AIN SHAMS UNIVERSITY
FACULTY OF ENGINEERING
International Credit Hours Programs (ICHEP)
Mechatronics and Automation

Computational Intelligence (CSE473)

Lab 5 Report

Name	ID	Section
Mahmoud Elsayd Abdelqader Labib Eldwakhly	21P0017	1

Submitted to Dr Hossam Hassan & Eng. Ali Ahmed

Fall 2025

1. Objective

The goal of this assignment is to implement and evaluate two classifiers on a synthetic double moon dataset, which consists of two interleaving half-moon shapes representing two classes:

- Class 1 (Red) – top moon
- Class 0 (Blue) – bottom moon

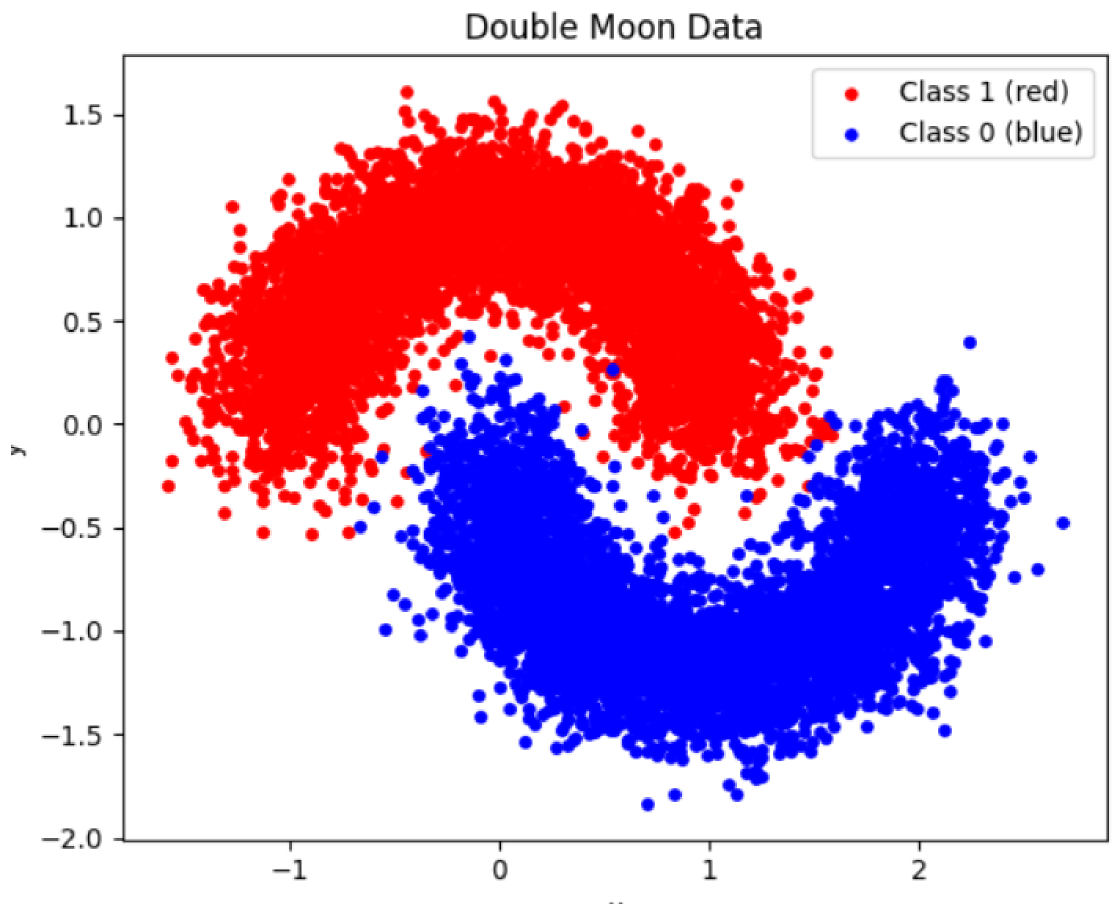
Specifically, the tasks are:

1. Generate the dataset with user-defined numbers of points per class.
2. Train a linear classifier (Logistic Regression).
3. Train a Multi-Layer Neural Network (MLNN / MLP).
4. Visualize decision boundaries and loss curves.
5. Compare performance between linear and non-linear classifiers.

2. Dataset Generation

- **Number of points per class Entered by user:**
    - Red (Class 1) = N1 = 5000 entered.
    - Blue (Class 0) = N2 = 5000 entered.
- **Double moon parameters:**
    - Radius $r = 1.0$
    - Vertical offset $d = 0.2$
    - Noise added with standard deviation $\sigma = 0.2$
- **Data split:**
    - Training: 64%
    - Validation: 16%
    - Test: 20%

- Red points form the upper half-moon.
- Blue points form the lower half-moon.
- Noise introduces slight overlap, making linear separation challenging.



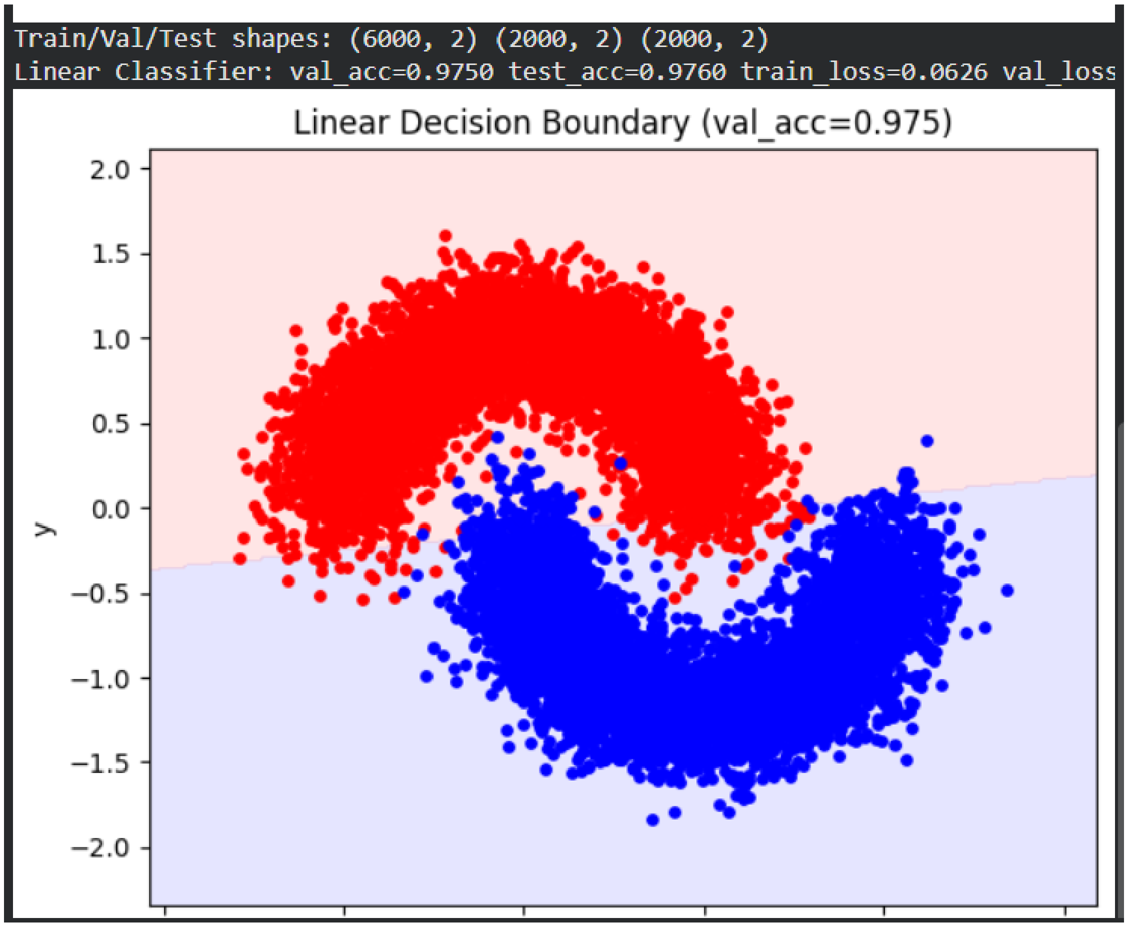
3. Linear Classifier

3.1 Implementation

- Logistic Regression (scikit-learn)
- Maximum iterations: 500
- Input: 2D points (x, y)
- Output: Binary probability $P(y = 1 \mid x)$

