3w3o1la4i

February 20, 2025

Name : Sakshi Dube Prn :202201040155 Div :C (computer) Batch :T3(mdm)

[ ]:

```

```

Logistic Regression implementation from scratch

[ ]:

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

# Load dataset
df = pd.read_csv("/age_predictions_cleaned.csv")

# Split data into features and target
X = df.iloc[:, :-1].values  # All columns except the last one
y = df.iloc[:, -1].values   # Last column as target

# Normalize features
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Sigmoid function
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

# Compute cost function
def compute_cost(X, y, weights):
    m = len(y)
    h = sigmoid(X @ weights)
    cost = (-1/m) * (y.T @ np.log(h) + (1 - y).T @ np.log(1 - h))
    return cost

# Gradient Descent
```

```python
def gradient_descent(X, y, weights, lr, epochs):
    m = len(y)
    for _ in range(epochs):
        h = sigmoid(X @ weights)
        gradient = (1/m) * X.T @ (h - y)
        weights -= lr * gradient
    return weights

# Add bias term
X_train_bias = np.c_[np.ones((X_train.shape[0], 1)), X_train]
X_test_bias = np.c_[np.ones((X_test.shape[0], 1)), X_test]

# Initialize weights
weights = np.zeros(X_train_bias.shape[1])

# Train model
weights = gradient_descent(X_train_bias, y_train, weights, lr=0.01, epochs=1000)

# Predictions
y_pred = sigmoid(X_test_bias @ weights) >= 0.5

# Accuracy
accuracy = np.mean(y_pred == y_test)
print(f"Accuracy: {accuracy:.4f}")
```

```
Accuracy: 0.7035
```

Logistic Regression implementation using library

[ ]:
```python
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Load dataset
df = pd.read_csv("/age_predictions_cleaned.csv")

# Split data into features and target
X = df.iloc[:, :-1].values  # All columns except the last one
y = df.iloc[:, -1].values   # Last column as target

# Normalize features
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Split dataset
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Train Logistic Regression model
model = LogisticRegression()
model.fit(X_train, y_train)

# Predictions
y_pred = model.predict(X_test)

# Accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")
```

```
Accuracy: 0.7007
```

[ ]:
```
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report,␣
 ↪confusion_matrix

# Load dataset
df = pd.read_csv("/age_predictions_cleaned.csv")

# Split data into features and target
X = df.iloc[:, :-1].values  # All columns except the last one
y = df.iloc[:, -1].values   # Last column as target

# Normalize features
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Train Logistic Regression model
model = LogisticRegression()
model.fit(X_train, y_train)

