FINAL EXAMINATION

- Các bạn có thời gian làm bài là 2.5h (Từ 15h40 - 18h00)
- Chỉ được sử dụng tài liệu có sẵn và không được sử dụng những tài liệu tham khảo là AI(ChatGPT, Gemini,...)
- Cố gắng làm hết khả năng của mình nha :)

Câu 1: 2 Điểm

In [28]:
```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report
# 1. Tải dữ liệu Iris
iris = load_iris()
X = iris.data
# 2. Chia dữ liệu thành tập huấn luyện và kiểm tra
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.9, random_sta
# 3. Tạo mô hình Naive Bayes
model = GaussianNB()
# 4. Huấn luyện mô hình
model.fit(, X_test)
# 5. Dự đoán trên tập kiểm tra
y_prediction = model.predict(X_test)

# 6. Đánh giá mô hình
accuracy = accuracy(y_test, y_prediction)
print("Độ chính xác của mô hình:", accuracy)

# 7. Dự đoán với một mẫu mới
new_sample = [5.1, 3.5, 1.4, 0.2]
predicted_class = model.predict(new_sample)
print(f"Mẫu mới được dự đoán là lớp: {iris.target_names[y_prediction]}")
```

```
  Cell In[28], line 13
    model.fit(, X_test)
              ^
SyntaxError: invalid syntax
```

Đoạn code trên đúng hay sai hay bất thường? Nếu sai hoặc bất thường thì sửa lại như thế nào cho đúng ?

Đoạn code trên bất thường : không có y; huấn luyện mô hình thiếu y test, accuracy thêm thư viện là accyracy_score, dư thư viện classification_report, test_size=0.3 chứ không phải 0.9,model.fit(, X_test) là model.fit(X_train, y_train)

In [110…

```python
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report

# 1. Tải dữ liệu Iris
iris = load_iris()
X = iris.data
y = iris.target

# 2. Chia dữ liệu thành tập huấn luyện và kiểm tra
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_sta

# 3. Tạo mô hình Naive Bayes
model = GaussianNB()

# 4. Huấn luyện mô hình
model.fit(X_train, y_train)

# 5. Dự đoán trên tập kiểm tra
y_prediction = model.predict(X_test)

# 6. Đánh giá mô hình
accuracy = accuracy_score(y_test, y_prediction)
print("Độ chính xác của mô hình:", accuracy)

# 7. Dự đoán với một mẫu mới
new_sample = [[5.1, 3., 1.4, 0.2]]
predicted_class = model.predict(new_sample)
print(f"Mẫu mới được dự đoán là lớp: {iris.target_names[predicted_class]}")
```

```
Độ chính xác của mô hình: 0.9777777777777777
Mẫu mới được dự đoán là lớp: ['setosa']
```

Câu 2: 3 Điểm

Giả sử SVM tuyến tính tối ưu có $\mathbf{w} = [1 \quad 1]^\top$ và $b = -2$. Tìm biên hình học $\gamma^{(i)}$ cho $\mathbf{x}^{(i)} = [2 \quad 5]^\top$ và $y^{(i)} = 1$.

Biên hình học cho ví dụ i là

$$\gamma^{(i)} = y^{(i)} \left(\frac{\mathbf{w}^\top \mathbf{x}^{(i)}}{\|\mathbf{w}\|} + \frac{b}{\|\mathbf{w}\|} \right),$$

theo code bên dưới là 3.536.

In [94]:

```python
import numpy as np

# Định nghĩa các biến
w = np.array([1, 1])
b = -2
x_i = np.array([2, 5])
```

```
y_i = 1

# Tính độ dài của w
norm_w = np.linalg.norm(w)

# Tính giá trị gamma
gamma = y_i * (np.dot(w, x_i) / norm_w + b / norm_w)

# Xuất kết quả
print("gamma =", round(gamma, 3))
```

```
gamma = 3.536
```

Kết quả tham khảo:

Gamma = 3.536

Câu 3: 5 Điểm

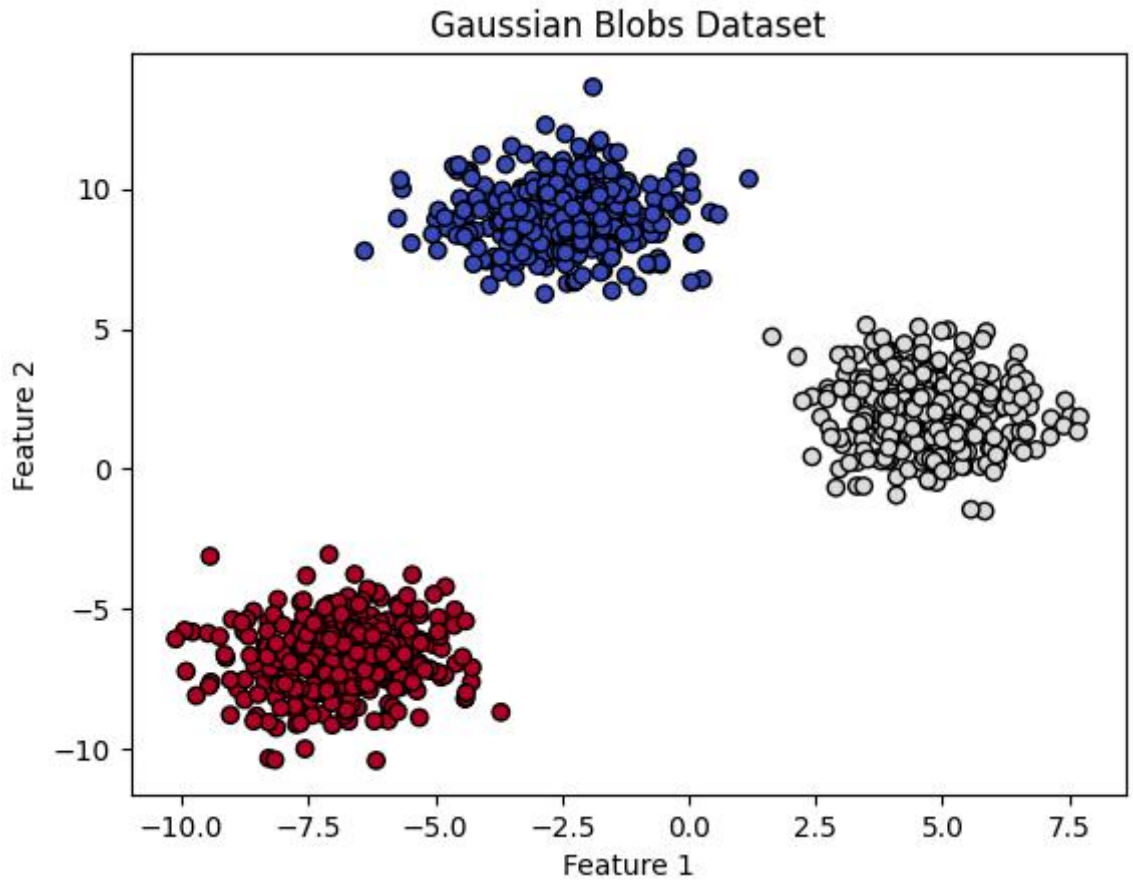
Điền vào chỗ trống bên dưới để mô hình MLP có thể hoạt động tốt và cho kết quả chính xác

In [137…

```
import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# 1. Tạo dữ liệu Gaussian blobs
X, y = make_blobs(n_samples=1000, centers=3, cluster_std=1.2, random_state=42)
# Vẽ dữ liệu để kiểm tra
plt.scatter(, , c=y, cmap=plt.cm.coolwarm, edgecolor='k')
plt.title("Gaussian Blobs Dataset")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()
```