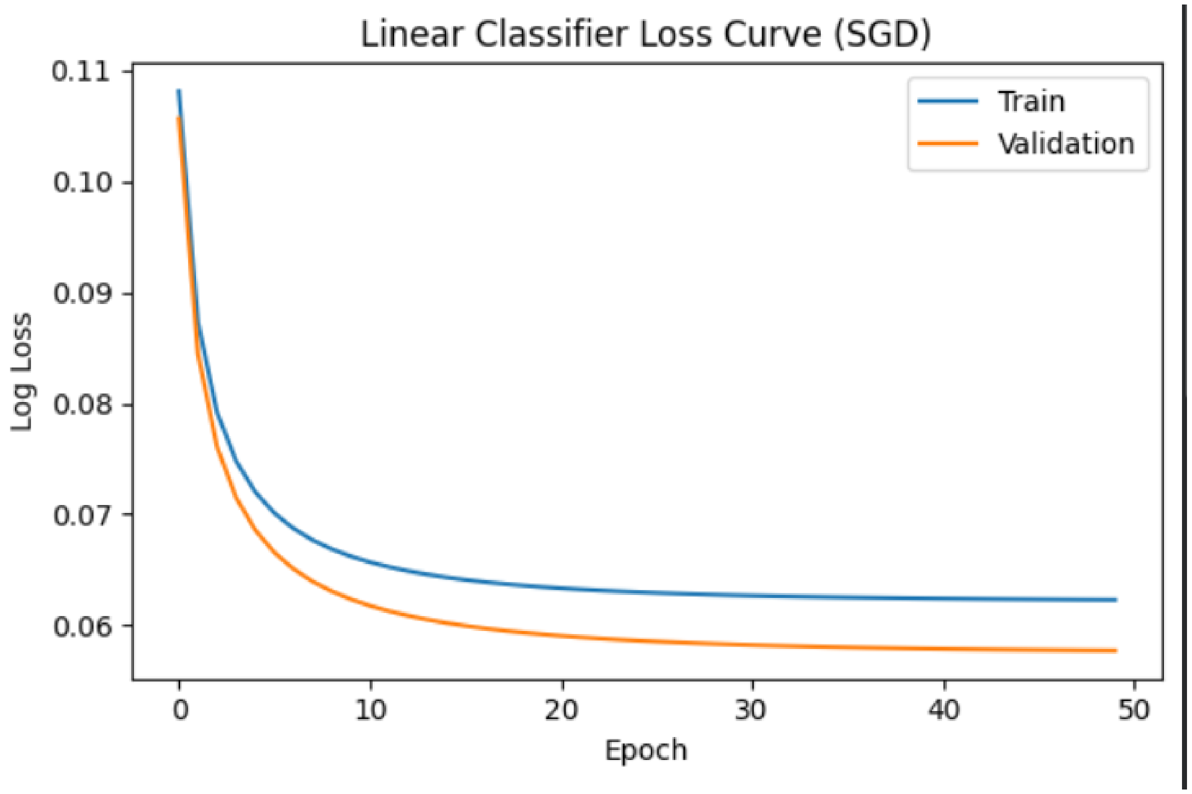
3.2 Training and Validation

- **Loss metric:** Log-loss (cross-entropy)
- **Accuracy metric:** fraction of correctly classified points



```
Train/Val/Test shapes: (6000, 2) (2000, 2) (2000, 2)
Linear Classifier: val_acc=0.9750 test_acc=0.9760 train_loss=0.0626 val_loss
```

- The linear classifier is able to separate points along a straight line.
- Overlapping regions of the moons are misclassified.



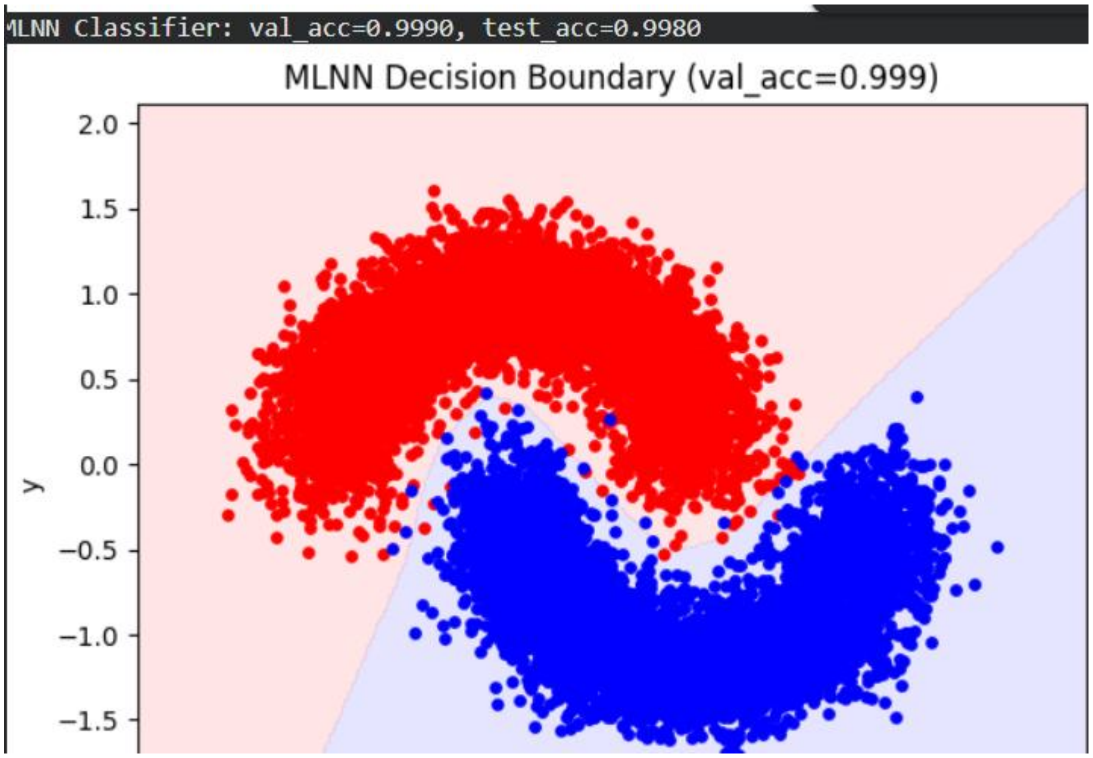
Multi-Layer Neural Network (MLNN)

Architecture

- Input layer: 2 neurons (x and y coordinates)
- Hidden layers:
    1. Linear(2 → 64) + ReLU
    2. Linear(64 → 64) + ReLU
- Output layer: Linear(64 → 1)
- Loss: Binary Cross-Entropy with Logits (BCEWithLogitsLoss)
- Optimizer: Adam, learning rate 0.01
- Early stopping: 20 epochs patience on validation loss

