(BS – CS-04B)

ARTIFICIAL INTELLIGENCE LAB

PROJECT REPORT

Submitted To:
Ms. Tayyaba

Submitted By:
Name: *Zimal Babar*
Reg No: *230201074*

Department of Computer Science,
Institute Of Space Technology, Islamabad.

DATE [24th January 2025]

DATASET: Employee Resignation Data Check

DESCRIPTION:

This dataset provides information on employees, including their education level, joining year, city, payment tier, age, gender, work experience, and whether they have ever been benched. The primary goal is to predict the target variable, "LeaveOrNot," which indicates whether an employee has left the organization (1) or not (0). With features such as employee's job-related details, and work history, this dataset is ideal for classification tasks. Techniques like K-Nearest Neighbors (KNN), Gaussian Naive Bayes, Multilayer Perceptron (MLP), and Neural Networks can be applied to analyze patterns and predict employee status.

CLASSIFICATION TASKS:

1. K Nearest Neighbors Classifier (KNN)

Description:

KNN is an instance-based learning algorithm that makes predictions by finding the majority class among the k nearest neighbors of a data point in the feature space.

Parameters:

Number of neighbors: 7

Accuracy: 0.8077

This indicates that the feature space is well-structured, and KNN effectively captures relationships between data points.

CODE:

```
# Implement KNearestNeighbor classifier
classifier = KNeighborsClassifier(n_neighbors=7)
KNeighborsClassifier()

#Training model
classifier.fit(X_train, y_train)

# Evaluating the model's accuracy on the test set
y_prediction = classifier.score(X_test,y_test)
print("Test data accuracy using KNN: ",y_prediction)
```

OUTPUT:

```
# Implement KNearestNeighbor classifier
classifier = KNeighborsClassifier(n_neighbors=7)
KNeighborsClassifier()

#Training model
classifier.fit(X_train, y_train)

# Evaluating the model's accuracy on the test set
y_prediction = classifier.score(X_test,y_test)
print("Test data accuracy using KNN: ",y_prediction)
```

[32] ✓ 0.0s Python

```
Test data accuracy using KNN:  0.807733619763695
```

2. GaussianNB Classifier

Description:
GNB assumes that features are independent (Naive assumption) and follow a Gaussian (normal) distribution. It calculates probabilities for each class and selects the one with the highest likelihood.

Accuracy: 0.7175
Lower than KNN, possibly because features may have complex relationships not captured by this model.

Code:

```
# Implement Gaussian classifier
gclassifier = GaussianNB()

#Training model
gclassifier.fit(X_train, y_train)

# Evaluating the model's accuracy on the test set
y_prediction = gclassifier.score(X_test,y_test)
print("Test data accuracy using gaussian: ",y_prediction)
```

Output:

```
# Implement Gaussian classifier
gclassifier = GaussianNB()

#Training model
gclassifier.fit(X_train, y_train)

# Evaluating the model's accuracy on the test set
y_prediction = gclassifier.score(X_test,y_test) # test data 20% is used now for accuracy check for n=7
print("Test data accuracy using gaussian: ",y_prediction)
```

[33] ✓ 0.0s Python

```
Test data accuracy using gaussian:  0.7175080558539205
```

3. MLP CLASSIFIER

Description:

MLP is a feedforward artificial neural network with one or more hidden layers. It can learn complex patterns and relationships in the data.

Parameters:

- No of hidden layers: 10
- No of neurons: 40
- Learning rate: 0.001
- Optimizer: adam

Accuracy: 0.78

Slightly lower than KNN, likely because:

- Insufficient training or suboptimal hyperparameters.

Code:

```
# Implement MLP classifier
mlp = MLPClassifier(hidden_layer_sizes=(40,10), activation = 'tanh',
learning_rate_init=0.001, solver= 'adam', max_iter=1000, random_state=100)

# Train the model on the training data
mlp.fit(X_train, y_train)

# Making predictions on the test data
y_pred = mlp.predict(X_test)

# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
```

Output:

4. Custom Neural Network

Description: A fully connected neural network implemented from scratch using the sigmoid activation function and optimized using the Adam optimizer.

