_scores.csv")
data.head(10)
data.plot(x='Hours', y='Scores', style='o')
plt.title('Hours vs Percentage')
plt.xlabel('Hours Studied')
plt.ylabel('Percentage Score')
plt.show()
```



```
# division of data into "attributes" (inputs) and "labels" (outputs)
X = data.iloc[:, :-1].values
y = data.iloc[:, 1].values
#Splitting the data into training and testing sets, and training the algorithm
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
regressor = LinearRegression()
regressor.fit(X_train.reshape(-1,1), y_train)
# Plotting the regression line
line = regressor.coef_*X+regressor.intercept_
# Plotting for the test data
plt.scatter(X, y)
plt.plot(X, line,color='red');
plt.show()
```




```
# Model Prediction
y_pred = regressor.predict(X_test)
```

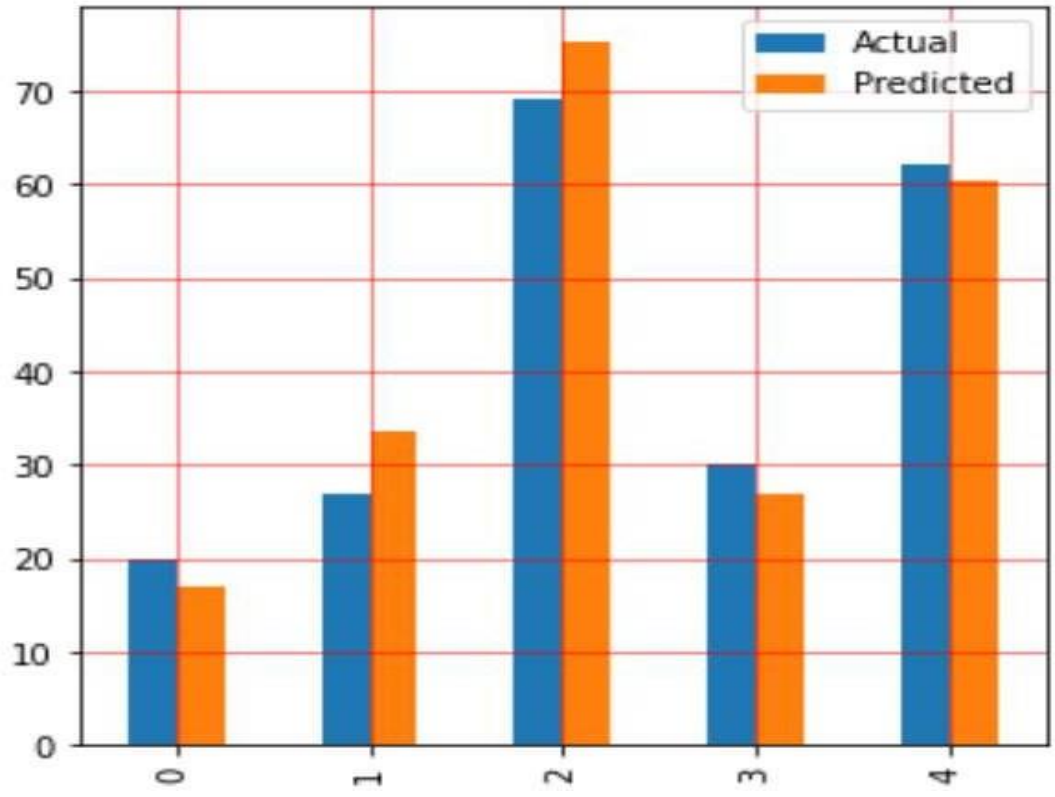
Comparing Actual vs Predicted

```
df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
df
#Estimating training and test score
print("Training Score:",regressor.score(X_train,y_train))
print("Test Score:",regressor.score(X_test,y_test))
```

```
Training Score: 0.9515510725211552
Test Score: 0.9454906892105355
```

Plotting the Bar graph to depict the difference between the actual and predicted value