Training

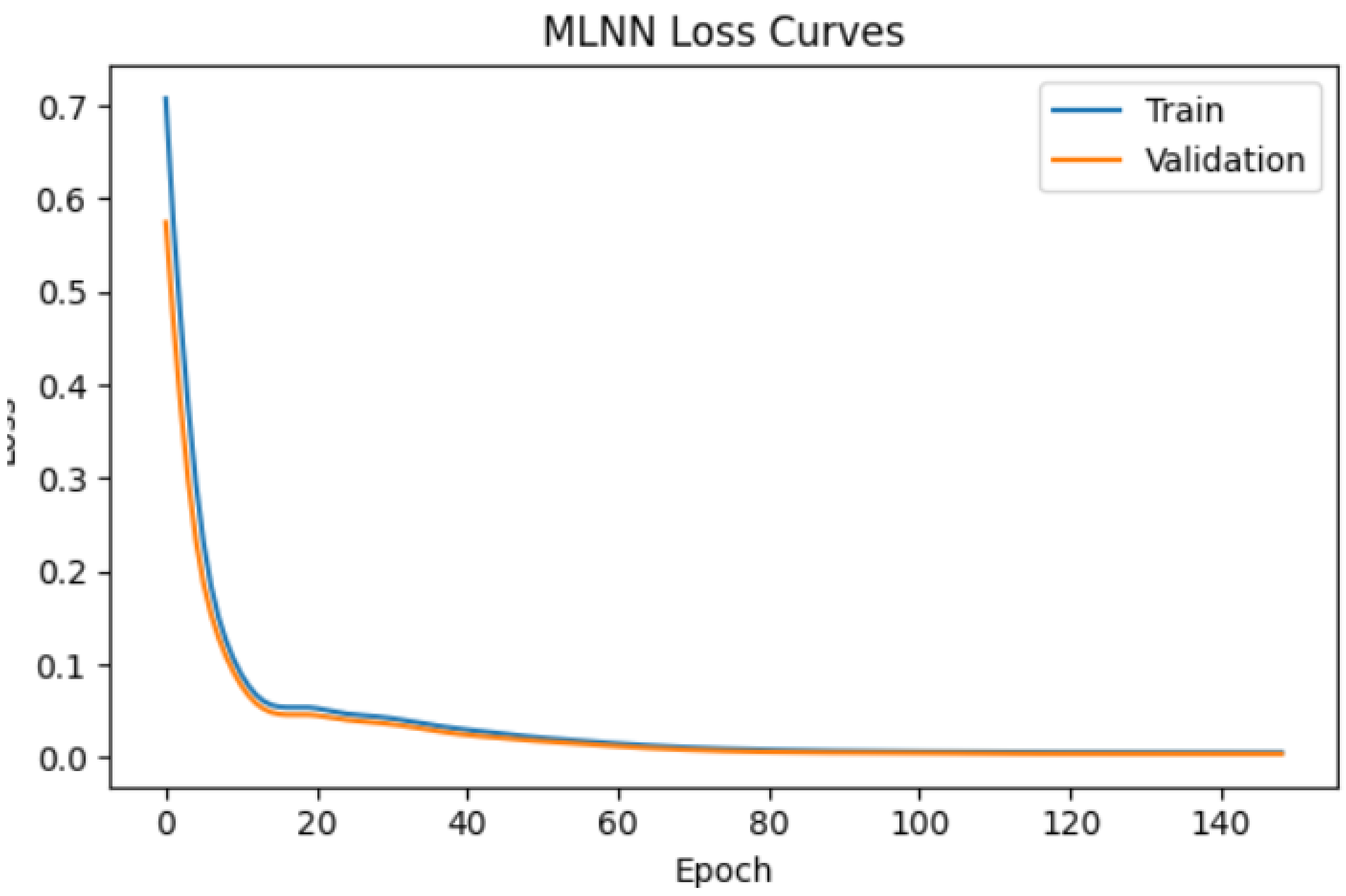
- Converted data to PyTorch tensors on CPU/GPU
- Tracked training and validation loss over epochs

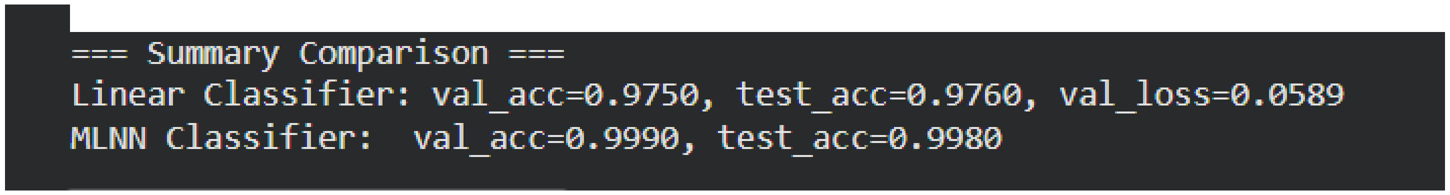


```
MLNN Classifier: val_acc=0.9990, test_acc=0.9980
```

- MLNN captures the non-linear shape of the moons.
- Decision boundary follows the curvature of the dataset.
- Higher accuracy than the linear model on overlapping regions.

- Loss decreases quickly in first epochs and stabilizes.
- Early stopping prevents overfitting.





```
=== Summary Comparison ===
Linear Classifier: val_acc=0.9750, test_acc=0.9760, val_loss=0.0589
MLNN Classifier:  val_acc=0.9990, test_acc=0.9980
```

- Linear classifier works for approximately linearly separable regions but fails on overlapping curved areas.
- MLNN significantly improves accuracy by learning non-linear boundaries.
- The double moon dataset is a classic example where non-linear models outperform linear ones.

Data is not perfectly separable

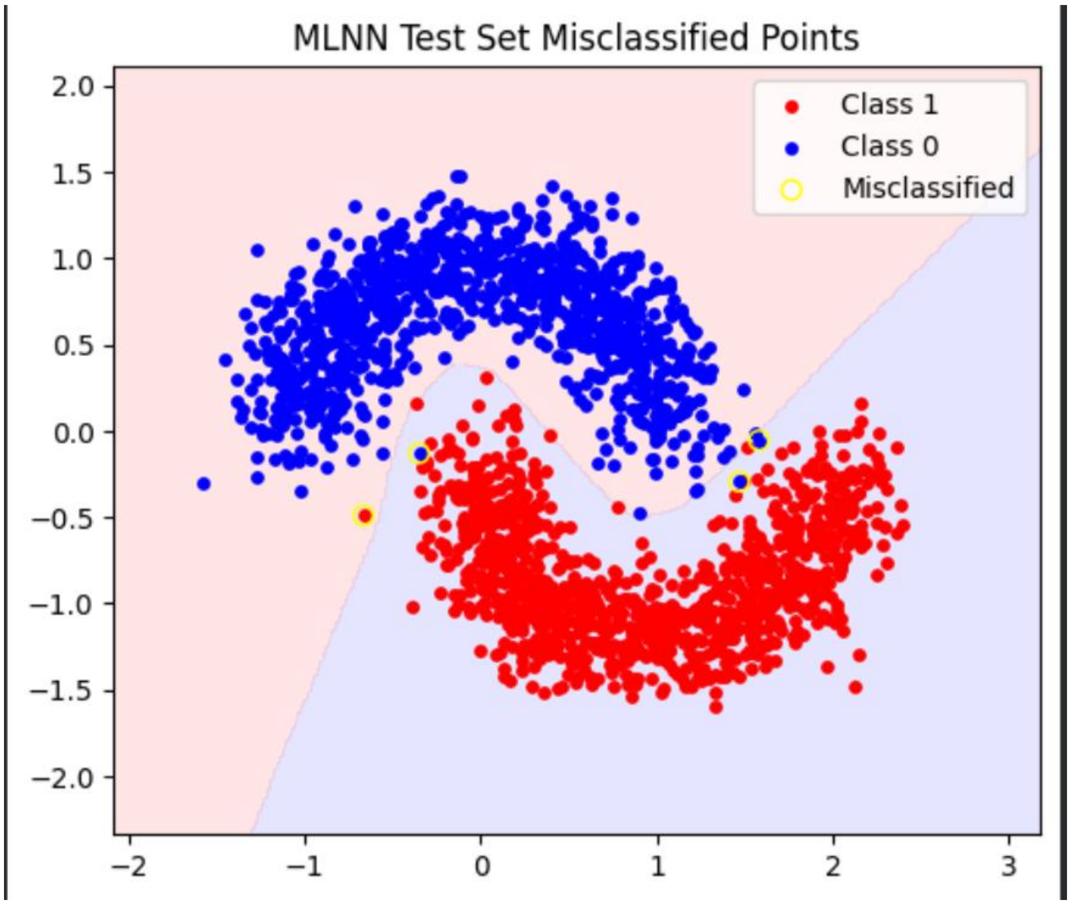
- The double moon shapes overlap slightly because of the noise (d=0.2) we add to each point.
- No classifier, even a complex MLNN, can achieve 100% accuracy unless the data is perfectly separable.