# Predictions
y_pred = model.predict(X_test)

# Accuracy
```

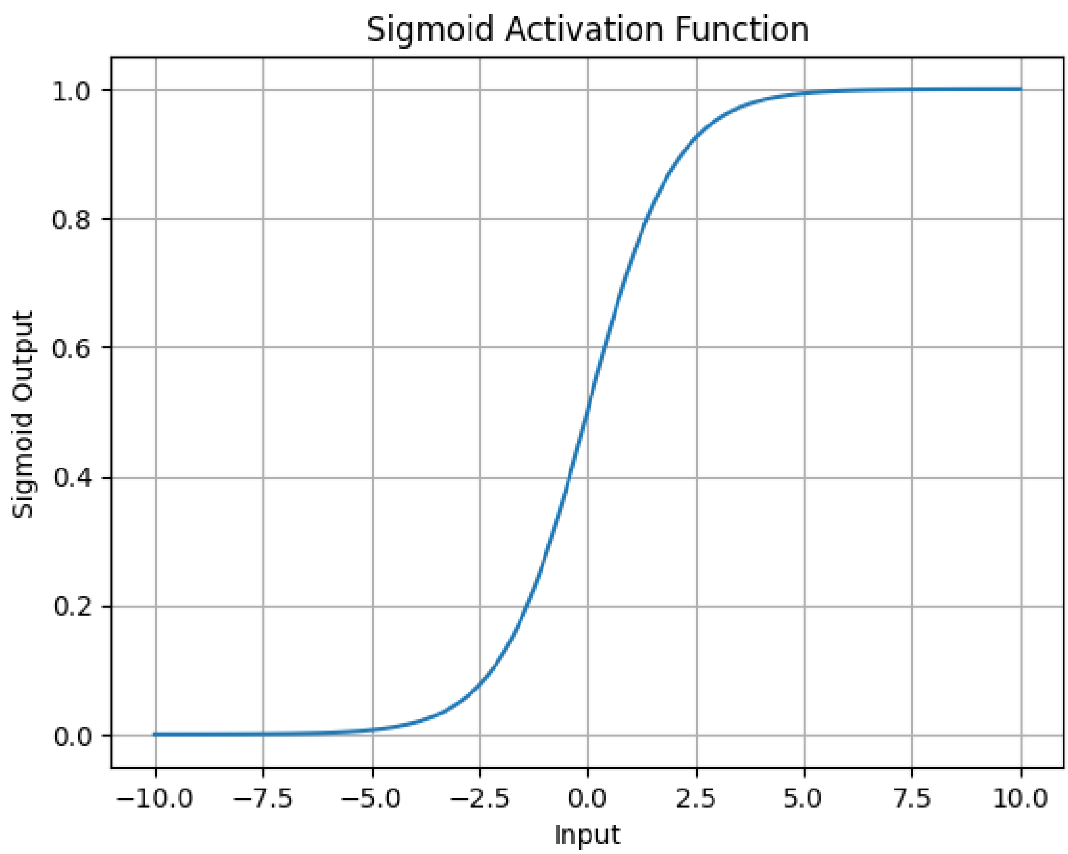
```
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")

# Confusion Matrix and Classification Report
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```

```
Accuracy: 0.7007
Confusion Matrix:
 [[257 106]
 [105 237]]
Classification Report:
              precision    recall  f1-score   support

           0       0.71      0.71      0.71       363
           1       0.69      0.69      0.69       342

    accuracy                           0.70       705
   macro avg       0.70      0.70      0.70       705
weighted avg       0.70      0.70      0.70       705
```

Activation Function

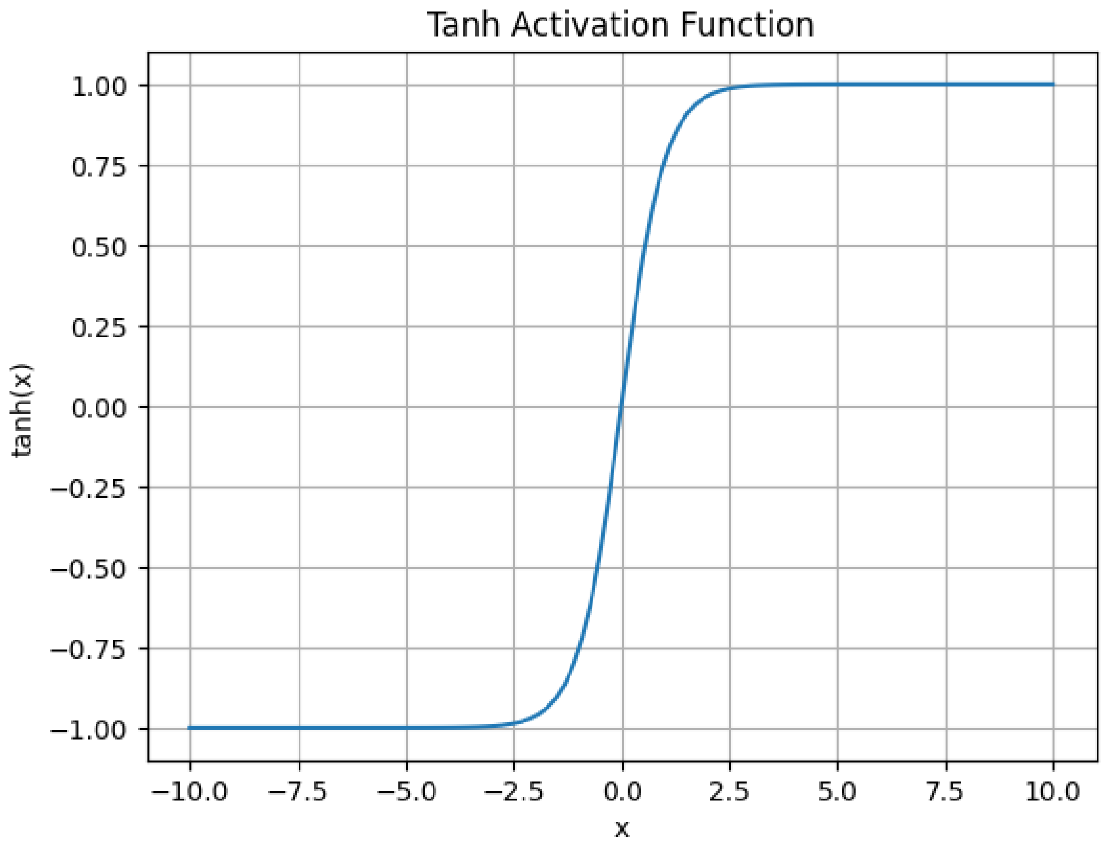
Sigmoid

[ ]:
```
import matplotlib.pyplot as plt

def plot_sigmoid():
    x = np.linspace(-10, 10, 100)
    y = sigmoid(x)
    plt.plot(x, y)
    plt.xlabel('Input')
    plt.ylabel('Sigmoid Output')
    plt.title('Sigmoid Activation Function')
    plt.grid(True)
    plt.show()

plot_sigmoid()
```