```
df.plot(kind='bar',figsize=(5,5))
plt.grid(which='major', linewidth='0.5', color='red')
plt.grid(which='minor', linewidth='0.5', color='blue')
plt.show()
```



```
# Testing the model with our own data
hours = 9.25
test = np.array([hours])
test = test.reshape(-1, 1)
own_pred = regressor.predict(test)
print("No of Hours = {}".format(hours))
print("Predicted Score = {}".format(own_pred[0]))
```

```
No of Hours = 9.25
Predicted Score = 93.69173248737535
```

```
from sklearn import metrics
print('Mean Absolute Error:',metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print('R-2:', metrics.r2_score(y_test, y_pred))
```

OUTPUT:

Mean Absolute Error: 4.183859899002975

Mean Squared Error: 21.598769307217406

Root Mean Squared Error: 4.647447612100367

R-2: 0.9454906892105355

Program 4. Write a program for the implementation of Logistic Regression using sample dataset.

Solution:

Sample Dataset:

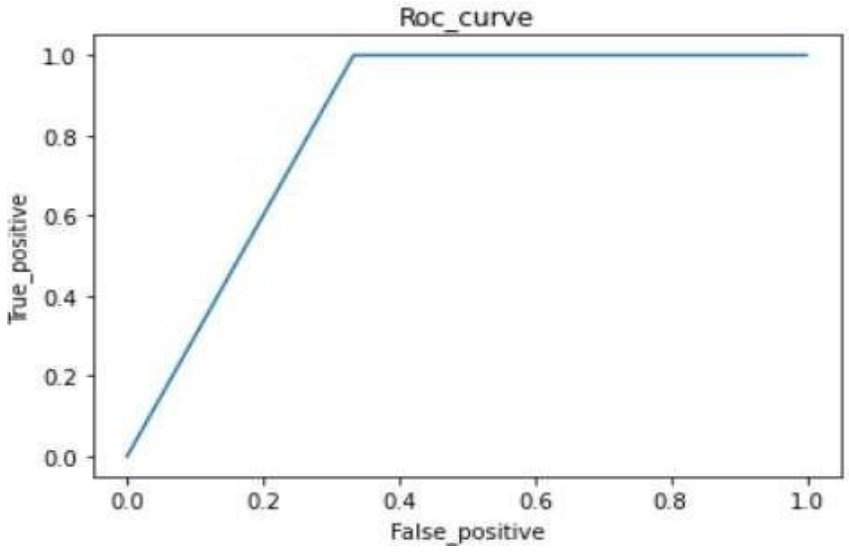
	age	bought_insurance
0	22	0
1	25	0
2	47	1
3	52	0
4	46	1

Python Code:

```
import pandas as pd
df=pd.read_csv('insurance_data.csv')
df.head()
import sklearn as sk
from sklearn.model_selection import train_test_split
age=df['age']
insurance=df['bought_insurance']
import numpy as np
insurance=np.array(insurance).reshape(-1,1)
age=np.array(age).reshape(-1,1)
x_train,x_test,y_train,y_test=train_test_split(age,insurance,test_size=0.2)
from sklearn.linear_model import LogisticRegression
model=LogisticRegression()
model.fit(x_train,y_train)
```