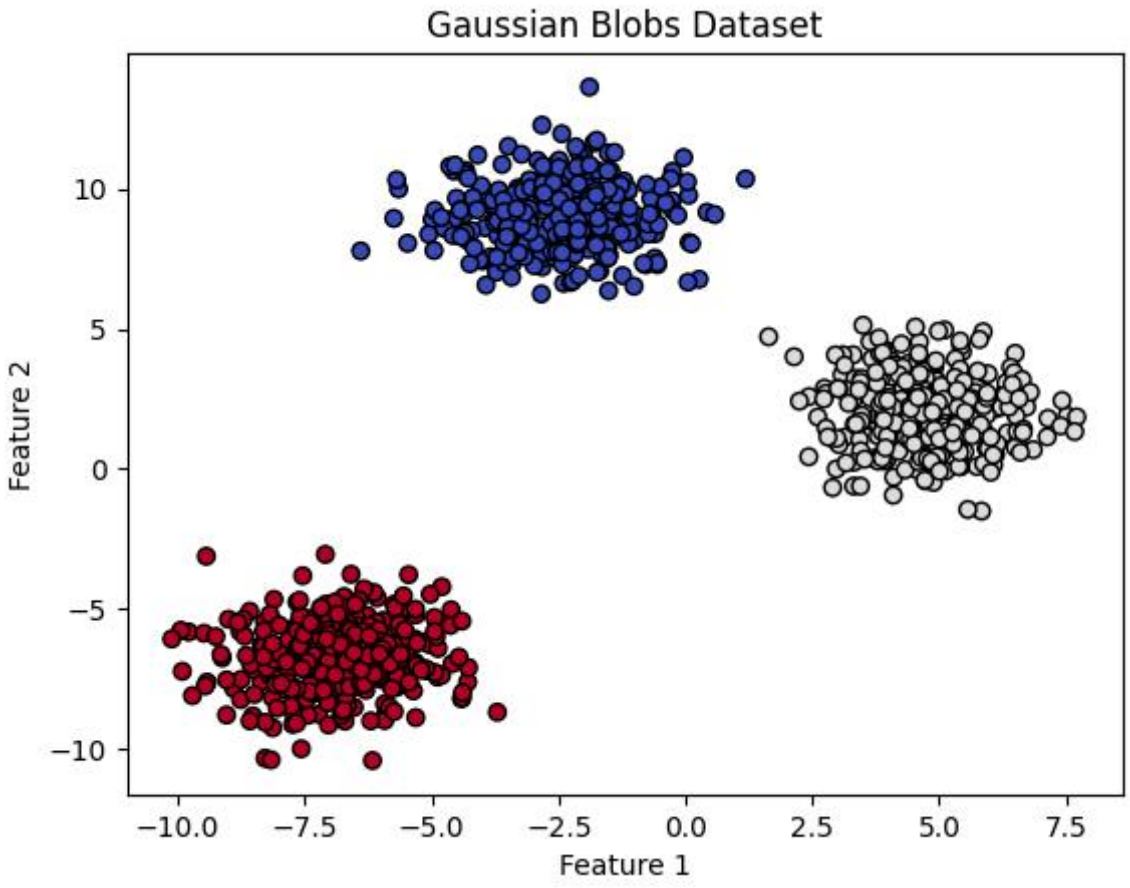
```
  Cell In[137], line 13
    plt.scatter(, , c=y, cmap=plt.cm.coolwarm, edgecolor='k')
                ^
SyntaxError: invalid syntax
```

Kết quả tham khảo :



In [138…

```python
import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# 1. Tạo dữ liệu Gaussian blobs
X, y = make_blobs(n_samples=1000, centers=3, cluster_std=1.2, random_state=42)

# Vẽ dữ liệu để kiểm tra
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.coolwarm, edgecolor='k')
plt.title("Gaussian Blobs Dataset")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()
```



In [ ]:
```python
# 2. Chia dữ liệu thành train/test
np.random.seed(42)
indices = #code here
np.random.shuffle(indices)
test_size = #code here # 20% cho tập test
train_size = #code here

train_indices = #code here
test_indices = #code here

X_train = #code here
y_train = #code here
X_test = #code here
y_test = #code here
```

In [139…
```python
# 2. Chia dữ liệu thành train/test
np.random.seed(42)
indices = np.arange(len(X))  #code here
np.random.shuffle(indices)  #code here
test_size = int(0.2 * len(X))  #code here # 20% cho tập test
train_size = len(X) - test_size  #code here

train_indices = indices[:train_size]  #code here
test_indices = indices[train_size:]  #code here

X_train = X[train_indices]  #code here
y_train = y[train_indices]  #code here
```

```
X_test = X[test_indices]  #code here
y_test = y[test_indices]  #code here
```