MLNN is a statistical model

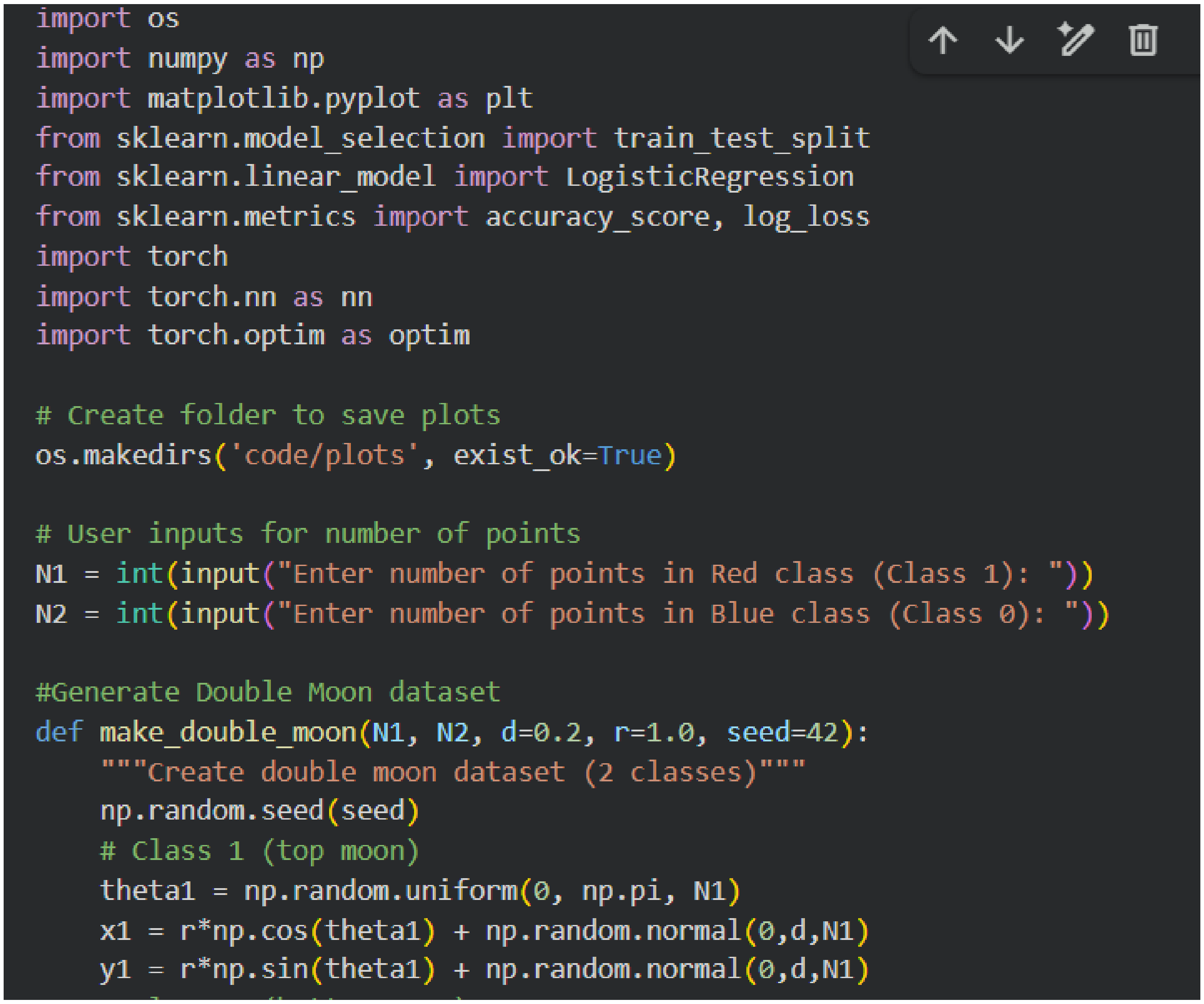
- MLNN outputs a probability via sigmoid.
- Using a threshold of 0.5 to assign class labels, points near the boundary may be misclassified.
- Increasing the network size or training longer can reduce errors, but some errors due to inherent noise will always exist.

Comparison with linear classifier

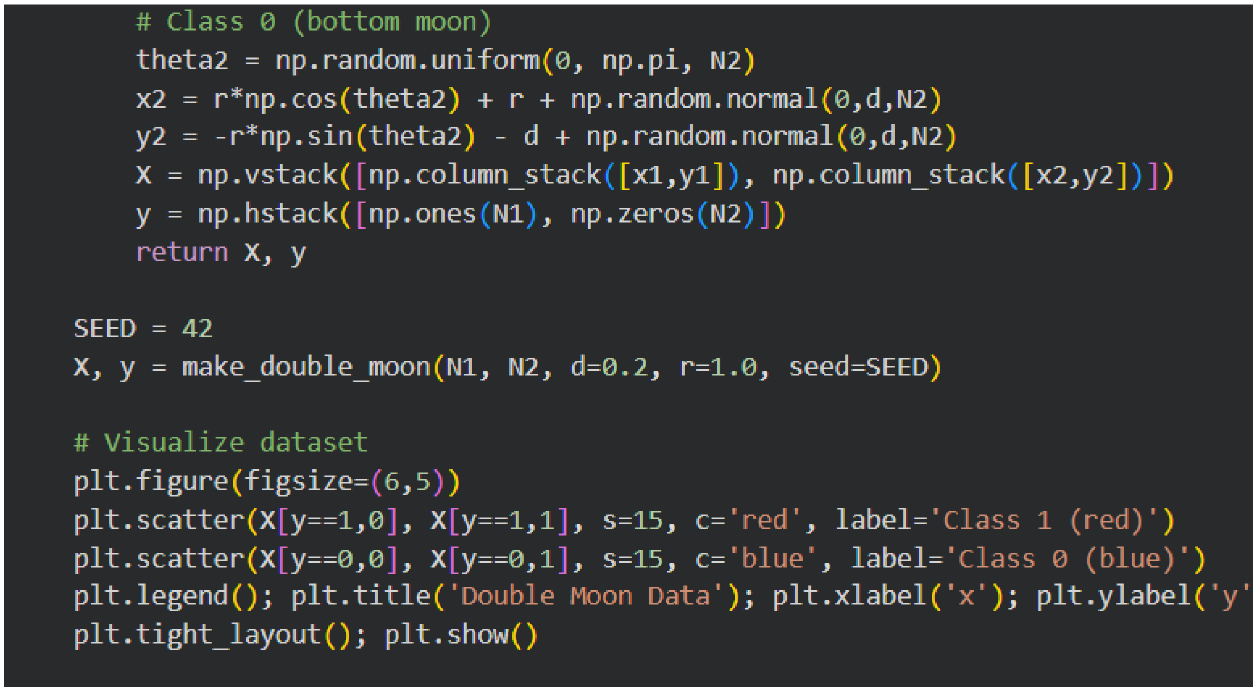
- Linear classifier: can’t model the curved boundary of the moons → more errors.
- MLNN: can model the nonlinear shape → fewer errors, but still not zero because of noise.