```
y_pred=model.predict(x_test)
from sklearn.metrics import classification_report, confusion_matrix
confusion_matrix(y_test,y_pred)
from sklearn.metrics import accuracy_score,confusion_matrix
print("Accuracy=",accuracy_score(y_test,y_pred)*100)
cm=confusion_matrix(y_test,y_pred)
print("Confusion Matrix \n",cm)
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve
fpr, tpr, _=roc_curve(y_test,y_pred)
plt.title('Roc_curve')
# plt.figure(''figsize''=100)
plt.xlabel('False_positive')
plt.ylabel('True_positive')
plt.plot(fpr,tpr)
```

OUTPUT:



Accuracy= 83.33333333333334

Confusion Matrix

[[2 1]

[0 3]]

Program 5. Write a program for the implementation of SVM Linear and Radial for understanding the performance parameters of learning algorithms.

Solution: SVM Linear:

Sample Dataset:

	Variance	Skewness	Curtosis	Entropy	Class
0	3.62160	8.6661	-2.8073	-0.44699	0
1	4.54590	8.1674	-2.4586	-1.46210	0
2	3.86600	-2.6383	1.9242	0.10645	0
3	3.45660	9.5228	-4.0112	-3.59440	0
4	0.32924	-4.4552	4.5718	-0.98880	0

Python Code:

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

%matplotlib inline

data = pd.read_csv("bill_authentication.csv")

data.head()

#dividing into attricutes and lables

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

y =data['Class']

#spliting train and test data
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20)
#model training
from sklearn.svm import SVC
svclassifier = SVC(kernel='linear')
svclassifier.fit(X_train, y_train)
#predicting output on test data
y_pred = svclassifier.predict(X_test)
#evaluating the eprformance
from sklearn.metrics import confusion_matrix,accuracy_score
print("Confusion Matrix \n ",confusion_matrix(y_test,y_pred))
print("Accuracy=",accuracy_score(y_test,y_pred)*100)
```

OUTPUT:

```
Confusion Matrix
 [[134  0]
 [  2 139]]
Accuracy= 99.27272727272727
```

SVM Radial:

Sample Dataset:

	sepallength	sepalwidth	petallength	petalwidth	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

Python Code:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
irisdata = pd.read_csv("iris_csv.csv")
print(irisdata.head())
X = irisdata.drop('class', axis=1)
y = irisdata['class']
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.30)
#Using sigmoid kernel
from sklearn.svm import SVC
svclassifier = SVC(kernel='sigmoid')
svclassifier.fit(X_train, y_train)
y_pred = svclassifier.predict(X_test)
#evaluating the performance
from sklearn.metrics import confusion_matrix,accuracy_score
print(confusion_matrix(y_test,y_pred))
print("Accuracy=",accuracy_score(y_test,y_pred)*100)
```

OUTPUT:

```
Confusion Matrix
 [[ 0  0 15]
 [ 0  0 16]
 [ 0  0 14]]
Accuracy= 31.11111111111111
```

#Using gaussian kernel

```
from sklearn.svm import SVC
svclassifier = SVC(kernel='rbf')
svclassifier.fit(X_train, y_train)
y_pred = svclassifier.predict(X_test)
#evaluating the performance
from sklearn.metrics import confusion_matrix,accuracy_score
print("Confusion Matrix\n",confusion_matrix(y_test,y_pred))
print("Accuracy=",accuracy_score(y_test,y_pred)*100)
```

OUTPUT:

```
Confusion Matrix
 [[15  0  0]
 [ 0 13  3]
 [ 0  0 14]]
Accuracy= 93.33333333333333
```

Assignment – 2

Program 6. Write a program to demonstrate the working of Decision Tree based ID3 algorithm. Use an appropriate dataset for building a decision tree and apply this knowledge to classify a new sample.

Solution:

Sample Dataset:

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0

Python Code:

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.tree import DecisionTreeClassifier, export_graphviz

from sklearn import tree

from sklearn.preprocessing import StandardScaler, LabelEncoder

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score, confusion_matrix, plot_confusion_matrix

df = pd.read_csv('suv_data.csv')

df.head()

df.drop(['User ID'], axis=1, inplace = True)

le = LabelEncoder()

#converting gender column to numerical value

df['Gender'] = le.fit_transform(df['Gender'])
```

```
sc = StandardScaler()

#scailing the data

df.iloc[:,:3]  = sc.fit_transform(df.iloc[:,:3])

X = df.drop(['Purchased'], axis=1)

Y = df['Purchased']

train_x, test_x, train_y, test_y = train_test_split(X, Y, test_size = 0.3, random_state=99)

dt = DecisionTreeClassifier()

dt.fit(train_x, train_y)

y_pred  = dt.predict(test_x)

y_pred

results = pd.DataFrame({'Actual' : test_y, 'Predicted' : y_pred})

results

print("Accuracy Score",accuracy_score(test_y, y_pred)*100)

print("Confusion Matrix\n",confusion_matrix(test_y,y_pred))

fig=plt.figure(figsize=(28,20))

tree.plot_tree(dt, feature_names=train_x.columns, class_names=['0','1'], filled=True,
rounded=True, fontsize=12)
```