In [ ]:
```
# Chuẩn hóa dữ liệu
mean = #code here  # Trung bình của mỗi đặc trưng
std = #code here   # Độ lệch chuẩn của mỗi đặc trưng
X_train = #code here # Chuẩn hóa tập train
X_test = #code here   # Chuẩn hóa tập test theo mean và std của tập train
# Chuyển đổi dữ liệu thành PyTorch tensors
X_train = #code here)
y_train = #code here
X_test = #code here
y_test = #code here
```

In [140…
```
# Chuẩn hóa dữ liệu
mean = X_train.mean(axis=0)  # Trung bình của mỗi đặc trưng
std = X_train.std(axis=0)    # Độ lệch chuẩn của mỗi đặc trưng
X_train = (X_train - mean) / std  # Chuẩn hóa tập train
X_test = (X_test - mean) / std    # Chuẩn hóa tập test theo mean và std của tập tra

# Chuyển đổi dữ liệu thành PyTorch tensors
X_train = torch.tensor(X_train, dtype=torch.float32)  #code here
y_train = torch.tensor(y_train, dtype=torch.long)     #code here
X_test = torch.tensor(X_test, dtype=torch.float32)    #code here
y_test = torch.tensor(y_test, dtype=torch.long)       #code here
```

In [ ]:
```
# 3. Xây dựng mô hình MLP
class MLP(nn.Module):
    def __init__(self):
        super(MLP, self).__init__()
        self.fc1 = #code here  # 2 đặc trưng đầu vào -> 4 nơ-ron ẩn
        self.relu1 = #code here      # Hàm kích hoạt ReLU
        self.fc2 = #code here  # 4 nơ-ron ẩn -> 2 nơ-ron ẩn
        self.relu2 = #code here    # Hàm kích hoạt ReLU
        self.fc3 = #code here  # 2 nơ-ron ẩn -> 3 lớp đầu ra

    def forward(self, x):
        x = #code here
        x = #code here
        x = #code here
        x = #code here
        x = #code here
        return x
```

In [141…
```
# 3. Xây dựng mô hình MLP
class MLP(nn.Module):
    def __init__(self):
        super(MLP, self).__init__()
        self.fc1 = nn.Linear(2, 4)  # 2 đặc trưng đầu vào -> 4 nơ-ron ẩn
        self.relu1 = nn.ReLU()     # Hàm kích hoạt ReLU
        self.fc2 = nn.Linear(4, 2)  # 4 nơ-ron ẩn -> 2 nơ-ron ẩn
        self.relu2 = nn.ReLU()     # Hàm kích hoạt ReLU
        self.fc3 = nn.Linear(2, 3)  # 2 nơ-ron ẩn -> 3 lớp đầu ra

    def forward(self, x):
```

