**Mini Project 2**

Author:

|  |  |
| --- | --- |
| **Abhirup Ranjan** | **110091866** |

Submitted To:

|  |
| --- |
| **Professor** |
| **Dr. Adel Abusitta** |

**Table of Contents**

[1. Problem Statement 1 2](#_Toc138364528)

[1.1 Python Code 2](#_Toc138364529)

[1.2 Terminal Window Screenshot 4](#_Toc138364530)

# **Problem Statement**

Write an artificial neural network program for movie recommendation that takes feedback from three or more friends on a scale of 1 to 5 and decides whether to attend the movie or not. Your program should determine the accuracy of its recommendations.

For the training dataset, you can either create a fake dataset or use an already available one.

# **1.1 Python Code**

import numpy as np #provides support for arrays and mathematical operations.

# Creating a fake dataset

# Each row represents a movie, and each column represents a friend's feedback

# The last column represents whether the movie was attended (1) or not (0)

trainingDataset = np.array([

[5, 4, 3, 1],

[1, 2, 5, 0],

[3, 2, 1, 0],

[4, 5, 2, 1],

[3, 3, 2, 0],

[1, 3, 4, 0],

[2, 4, 5, 1],

[4, 3, 2, 0],

[5, 1, 3, 1],

[2, 3, 5, 1],

[3, 4, 1, 0],

[0, 0, 0, 0],

[1, 1, 1, 0],

[5, 5, 5, 1]

])

# Define the neural network architecture

inputSize = trainingDataset.shape[1] - 1 # Number of features (feedback columns)

outputSize = 2 # Binary classification (attend[1] or not[0])

hiddenSize = 10 # Number of neurons in the hidden layer

# Initialize random weights for the neural network

np.random.seed(0)

#represents the weights between the input layer and the hidden layer

weights\_input\_hidden = 2 \* np.random.random((inputSize, hiddenSize)) - 1

#represents the weights between the hidden layer and the output layer

weights\_hidden\_output = 2 \* np.random.random((hiddenSize, outputSize)) - 1

# Sigmoid activation function this maps the output to a value between 0 and 1

def sigmoid(a):

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

# Derivative of the sigmoid function used in the backpropagation

def sigmoidDerivative(i):

return i \* (1 - i)

# Train the neural network using backpropagation

epochs = 10000

learningRate = 0.1

for epoch in range(epochs):

# Forward propagation

inputLayer = trainingDataset[:, :-1]

hiddenLayer = sigmoid(np.dot(inputLayer, weights\_input\_hidden))

outputLayer = sigmoid(np.dot(hiddenLayer, weights\_hidden\_output))

# Backpropagation

outputError = trainingDataset[:, -1].reshape(-1, 1) - outputLayer

outputDelta = outputError \* sigmoidDerivative(outputLayer)

hiddenError = outputDelta.dot(weights\_hidden\_output.T)

hiddenDelta = hiddenError \* sigmoidDerivative(hiddenLayer)

# Update weights

weights\_hidden\_output = weights\_hidden\_output + hiddenLayer.T.dot(outputDelta) \* learningRate

weights\_input\_hidden = weights\_input\_hidden + inputLayer.T.dot(hiddenDelta) \* learningRate

# Function to predict movie attendance based on feedback

def predictMovieAttendance(feedback):

hiddenOutput = sigmoid(np.dot(feedback, weights\_input\_hidden))

predictedOutput = sigmoid(np.dot(hiddenOutput, weights\_hidden\_output))

if predictedOutput[0][0] >= 0.5:

return "You can ATTEND this movie!!!"

else:

return "It is recommended to NOT attend the movie!!!"

# Test the accuracy of the model

testData = np.array([

[1, 1, 1, 0], # Should not attend (0)

[4, 4, 4, 1] # Should attend (1)

])

#iterates over the test cases, compares the predicted attendance with the actual attendance

# and calculates the accuracy.

correctPredictions = 0

for testCase in testData:

feedback = testCase[:-1].reshape(1, -1)

actualAttendance = testCase[-1]

predictedAttendance = 1 if predictMovieAttendance(feedback) == "You can ATTEND this movie!!!" else 0

if actualAttendance == predictedAttendance:

correctPredictions = correctPredictions + 1

accuracy = (correctPredictions / len(testData)) \* 100

print("Accuracy: {:.2f}%".format(accuracy))

# Make predictions for new examples

newFeedback = np.array([[3, 3, 5]]) # Example feedback

prediction = predictMovieAttendance(newFeedback)

print("Prediction for new feedback:", prediction)

# **1.2 Terminal Window Screenshot**

A screen shot of a computer

Description automatically generated with medium confidence

**Figure 1.1**