OUTPUT:

```
Accuracy Score 86.66666666666667

Confusion Matrix

 [[76  7]

 [ 9 28]]
```

Program 7. Write a program to build an ANN by implementing the Back Propagation Algorithm.

Solution:

Python Code:

```
import random

from math import exp

from random import seed

def initialize_network(n_inputs, n_hidden, n_outputs):

   network = list()

   hidden_layer = [{'weights':[random.uniform(-0.5,0.5) for i in range(n_inputs + 1)]} for i in
range(n_hidden)]

   network.append(hidden_layer)

   output_layer = [{'weights':[random.uniform(-0.5,0.5) for i in range(n_hidden + 1)]} for i in
range(n_outputs)]

   network.append(output_layer)

   i= 1

   print("\n The initialised Neural Network:\n")

   for layer in network:

      j=1

      for sub in layer:

         print("\n Layer[%d] Node[%d]:\n" %(i,j),sub)

         j=j+1

      i=i+1

   return network

def activate(weights, inputs):

   activation = weights[-1]

   for i in range(len(weights)-1):

      activation += weights[i] * inputs[i]
```

```
    return activation
def transfer(activation):
    return 1.0 / (1.0 + exp(-activation))
def forward_propagate(network, row):
    inputs = row
    for layer in network:
        new_inputs = []
        for neuron in layer:
            activation = activate(neuron['weights'], inputs)
            neuron['output'] = transfer(activation)
            new_inputs.append(neuron['output'])
        inputs = new_inputs
    return inputs
def transfer_derivative(output):
    return output * (1.0 - output)
def backward_propagate_error(network, expected):
    for i in reversed(range(len(network))):
        layer = network[i]
        errors = list()
          if i != len(network)-1:
            for j in range(len(layer)):
                error = 0.0
                for neuron in network[i + 1]:
                    error += (neuron['weights'][j] * neuron['delta'])
                errors.append(error)
        else:
```

```
            for j in range(len(layer)):
                neuron = layer[j]
                errors.append(expected[j] - neuron['output'])

        for j in range(len(layer)):
            neuron = layer[j]
            neuron['delta'] = errors[j] * transfer_derivative(neuron['output'])

def update_weights(network, row, l_rate):
    for i in range(len(network)):
        inputs = row[:-1]
        if i != 0:
            inputs = [neuron['output'] for neuron in network[i - 1]]
        for neuron in network[i]:
            for j in range(len(inputs)):
                neuron['weights'][j] += l_rate * neuron['delta'] * inputs[j]
            neuron['weights'][-1] += l_rate * neuron['delta']
def train_network(network, train, l_rate, n_epoch, n_outputs):
   print("\n Network Training Begins:\n")
    for epoch in range(n_epoch):
        sum_error = 0
        for row in train:
            outputs = forward_propagate(network, row)
            expected = [0 for i in range(n_outputs)]
            expected[row[-1]] = 1
            sum_error += sum([(expected[i]-outputs[i])**2 for i in range(len(expected))])
```

```
            backward_propagate_error(network, expected)
            update_weights(network, row, l_rate)
        print('>epoch=%d, lrate=%.3f, error=%.3f' % (epoch, l_rate, sum_error))
    print("\n Network Training Ends:\n")
seed(2)
dataset = [[2.7810836,2.550537003,0],
    [1.465489372,2.362125076,0],
    [3.396561688,4.400293529,0],
    [1.38807019,1.850220317,0],
    [3.06407232,3.005305973,0],
    [7.627531214,2.759262235,1],
    [5.332441248,2.088626775,1],
    [6.922596716,1.77106367,1],
    [8.675418651,-0.242068655,1],
    [7.673756466,3.508563011,1]]

print("\n The input Data Set :\n",dataset)
n_inputs = len(dataset[0]) - 1
print("\n Number of Inputs :\n",n_inputs)
n_outputs = len(set([row[-1] for row in dataset]))
print("\n Number of Outputs :\n",n_outputs)
network = initialize_network(n_inputs, 2, n_outputs)
train_network(network, dataset, 0.5, 20, n_outputs)
print("\n Final Neural Network :")
    i= 1
for layer in network:
```

```
    j=1
    for sub in layer:
        print("\n Layer[%d] Node[%d]:\n" %(i,j),sub)
        j=j+1
    i=i+1
```