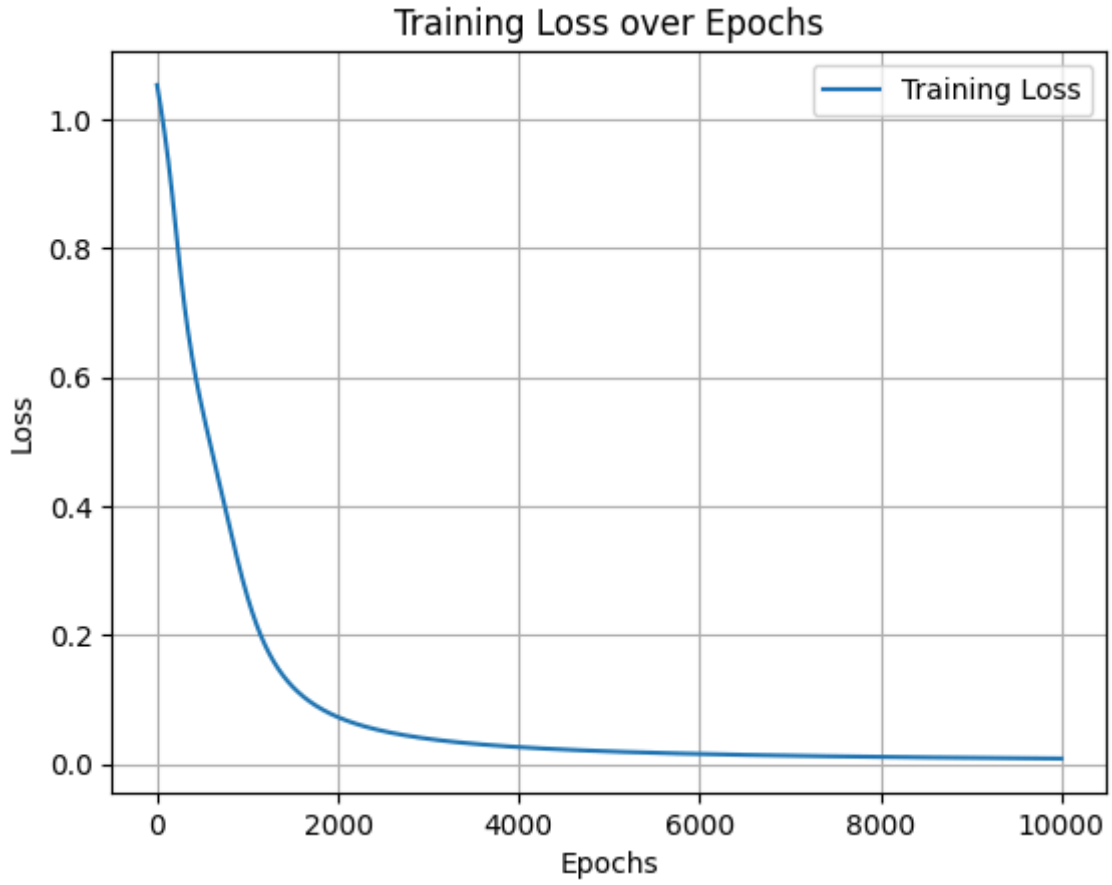
```
        x = self.fc1(x)  # code here
        x = self.relu1(x)  # code here
        x = self.fc2(x)  # code here
        x = self.relu2(x)  # code here
        x = self.fc3(x)  # code here
        return x
```

In [ ]:
```
# 4. Khởi tạo mô hình, hàm mất mát, và bộ tối ưu
model =
criterion =   # Hàm mất mát CrossEntropy cho bài toán phân loại
optimizer =   # Tối ưu hóa bằng SGD với lr= 0.01
```

In [153…
```
# 4. Khởi tạo mô hình, hàm mất mát, và bộ tối ưu
model = MLP()  # Khởi tạo mô hình
criterion = nn.CrossEntropyLoss()  # Hàm mất mát CrossEntropy cho bài toán phân loạ
optimizer = optim.SGD(model.parameters(), lr=0.01)  # Tối ưu hóa bằng SGD với lr=0.
```

In [ ]:
```
losses = []

# Huấn luyện mô hình
epochs = 10000
for epoch in range(epochs):
    model.train()
    outputs = model(X_train)
    loss = criterion(outputs, y_train)

    # Lưu loss vào danh sách
    #code here

    #code here
    #code here
    #code here

    if (epoch + 1) % 100 == 0:
        print(f"Epoch [{epoch+1}/{epochs}], Loss: {loss.item():.4f}")

# 6. Đánh giá mô hình
model.eval()
with torch.no_grad():
    outputs = model(X_test)
    _, predicted = torch.max(outputs, 1)  # Lấy nhãn dự đoán
    accuracy = (predicted == y_test).float().mean()  # Tính độ chính xác
    print(f"Accuracy on test set: {accuracy.item():.4f}")

# Vẽ biểu đồ loss
#code here
```