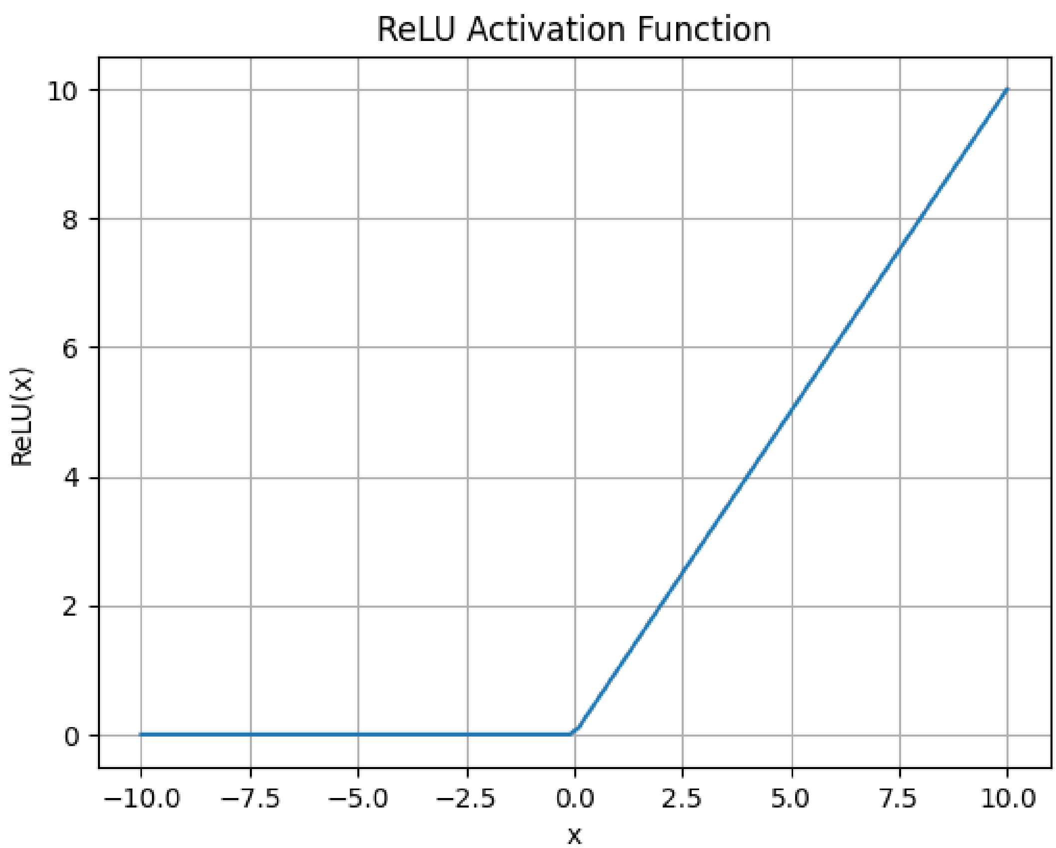
tanh

[ ]:
```
def tanh(x):
    return np.tanh(x)

x = np.linspace(-10, 10, 100)
plt.plot(x, tanh(x))
plt.title("Tanh Activation Function")
plt.xlabel("x")
plt.ylabel("tanh(x)")
plt.grid(True)
plt.show()
```



ReLU

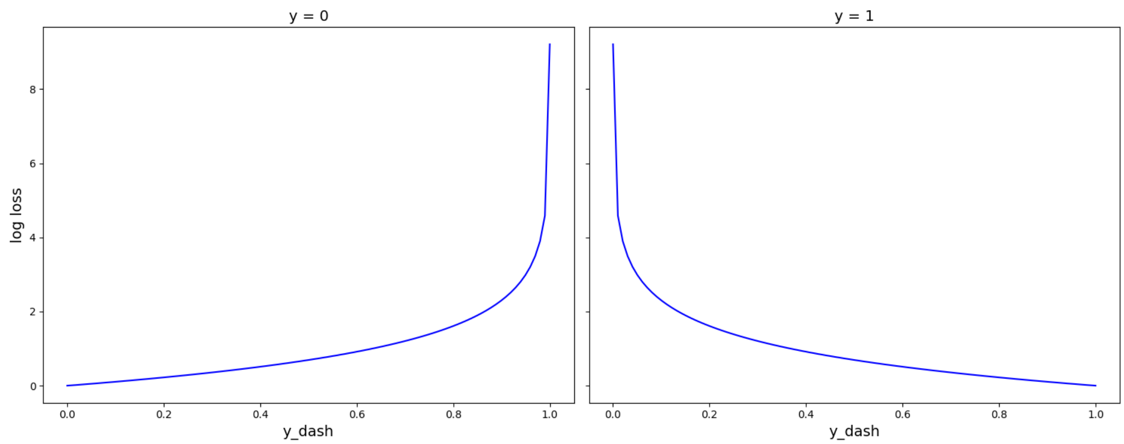
[ ]:
```python
def relu(x):
    return np.maximum(0, x)

x = np.linspace(-10, 10, 100)
plt.plot(x, relu(x))
plt.title("ReLU Activation Function")
plt.xlabel("x")
plt.ylabel("ReLU(x)")
plt.grid(True)
plt.show()
```



Log loss funtion

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt

def log_loss(y, y_dash):
    """Computes log loss for inputs true value (0 or 1) and predicted value␣
 ↪(between 0 and 1)."""
    loss = - (y * np.log(y_dash)) - ((1 - y) * np.log(1 - y_dash))
    return loss

# Log loss for y = 0 and y = 1
fig, ax = plt.subplots(1, 2, figsize=(15, 6), sharex=True, sharey=True)
y_dash = np.linspace(0.0001, 0.9999, 100)

# Plot for y = 0
ax[0].plot(y_dash, [log_loss(0, yd) for yd in y_dash], color='blue')
ax[0].set_title("y = 0", fontsize=14)
ax[0].set_xlabel("y_dash", fontsize=14)
ax[0].set_ylabel("log loss", fontsize=14)
```