OUTPUT:

```
 Final Neural Network :

 Layer[1] Node[1]:
 {'weights': [0.8642508164347664, -0.8497601716670761, -0.8668929014392035], 'output': 0.9295587965836384, 'delta': 0.005645382
825629247}

 Layer[1] Node[2]:
 {'weights': [-1.2934302410111027, 1.7109363237151511, 0.7125327507327331], 'output': 0.04760703296164143, 'delta': -0.00592855
9978815065}

 Layer[2] Node[1]:
 {'weights': [-1.3098359335096292, 2.16462207144596, -0.3079052288835877], 'output': 0.1989556395205846, 'delta': -0.0317080164
8036036}

 Layer[2] Node[2]:
 {'weights': [1.5506793402414165, -2.11315950446121, 0.1333585709422027], 'output': 0.8095042653312078, 'delta': 0.029375796661
413225}
```

Program 8. Write a program to build an ANN by implementing the Feed Forward Algorithm.

Solution:

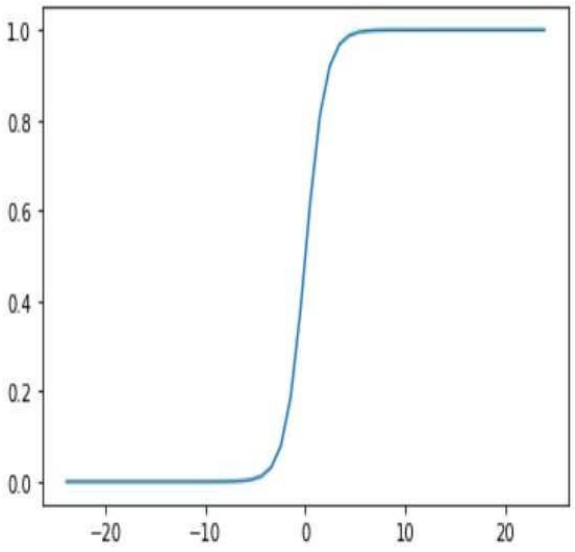
Python Code:

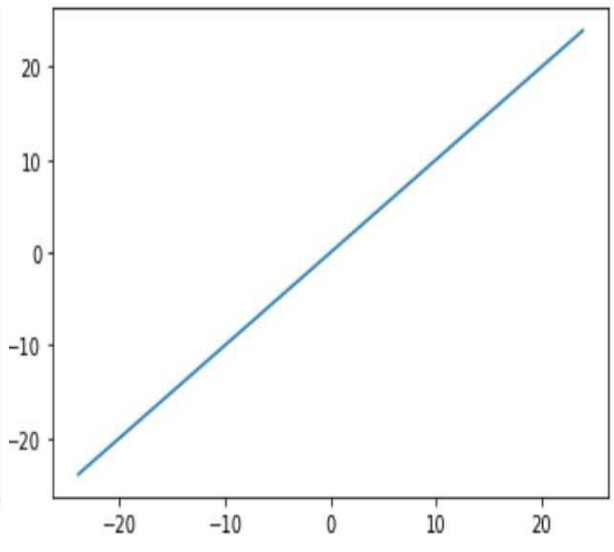
```
import matplotlib.pyplot as plt
import numpy as np
class nn():
    def __init_(self):
        self.bias=1
    def func(self,wt,ip):
        self.s=0
        for i in range(len(wt)):
```

```
            self.s+=wt[i]*ip[i]
        return self.s+self.bias
def Linear(x):
    return x
def sigmoid(x):
    return 1/(1+np.exp(-x))
wt=[0.1,0.5,1,2,2.6,1.5,1.3,2.2,0.9]
ip=[0.3,1.9,2.0,2.7,1.5,0.8,1.1,2,4]
n=nn()
print(n.func(wt,ip))
x=np.linspace(-n.func(wt,ip),n.func(wt,ip))
plt.plot(x,sigmoid(x))
plt.plot(x,Linear(x))
```

OUTPUT:

23.910000000000004





Program 9. Write a program for calculating the Best Association rule based on classification using min support and confidence threshold.

Solution: Sample Dataset:

	products
0	MILK,BREAD,BISCUIT
1	BREAD,MILK,BISCUIT,CORNFLAKES
2	BREAD,TEA,BOURNVITA
3	JAM,MAGGI,BREAD,MILK
4	MAGGI,TEA,BISCUIT

Python Code:

```
import pandas as pd

import numpy as np

from mlxtend.frequent_patterns import apriori, association_rules

df = pd.read_csv('GroceryStoreDataSet.csv', names = ['products'], sep = ',')

df.head()

data = list(df["products"].apply(lambda x:x.split(",") ))

data

from mlxtend.preprocessing import TransactionEncoder

a = TransactionEncoder()

a_data = a.fit(data).transform(data)

df = pd.DataFrame(a_data,columns=a.columns_)

df = df.replace(False,0)

df

df = apriori(df, min_support = 0.2, use_colnames = True, verbose = 1)

df

df_ar = association_rules(df, metric = "confidence", min_threshold = 0.6)

df_ar
```

OUTPUT:

antecedents	consequents	antecedent support	consequent support	support	confidence
(MILK)	(BREAD)	0.25	0.65	0.2	0.800000
(SUGER)	(BREAD)	0.30	0.65	0.2	0.666667
(CORNFLAKES)	(COFFEE)	0.30	0.40	0.2	0.666667
(SUGER)	(COFFEE)	0.30	0.40	0.2	0.666667
(MAGGI)	(TEA)	0.25	0.35	0.2	0.800000

Program 10. Write a program to perform basic operations of TensorFlow.

Solution: import tensorflow as tf

```
tensor_a = tf.constant([[4,6]])
tensor_b = tf.constant([[2, 1]])
tensor_add = tf.add(tensor_a, tensor_b)    # addition
tensor_sub=tf.subtract(tensor_a,tensor_b)    #subtraction
tensor_mult=tf.multiply(tensor_a,tensor_b)    #multiplication
tensor_div=tf.divide(tensor_a,tensor_b)        #division
print(tensor_add)
print(tensor_sub)
print(tensor_mult)
print(tensor_div)
m_shape = tf.constant([ [10, 11],[12, 13], [14, 15] ] )        # Shape of tensor
 m_shape.shape
```

OUTPUT: tf.Tensor([[6 7]], shape=(1, 2), dtype=int32)

tf.Tensor([[2 5]], shape=(1, 2), dtype=int32)

tf.Tensor([[8 6]], shape=(1, 2), dtype=int32)

tf.Tensor([[2. 6.]], shape=(1, 2), dtype=float64)

TensorShape([3, 2])