Kết quả tham khảo



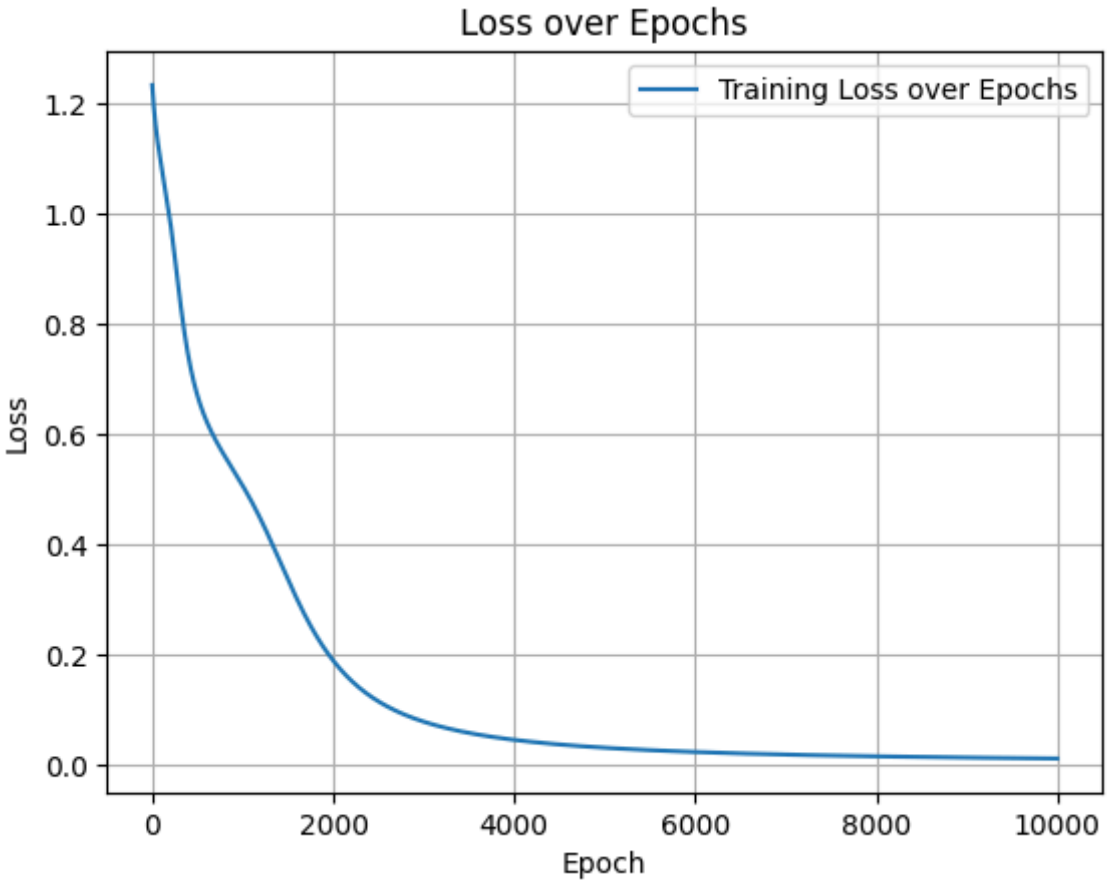
In [154…

```python
# Khởi tạo danh sách để lưu giá trị loss
losses = []

# Huấn luyện mô hình
epochs = 10000
# Vòng lặp huấn luyện
for epoch in range(epochs):
    model.train()
    outputs = model(X_train)
    loss = criterion(outputs, y_train)

    # Lưu loss vào danh sách
    losses.append(loss.item())


    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
    if (epoch + 1) % 100 == 0:
        print(f"Epoch [{epoch+1}/{epochs}], Loss: {loss.item():.4f}")

# 6. Đánh giá mô hình
model.eval()
with torch.no_grad():
    outputs = model(X_test)
    _, predicted = torch.max(outputs, 1)  # Lấy nhãn dự đoán
    accuracy = (predicted == y_test).float().mean()  # Tính độ chính xác
    print(f"Accuracy on test set: {accuracy.item():.4f}")
```

```
# Vẽ biểu đồ loss
#code here
# Vẽ biểu đồ loss
plt.plot(losses, label="Training Loss over Epochs")
plt.title("Loss over Epochs")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
plt.grid(True)
plt.show()
```