```
# Plot for y = 1
ax[1].plot(y_dash, [log_loss(1, yd) for yd in y_dash], color='blue')
ax[1].set_title("y = 1", fontsize=14)
ax[1].set_xlabel("y_dash", fontsize=14)

plt.tight_layout()
plt.show()
```

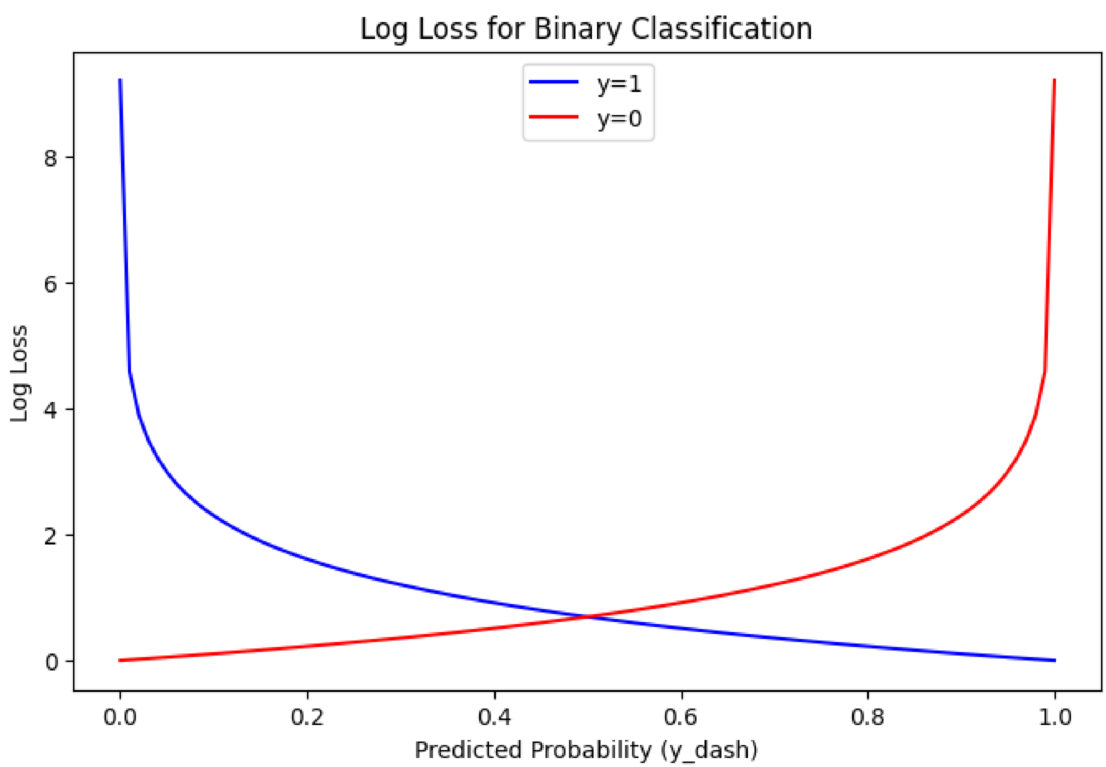


[ ]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

def log_loss(y_true, y_pred):
    """Computes log loss for an array of true values (0 or 1) and predicted␣
 ↪probabilities (between 0 and 1)."""
    epsilon = 1e-15  # To avoid log(0)
    y_pred = np.clip(y_pred, epsilon, 1 - epsilon)
    loss = -np.mean(y_true * np.log(y_pred) + (1 - y_true) * np.log(1 - y_pred))
    return loss

# Load the dataset
df = pd.read_csv("/content/age_predictions_cleaned.csv")

# Assuming the last column is the true label (0 or 1) and the second last␣
 ↪column is the predicted probability
y_true = df.iloc[:, -1].values  # Target labels (0 or 1)
y_pred = df.iloc[:, -2].values  # Predicted probabilities (between 0 and 1)

# Compute log loss
loss_value = log_loss(y_true, y_pred)
```

```
print(f"Log Loss: {loss_value}")

# Plot log loss curve for reference
y_dash = np.linspace(0.0001, 0.9999, 100)

plt.figure(figsize=(8, 5))
plt.plot(y_dash, [-np.log(yd) for yd in y_dash], label="y=1", color='blue')
plt.plot(y_dash, [-np.log(1 - yd) for yd in y_dash], label="y=0", color='red')
plt.xlabel("Predicted Probability (y_dash)")
plt.ylabel("Log Loss")
plt.title("Log Loss for Binary Classification")
plt.legend()
plt.show()
```

```
Log Loss: 17.26002955317878
```