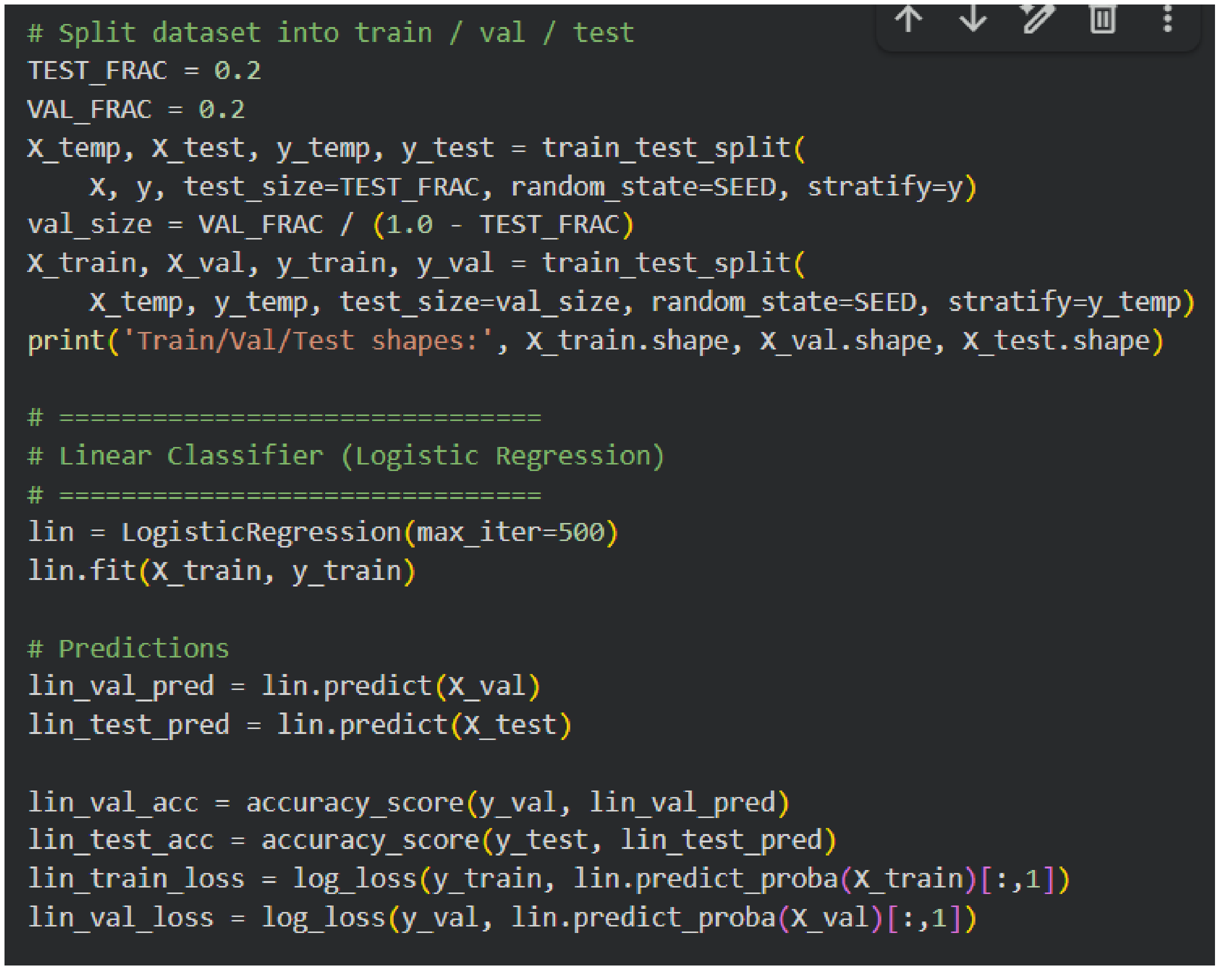
Code :



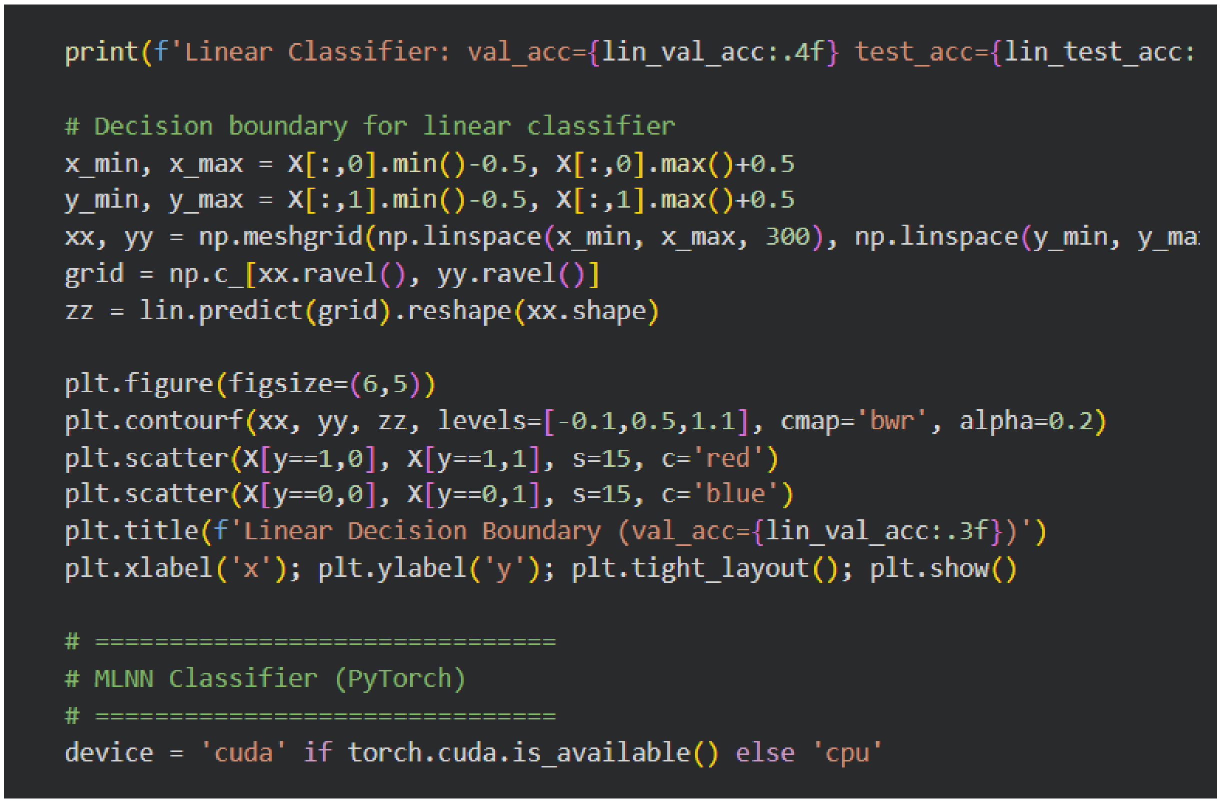
```python
import os
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, log_loss
import torch
import torch.nn as nn
import torch.optim as optim

# Create folder to save plots
os.makedirs('code/plots', exist_ok=True)

# User inputs for number of points
N1 = int(input("Enter number of points in Red class (Class 1): "))
N2 = int(input("Enter number of points in Blue class (Class 0): "))

#Generate Double Moon dataset
def make_double_moon(N1, N2, d=0.2, r=1.0, seed=42):
    """Create double moon dataset (2 classes)"""
    np.random.seed(seed)
    # Class 1 (top moon)
    theta1 = np.random.uniform(0, np.pi, N1)
    x1 = r*np.cos(theta1) + np.random.normal(0,d,N1)
    y1 = r*np.sin(theta1) + np.random.normal(0,d,N1)
```