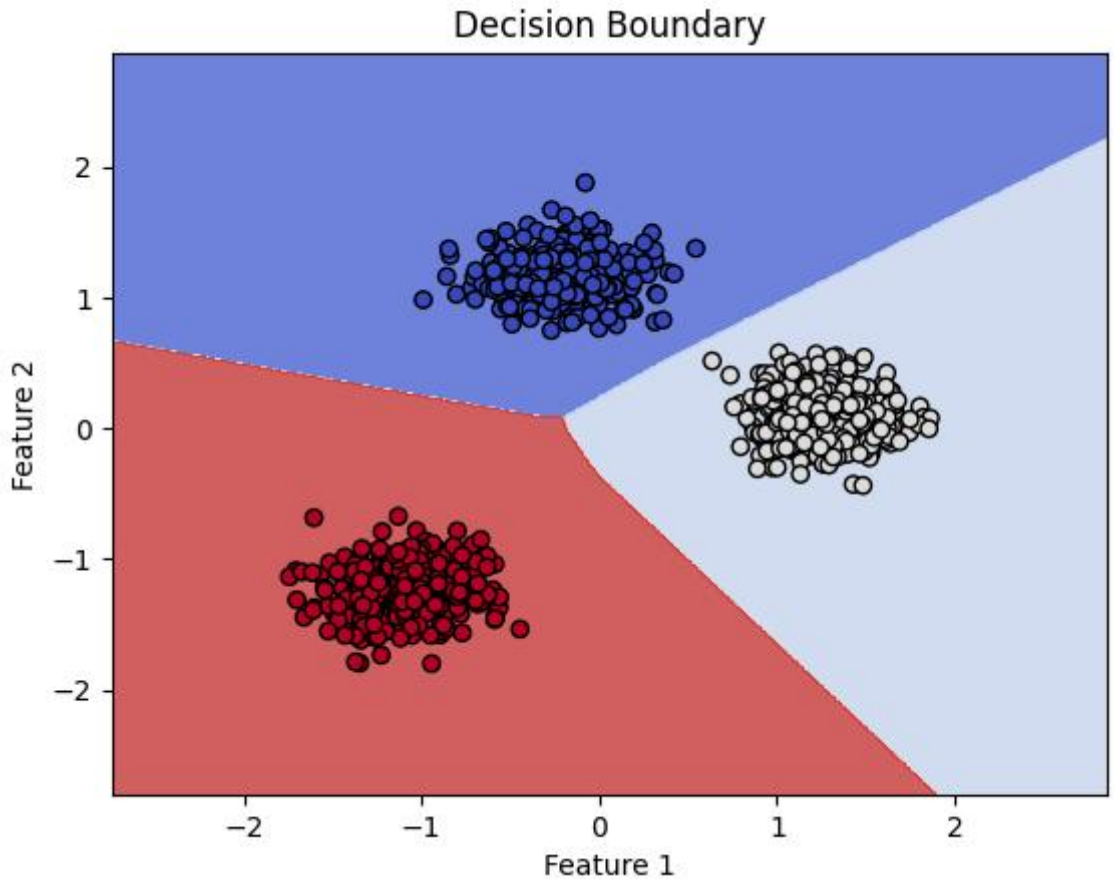
```
Epoch [100/10000], Loss: 1.0891
Epoch [200/10000], Loss: 0.9859
Epoch [300/10000], Loss: 0.8534
Epoch [400/10000], Loss: 0.7414
Epoch [500/10000], Loss: 0.6702
Epoch [600/10000], Loss: 0.6239
Epoch [700/10000], Loss: 0.5895
Epoch [800/10000], Loss: 0.5607
Epoch [900/10000], Loss: 0.5338
Epoch [1000/10000], Loss: 0.5065
Epoch [1100/10000], Loss: 0.4773
Epoch [1200/10000], Loss: 0.4452
Epoch [1300/10000], Loss: 0.4103
Epoch [1400/10000], Loss: 0.3736
Epoch [1500/10000], Loss: 0.3365
Epoch [1600/10000], Loss: 0.3008
Epoch [1700/10000], Loss: 0.2676
Epoch [1800/10000], Loss: 0.2379
Epoch [1900/10000], Loss: 0.2119
Epoch [2000/10000], Loss: 0.1892
Epoch [2100/10000], Loss: 0.1697
Epoch [2200/10000], Loss: 0.1528
Epoch [2300/10000], Loss: 0.1384
Epoch [2400/10000], Loss: 0.1259
Epoch [2500/10000], Loss: 0.1151
Epoch [2600/10000], Loss: 0.1057
Epoch [2700/10000], Loss: 0.0974
Epoch [2800/10000], Loss: 0.0902
Epoch [2900/10000], Loss: 0.0839
Epoch [3000/10000], Loss: 0.0782
Epoch [3100/10000], Loss: 0.0732
Epoch [3200/10000], Loss: 0.0687
Epoch [3300/10000], Loss: 0.0647
Epoch [3400/10000], Loss: 0.0611
Epoch [3500/10000], Loss: 0.0578
Epoch [3600/10000], Loss: 0.0548
Epoch [3700/10000], Loss: 0.0521
Epoch [3800/10000], Loss: 0.0496
Epoch [3900/10000], Loss: 0.0473
Epoch [4000/10000], Loss: 0.0452
Epoch [4100/10000], Loss: 0.0433
Epoch [4200/10000], Loss: 0.0415
Epoch [4300/10000], Loss: 0.0398
Epoch [4400/10000], Loss: 0.0383
Epoch [4500/10000], Loss: 0.0369
Epoch [4600/10000], Loss: 0.0355
Epoch [4700/10000], Loss: 0.0343
Epoch [4800/10000], Loss: 0.0331
Epoch [4900/10000], Loss: 0.0320
Epoch [5000/10000], Loss: 0.0309