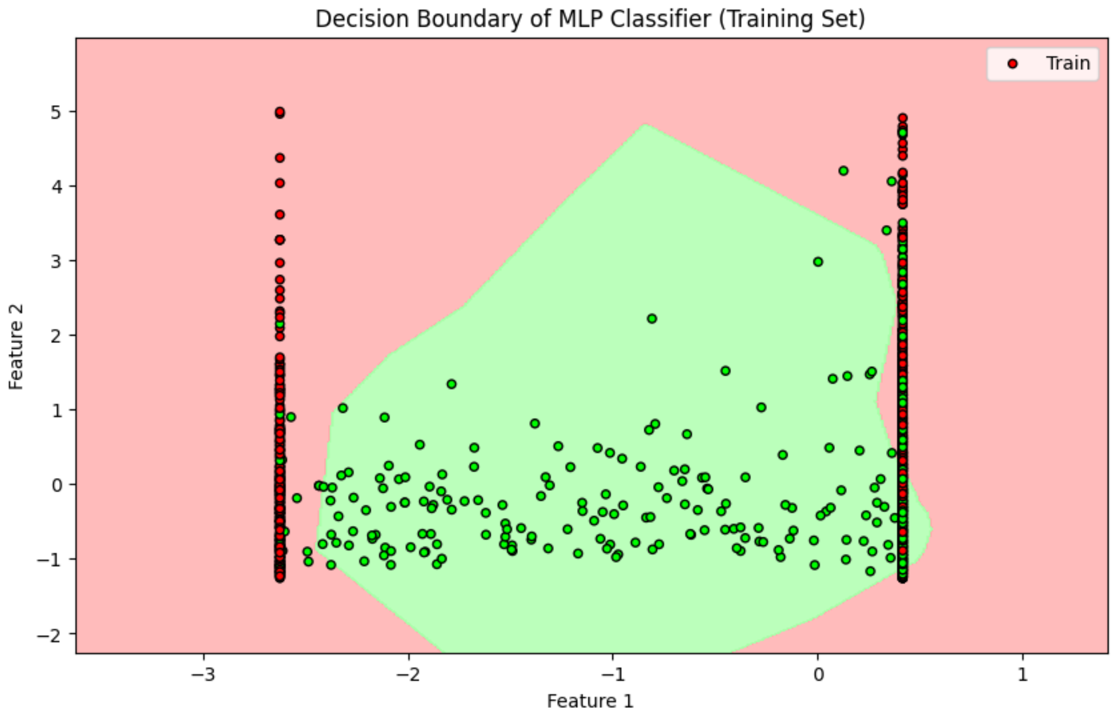
0.1 Sklearn Implementation of MultiLayer Perceptron(MLP)

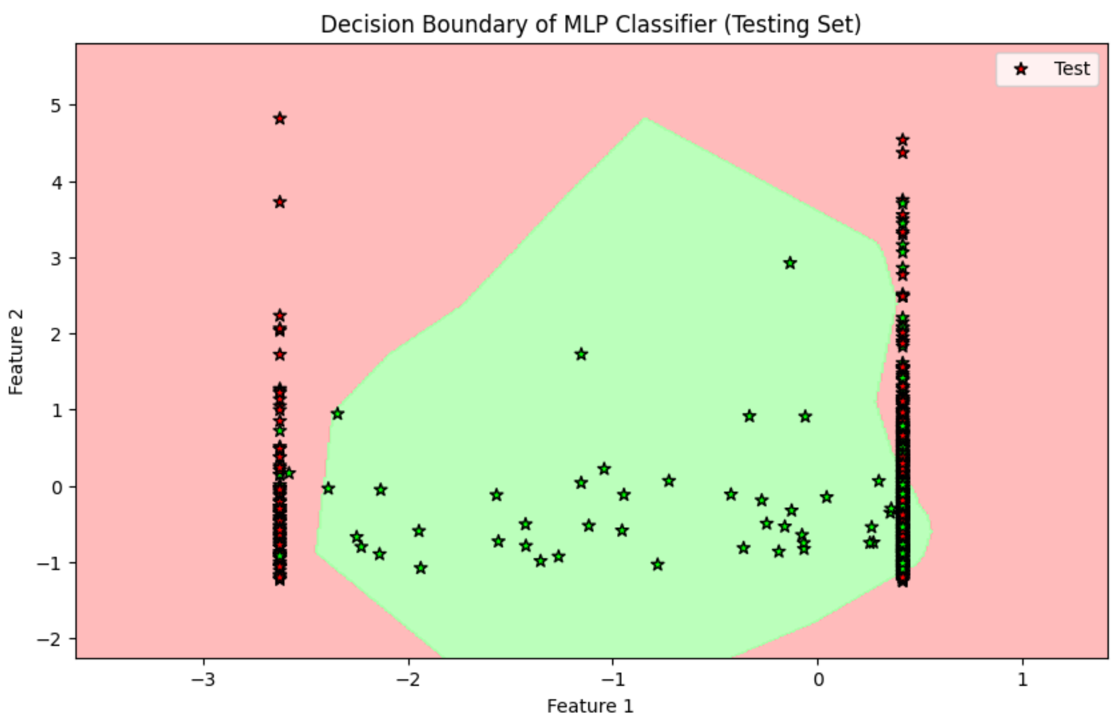
[ ]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, classification_report
```

```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

# Load the dataset
df = pd.read_csv("/age_predictions_cleaned.csv")  # Ensure the correct path

# Use first two features for visualization (modify if necessary)
X = df.iloc[:, :-1].values[:, :2]  # Select first two features
y = df.iloc[:, -1].values  # Target labels

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Define and train the MLP classifier
mlp = MLPClassifier(hidden_layer_sizes=(10, 10), max_iter=1000, random_state=42)
mlp.fit(X_train, y_train)

# Make predictions
y_pred = mlp.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
print("\nClassification Report:\n", classification_report(y_test, y_pred))