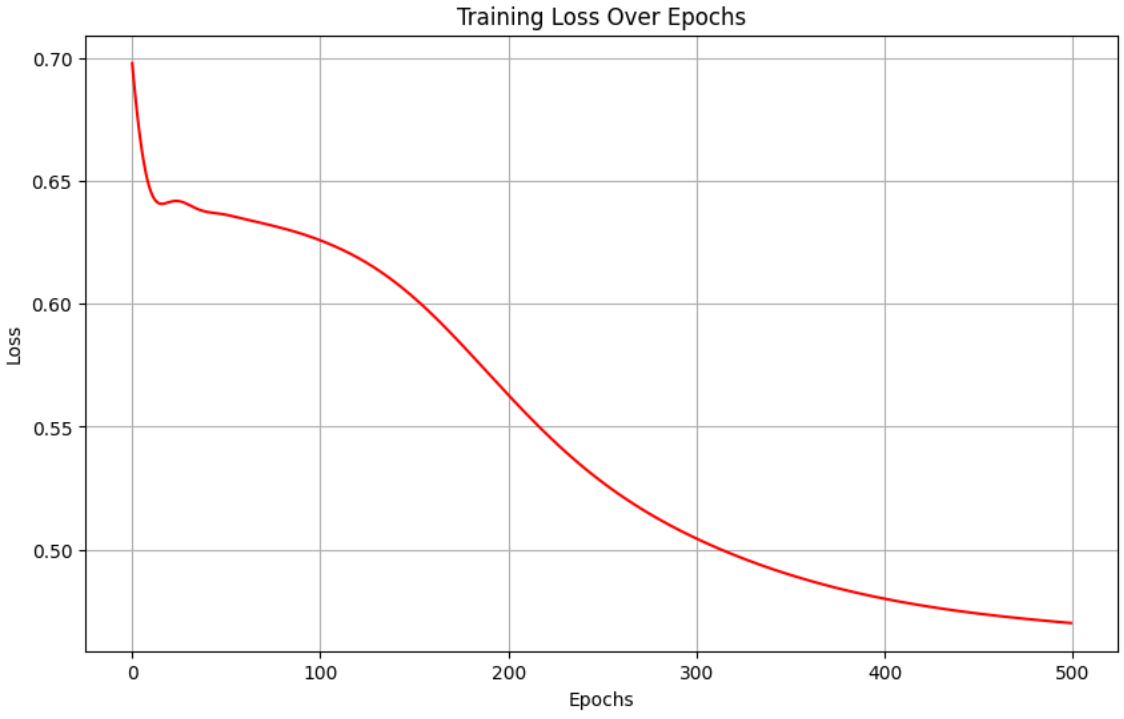
Architecture:

Input Layer: Features count
Hidden Layer: 50 neurons
Output Layer: Binary classification
Epochs: 500
Learning Rate: 0.001

Performance:

Training Accuracy: 82.0%
Test Accuracy: 79.0%
Observations:
The model shows excellent performance, suggesting it effectively generalizes to unseen data. The custom implementation also provides flexibility for experimentation but requires careful tuning to maintain performance.

Training Loss:



- The training loss starts at **~0.70** and drops consistently over time.
- Initially, there is a slight increase before it steadily declines.
- The loss shows a clear downward trend suggesting stable training.
- By epoch 500, the loss is below **0.50**, indicating that the model is still improving its fit on the training data.

OUTPUT:

```
Epoch 0, Loss: 0.6977, Training Accuracy: 0.66
Epoch 50, Loss: 0.6361, Training Accuracy: 0.66
Epoch 100, Loss: 0.6258, Training Accuracy: 0.66
Epoch 150, Loss: 0.6023, Training Accuracy: 0.66
Epoch 200, Loss: 0.5629, Training Accuracy: 0.71
Epoch 250, Loss: 0.5274, Training Accuracy: 0.76
Epoch 300, Loss: 0.5045, Training Accuracy: 0.79
Epoch 350, Loss: 0.4897, Training Accuracy: 0.80
Epoch 400, Loss: 0.4801, Training Accuracy: 0.81
Epoch 450, Loss: 0.4740, Training Accuracy: 0.81
Train Accuracy: 0.82
Test Accuracy: 0.79
```