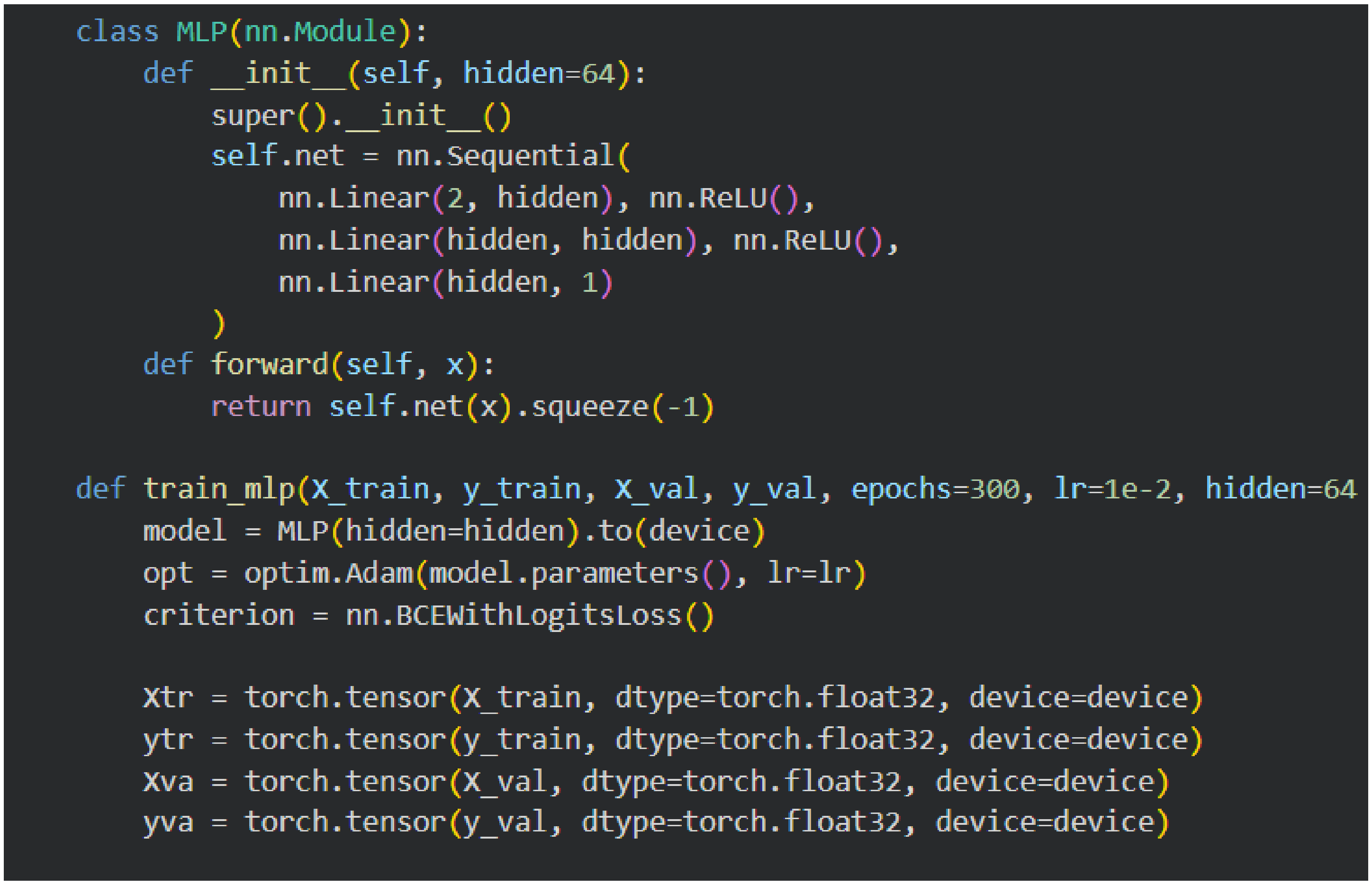
```python
    # Class 0 (bottom moon)
    theta2 = np.random.uniform(0, np.pi, N2)
    x2 = r*np.cos(theta2) + r + np.random.normal(0,d,N2)
    y2 = -r*np.sin(theta2) - d + np.random.normal(0,d,N2)
    X = np.vstack([np.column_stack([x1,y1]), np.column_stack([x2,y2])])
    y = np.hstack([np.ones(N1), np.zeros(N2)])
    return X, y

SEED = 42
X, y = make_double_moon(N1, N2, d=0.2, r=1.0, seed=SEED)

# Visualize dataset
plt.figure(figsize=(6,5))
plt.scatter(X[y==1,0], X[y==1,1], s=15, c='red', label='Class 1 (red)')
plt.scatter(X[y==0,0], X[y==0,1], s=15, c='blue', label='Class 0 (blue)')
plt.legend(); plt.title('Double Moon Data'); plt.xlabel('x'); plt.ylabel('y'
plt.tight_layout(); plt.show()
```