# Plot decision boundaries for training set
x_min, x_max = X_train[:, 0].min() - 1, X_train[:, 0].max() + 1
y_min, y_max = X_train[:, 1].min() - 1, X_train[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01), np.arange(y_min, y_max, 0.
 ↪01))
Z_train = mlp.predict(np.c_[xx.ravel(), yy.ravel()]).reshape(xx.shape)

# Define color maps
cmap_light = ListedColormap(['#FFAAAA', '#AAFFAA'])
cmap_bold = ListedColormap(['#FF0000', '#00FF00'])
plt.figure(figsize=(10, 6))
plt.contourf(xx, yy, Z_train, alpha=0.8, cmap=cmap_light)
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap=cmap_bold,␣
 ↪edgecolor='k', s=20, label='Train')
plt.title("Decision Boundary of MLP Classifier (Training Set)")
```

```python
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.legend()
plt.show()

# Plot decision boundaries for testing set
x_min, x_max = X_test[:, 0].min() - 1, X_test[:, 0].max() + 1
y_min, y_max = X_test[:, 1].min() - 1, X_test[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01), np.arange(y_min, y_max, 0.
↪01))
Z_test = mlp.predict(np.c_[xx.ravel(), yy.ravel()]).reshape(xx.shape)

plt.figure(figsize=(10, 6))
plt.contourf(xx, yy, Z_test, alpha=0.8, cmap=cmap_light)
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap=cmap_bold,␣
↪edgecolor='k', s=50, label='Test', marker='*')
plt.title("Decision Boundary of MLP Classifier (Testing Set)")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.legend()
plt.show()
```

```
Accuracy: 0.6695035460992907

Classification Report:
              precision    recall  f1-score   support

           0       0.71      0.60      0.65       363
           1       0.64      0.75      0.69       342

    accuracy                           0.67       705
   macro avg       0.67      0.67      0.67       705
weighted avg       0.68      0.67      0.67       705
```





0.2 Keras Implementation of MultiLayer Perceptron(MLP)

[ ]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Dense
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, ConfusionMatrixDisplay

# Load the dataset
df = pd.read_csv("/age_predictions_cleaned.csv")  # Ensure the correct path

# Select the features and target (assuming last column is target)
X = df.iloc[:, :-1].values  # Features (all columns except last)
y = df.iloc[:, -1].values  # Target (last column)

# Split the dataset into training (80%) and testing (20%) sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Standardize the data (helps with convergence and performance)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)  # Fit the scaler on training data and␣
 ↪transform it
X_test = scaler.transform(X_test)  # Transform the testing data using the same␣
 ↪scaler

# Step 2: Build the ANN model
model = Sequential([
    Dense(32, activation='relu', input_dim=X_train.shape[1]),  # Hidden layer␣
 ↪with 32 neurons and ReLU activation
    Dense(16, activation='relu'),  # Another hidden layer with 16 neurons and␣
 ↪ReLU activation
    Dense(1, activation='sigmoid')  # Output layer with 1 neuron and sigmoid␣
 ↪activation for binary classification
])

# Step 3: Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy',␣
 ↪metrics=['accuracy'])

# Step 4: Train the model
history = model.fit(X_train, y_train, epochs=20, batch_size=32,␣
 ↪validation_split=0.2, verbose=1)

# Step 5: Evaluate the model
```

```python
loss, accuracy = model.evaluate(X_test, y_test)
print(f"\nTest Loss: {loss:.4f}")
print(f"Test Accuracy: {accuracy:.4f}")

# Step 6: Generate predictions
y_pred = (model.predict(X_test) > 0.5).astype(int)

# Step 7: Classification Report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Step 8: Visualize confusion matrix
ConfusionMatrixDisplay.from_predictions(y_test, y_pred)
plt.show()
```