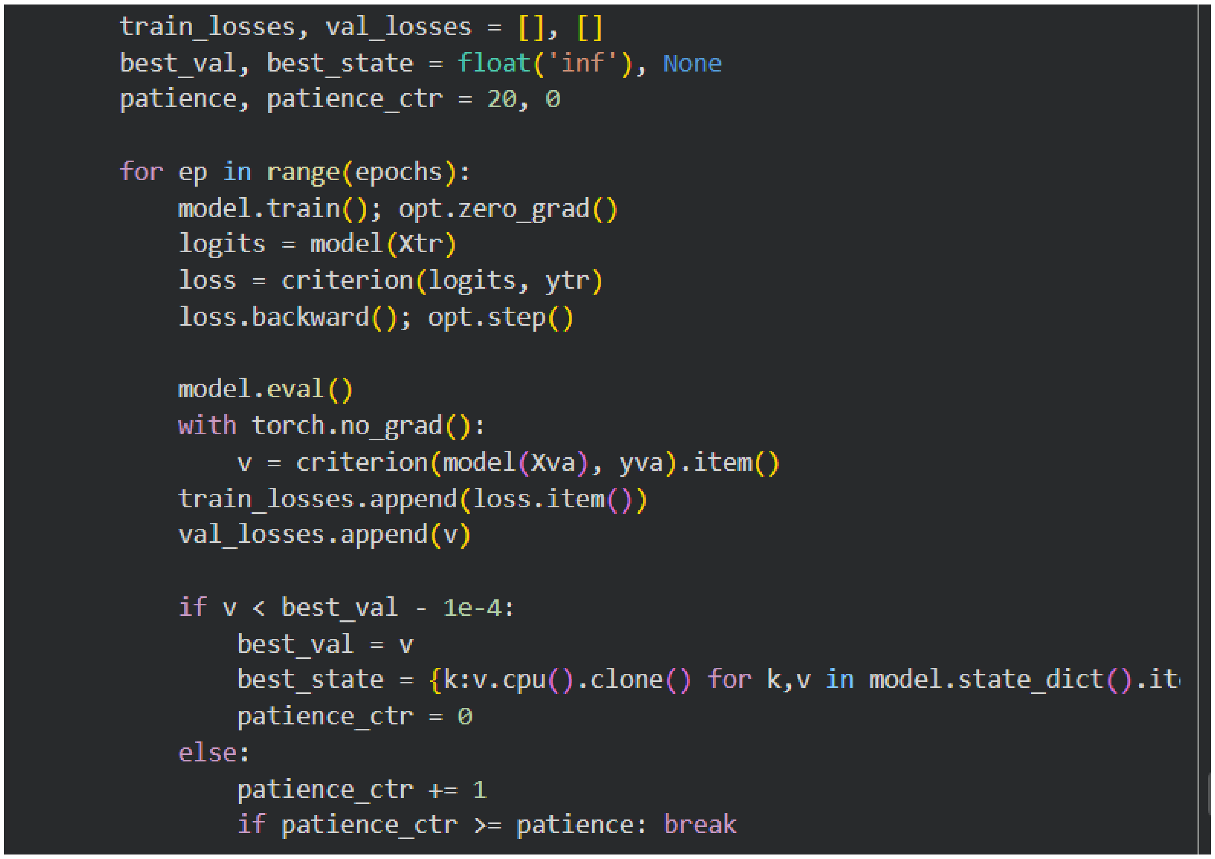
```python
# Split dataset into train / val / test
TEST_FRAC = 0.2
VAL_FRAC = 0.2
X_temp, X_test, y_temp, y_test = train_test_split(
    X, y, test_size=TEST_FRAC, random_state=SEED, stratify=y)
val_size = VAL_FRAC / (1.0 - TEST_FRAC)
X_train, X_val, y_train, y_val = train_test_split(
    X_temp, y_temp, test_size=val_size, random_state=SEED, stratify=y_temp)
print('Train/Val/Test shapes:', X_train.shape, X_val.shape, X_test.shape)

# ===============================
# Linear Classifier (Logistic Regression)
# ===============================
lin = LogisticRegression(max_iter=500)
lin.fit(X_train, y_train)

# Predictions
lin_val_pred = lin.predict(X_val)
lin_test_pred = lin.predict(X_test)

lin_val_acc = accuracy_score(y_val, lin_val_pred)
lin_test_acc = accuracy_score(y_test, lin_test_pred)
lin_train_loss = log_loss(y_train, lin.predict_proba(X_train)[:,1])
lin_val_loss = log_loss(y_val, lin.predict_proba(X_val)[:,1])
```



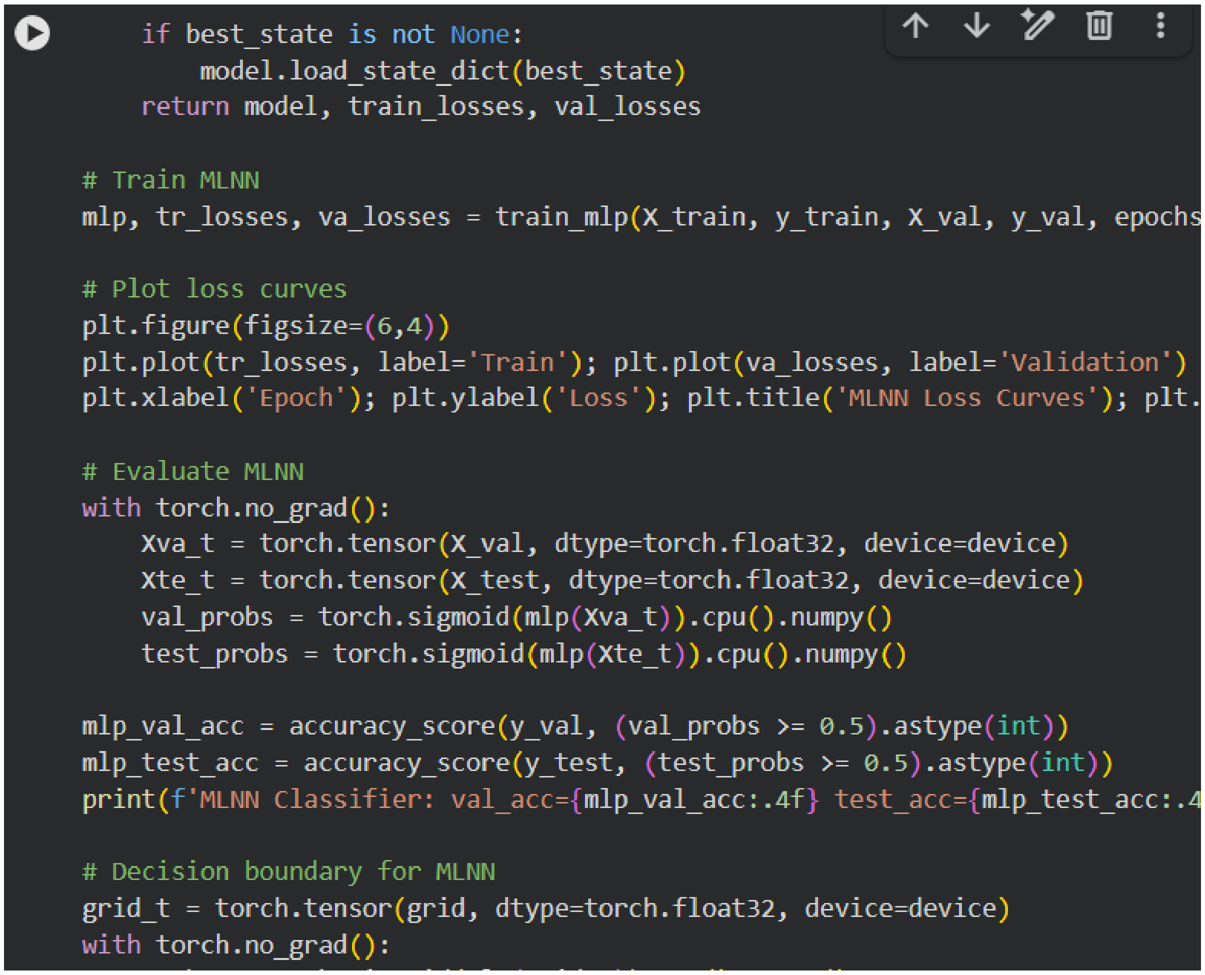
```python
print(f'Linear Classifier: val_acc={lin_val_acc:.4f} test_acc={lin_test_acc:

# Decision boundary for linear classifier
x_min, x_max = X[:,0].min()-0.5, X[:,0].max()+0.5
y_min, y_max = X[:,1].min()-0.5, X[:,1].max()+0.5
xx, yy = np.meshgrid(np.linspace(x_min, x_max, 300), np.linspace(y_min, y_ma
grid = np.c_[xx.ravel(), yy.ravel()]
zz = lin.predict(grid).reshape(xx.shape)

plt.figure(figsize=(6,5))
plt.contourf(xx, yy, zz, levels=[-0.1,0.5,1.1], cmap='bwr', alpha=0.2)
plt.scatter(X[y==1,0], X[y==1,1], s=15, c='red')
plt.scatter(X[y==0,0], X[y==0,1], s=15, c='blue')
plt.title(f'Linear Decision Boundary (val_acc={lin_val_acc:.3f})')
plt.xlabel('x'); plt.ylabel('y'); plt.tight_layout(); plt.show()

# ===============================
# MLNN Classifier (PyTorch)
# ===============================
device = 'cuda' if torch.cuda.is_available() else 'cpu'
```