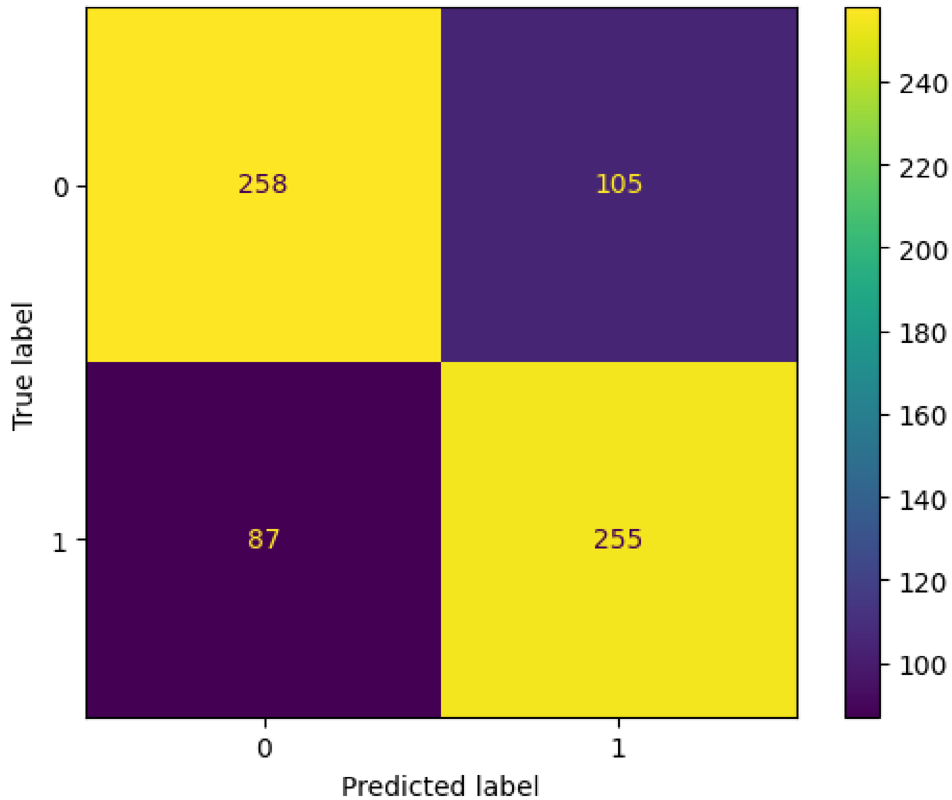
```
Epoch 1/20
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When
using Sequential models, prefer using an `Input(shape)` object as the first
layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
71/71            2s 6ms/step -
accuracy: 0.5385 - loss: 0.6911 - val_accuracy: 0.6897 - val_loss: 0.6074
Epoch 2/20
71/71            0s 3ms/step -
accuracy: 0.7071 - loss: 0.5924 - val_accuracy: 0.6968 - val_loss: 0.5816
Epoch 3/20
71/71            0s 3ms/step -
accuracy: 0.7086 - loss: 0.5674 - val_accuracy: 0.6950 - val_loss: 0.5796
Epoch 4/20
71/71            0s 3ms/step -
accuracy: 0.7296 - loss: 0.5470 - val_accuracy: 0.7057 - val_loss: 0.5786
Epoch 5/20
71/71            0s 3ms/step -
accuracy: 0.7077 - loss: 0.5587 - val_accuracy: 0.7039 - val_loss: 0.5783
Epoch 6/20
71/71            0s 3ms/step -
accuracy: 0.7158 - loss: 0.5502 - val_accuracy: 0.7039 - val_loss: 0.5747
Epoch 7/20
71/71            0s 3ms/step -
accuracy: 0.7072 - loss: 0.5595 - val_accuracy: 0.7074 - val_loss: 0.5743
Epoch 8/20
71/71            0s 3ms/step -
accuracy: 0.7155 - loss: 0.5503 - val_accuracy: 0.7074 - val_loss: 0.5728
Epoch 9/20
71/71            0s 3ms/step -
```

```
accuracy: 0.7371 - loss: 0.5387 - val_accuracy: 0.7145 - val_loss: 0.5740
Epoch 10/20
71/71            0s 3ms/step -
accuracy: 0.7317 - loss: 0.5439 - val_accuracy: 0.7128 - val_loss: 0.5697
Epoch 11/20
71/71            0s 3ms/step -
accuracy: 0.7120 - loss: 0.5476 - val_accuracy: 0.7074 - val_loss: 0.5681
Epoch 12/20
71/71            0s 4ms/step -
accuracy: 0.7307 - loss: 0.5395 - val_accuracy: 0.7199 - val_loss: 0.5675
Epoch 13/20
71/71            0s 3ms/step -
accuracy: 0.7221 - loss: 0.5414 - val_accuracy: 0.7057 - val_loss: 0.5685
Epoch 14/20
71/71            0s 3ms/step -
accuracy: 0.7237 - loss: 0.5410 - val_accuracy: 0.7057 - val_loss: 0.5680
Epoch 15/20
71/71            0s 4ms/step -
accuracy: 0.7332 - loss: 0.5262 - val_accuracy: 0.7270 - val_loss: 0.5607
Epoch 16/20
71/71            0s 4ms/step -
accuracy: 0.7345 - loss: 0.5353 - val_accuracy: 0.7234 - val_loss: 0.5618
Epoch 17/20
71/71            1s 6ms/step -
accuracy: 0.7268 - loss: 0.5333 - val_accuracy: 0.7181 - val_loss: 0.5620
Epoch 18/20
71/71            1s 6ms/step -
accuracy: 0.7380 - loss: 0.5207 - val_accuracy: 0.7305 - val_loss: 0.5586
Epoch 19/20
71/71            0s 4ms/step -
accuracy: 0.7292 - loss: 0.5220 - val_accuracy: 0.7163 - val_loss: 0.5583
Epoch 20/20
71/71            1s 6ms/step -
accuracy: 0.7330 - loss: 0.5345 - val_accuracy: 0.7287 - val_loss: 0.5562
23/23            0s 5ms/step -
accuracy: 0.7231 - loss: 0.5621

Test Loss: 0.5551
Test Accuracy: 0.7277
23/23            0s 5ms/step

Classification Report:
              precision    recall  f1-score   support

           0       0.75      0.71      0.73       363
           1       0.71      0.75      0.73       342

    accuracy                           0.73       705
```

```
   macro avg       0.73      0.73      0.73       705
weighted avg       0.73      0.73      0.73       705
```



0.3 Backward Propogation from Sratch

[ ]:

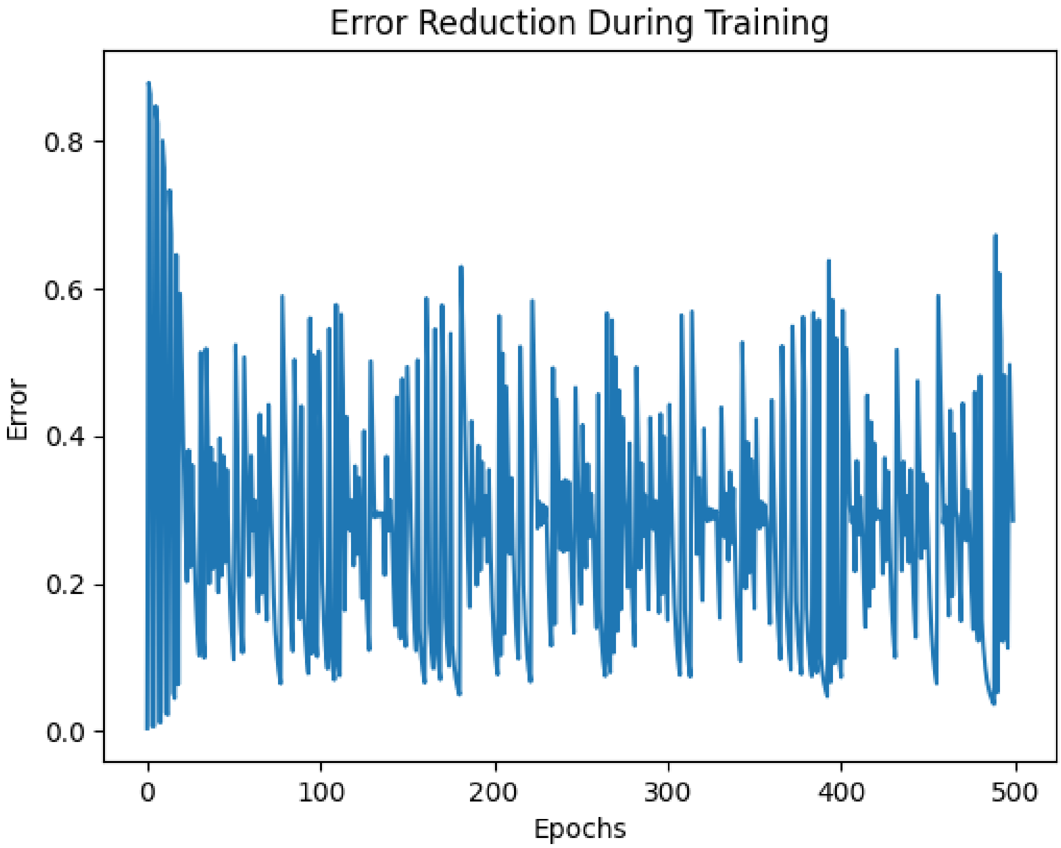
```

```

[ ]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math

# Load the dataset
df = pd.read_csv("/age_predictions_cleaned.csv")  # Ensure the correct path

# Select the features and target (assuming last column is target)
X = df.iloc[:, :-1].values  # Features (all columns except last)
y = df.iloc[:, -1].values  # Target (last column)

# Define network architecture
input_size = X.shape[1]  # Number of input features
```

```
hidden_size = 4  # Number of hidden layer neurons
output_size = 1  # Assuming binary classification

# Sigmoid Activation Function
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

# Derivative of Sigmoid Function
def sigmoid_derivative(x):
    return x * (1 - x)

# Feedforward function
def feed_forward(b1, b2, w1, w2, x):
    hidden = sigmoid(np.dot(w1, x) + b1)
    output = sigmoid(np.dot(w2, hidden) + b2)
    return hidden, output

# Error Calculation
def find_error(output, desired):
    return np.mean((output - desired) ** 2)

# Backpropagation
def back_propagate(w1, w2, b1, b2, hidden, output, desired, x, alpha):
    output_error = (output - desired) * sigmoid_derivative(output)
    hidden_error = np.dot(w2.T, output_error) * sigmoid_derivative(hidden)

    w2 -= alpha * np.outer(output_error, hidden)
    w1 -= alpha * np.outer(hidden_error, x)

    b2 -= alpha * output_error
    b1 -= alpha * hidden_error

    return w1, w2, b1, b2

# Initialization
w1 = np.random.rand(hidden_size, input_size)  # Weights for input to hidden␣
 ↪layer
w2 = np.random.rand(output_size, hidden_size)  # Weights for hidden to output␣
 ↪layer
b1 = np.random.rand(hidden_size)  # Bias for hidden layer
b2 = np.random.rand(output_size)  # Bias for output layer

epochs = 500  # Training iterations
alpha = 0.5  # Learning rate
error = []

# Training Loop
```

```
for _ in range(epochs):
    idx = np.random.randint(0, X.shape[0])  # Select a random sample
    x = X[idx]
    desired = np.array([y[idx]])  # Ensure desired output is an array

    hidden, output = feed_forward(b1, b2, w1, w2, x)
    error.append(find_error(output, desired))
    w1, w2, b1, b2 = back_propagate(w1, w2, b1, b2, hidden, output, desired, x,␣
 ↪alpha)

# Plot Error Reduction Over Time
plt.plot(error)
plt.xlabel("Epochs")
plt.ylabel("Error")




plt.title("Error Reduction During Training")
plt.show()

# Final Output after Training
print("Final Output:", output)
```




```
Final Output: [0.46558981]
```