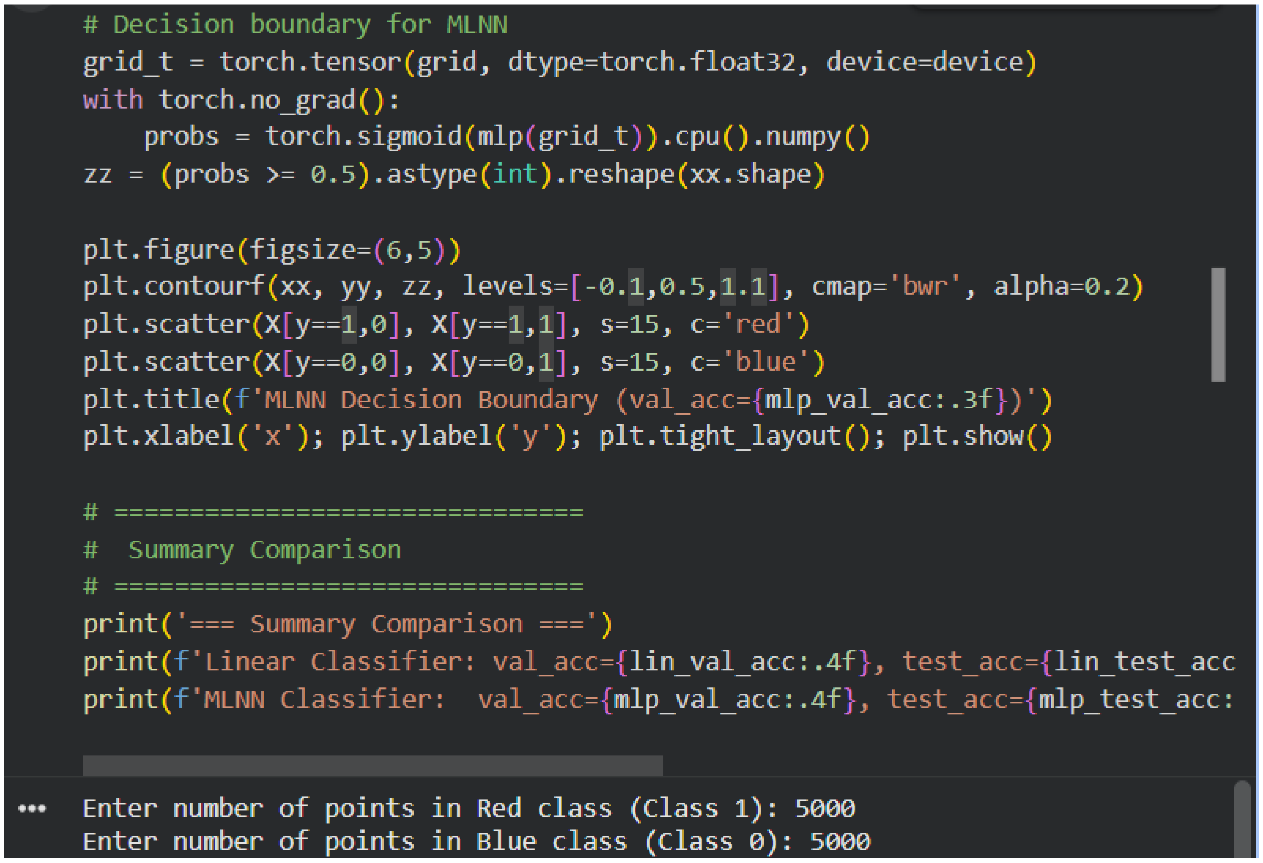
```
class MLP(nn.Module):
    def __init__(self, hidden=64):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(2, hidden), nn.ReLU(),
            nn.Linear(hidden, hidden), nn.ReLU(),
            nn.Linear(hidden, 1)
        )
    def forward(self, x):
        return self.net(x).squeeze(-1)

def train_mlp(X_train, y_train, X_val, y_val, epochs=300, lr=1e-2, hidden=64
    model = MLP(hidden=hidden).to(device)
    opt = optim.Adam(model.parameters(), lr=lr)
    criterion = nn.BCEWithLogitsLoss()

    Xtr = torch.tensor(X_train, dtype=torch.float32, device=device)
    ytr = torch.tensor(y_train, dtype=torch.float32, device=device)
    Xva = torch.tensor(X_val, dtype=torch.float32, device=device)
    yva = torch.tensor(y_val, dtype=torch.float32, device=device)
```



```
    train_losses, val_losses = [], []
    best_val, best_state = float('inf'), None
    patience, patience_ctr = 20, 0

    for ep in range(epochs):
        model.train(); opt.zero_grad()
        logits = model(Xtr)
        loss = criterion(logits, ytr)
        loss.backward(); opt.step()

        model.eval()
        with torch.no_grad():
            v = criterion(model(Xva), yva).item()
        train_losses.append(loss.item())
        val_losses.append(v)

        if v < best_val - 1e-4:
            best_val = v
            best_state = {k:v.cpu().clone() for k,v in model.state_dict().it
            patience_ctr = 0
        else:
            patience_ctr += 1
            if patience_ctr >= patience: break
```



```
    if best_state is not None:
        model.load_state_dict(best_state)
    return model, train_losses, val_losses

# Train MLNN
mlp, tr_losses, va_losses = train_mlp(X_train, y_train, X_val, y_val, epochs

# Plot loss curves
plt.figure(figsize=(6,4))
plt.plot(tr_losses, label='Train'); plt.plot(va_losses, label='Validation')
plt.xlabel('Epoch'); plt.ylabel('Loss'); plt.title('MLNN Loss Curves'); plt.

# Evaluate MLNN
with torch.no_grad():
    Xva_t = torch.tensor(X_val, dtype=torch.float32, device=device)
    Xte_t = torch.tensor(X_test, dtype=torch.float32, device=device)
    val_probs = torch.sigmoid(mlp(Xva_t)).cpu().numpy()
    test_probs = torch.sigmoid(mlp(Xte_t)).cpu().numpy()

mlp_val_acc = accuracy_score(y_val, (val_probs >= 0.5).astype(int))
mlp_test_acc = accuracy_score(y_test, (test_probs >= 0.5).astype(int))
print(f'MLNN Classifier: val_acc={mlp_val_acc:.4f} test_acc={mlp_test_acc:.4

# Decision boundary for MLNN
grid_t = torch.tensor(grid, dtype=torch.float32, device=device)
with torch.no_grad():
```



```
# Decision boundary for MLNN
grid_t = torch.tensor(grid, dtype=torch.float32, device=device)
with torch.no_grad():
    probs = torch.sigmoid(mlp(grid_t)).cpu().numpy()
zz = (probs >= 0.5).astype(int).reshape(xx.shape)

plt.figure(figsize=(6,5))
plt.contourf(xx, yy, zz, levels=[-0.1,0.5,1.1], cmap='bwr', alpha=0.2)
plt.scatter(X[y==1,0], X[y==1,1], s=15, c='red')
plt.scatter(X[y==0,0], X[y==0,1], s=15, c='blue')
plt.title(f'MLNN Decision Boundary (val_acc={mlp_val_acc:.3f})')
plt.xlabel('x'); plt.ylabel('y'); plt.tight_layout(); plt.show()

# ================================
#  Summary Comparison
# ================================
print('=== Summary Comparison ===')
print(f'Linear Classifier: val_acc={lin_val_acc:.4f}, test_acc={lin_test_acc
print(f'MLNN Classifier:  val_acc={mlp_val_acc:.4f}, test_acc={mlp_test_acc:
```

```
Enter number of points in Red class (Class 1): 5000
Enter number of points in Blue class (Class 0): 5000
```