Epoch [5100/10000], Loss: 0.0300
Epoch [5200/10000], Loss: 0.0291
Epoch [5300/10000], Loss: 0.0282
Epoch [5400/10000], Loss: 0.0274
Epoch [5500/10000], Loss: 0.0266
Epoch [5600/10000], Loss: 0.0259
```

```
Epoch [5700/10000], Loss: 0.0252
Epoch [5800/10000], Loss: 0.0245
Epoch [5900/10000], Loss: 0.0239
Epoch [6000/10000], Loss: 0.0233
Epoch [6100/10000], Loss: 0.0227
Epoch [6200/10000], Loss: 0.0221
Epoch [6300/10000], Loss: 0.0216
Epoch [6400/10000], Loss: 0.0211
Epoch [6500/10000], Loss: 0.0206
Epoch [6600/10000], Loss: 0.0202
Epoch [6700/10000], Loss: 0.0198
Epoch [6800/10000], Loss: 0.0193
Epoch [6900/10000], Loss: 0.0189
Epoch [7000/10000], Loss: 0.0185
Epoch [7100/10000], Loss: 0.0182
Epoch [7200/10000], Loss: 0.0178
Epoch [7300/10000], Loss: 0.0175
Epoch [7400/10000], Loss: 0.0171
Epoch [7500/10000], Loss: 0.0168
Epoch [7600/10000], Loss: 0.0165
Epoch [7700/10000], Loss: 0.0162
Epoch [7800/10000], Loss: 0.0159
Epoch [7900/10000], Loss: 0.0156
Epoch [8000/10000], Loss: 0.0154
Epoch [8100/10000], Loss: 0.0151
Epoch [8200/10000], Loss: 0.0149
Epoch [8300/10000], Loss: 0.0146
Epoch [8400/10000], Loss: 0.0144
Epoch [8500/10000], Loss: 0.0141
Epoch [8600/10000], Loss: 0.0139
Epoch [8700/10000], Loss: 0.0137
Epoch [8800/10000], Loss: 0.0135
Epoch [8900/10000], Loss: 0.0133
Epoch [9000/10000], Loss: 0.0131
Epoch [9100/10000], Loss: 0.0129
Epoch [9200/10000], Loss: 0.0127
Epoch [9300/10000], Loss: 0.0125
Epoch [9400/10000], Loss: 0.0124
Epoch [9500/10000], Loss: 0.0122
Epoch [9600/10000], Loss: 0.0120
Epoch [9700/10000], Loss: 0.0119
Epoch [9800/10000], Loss: 0.0117
Epoch [9900/10000], Loss: 0.0115
Epoch [10000/10000], Loss: 0.0114
Accuracy on test set: 1.0000
```



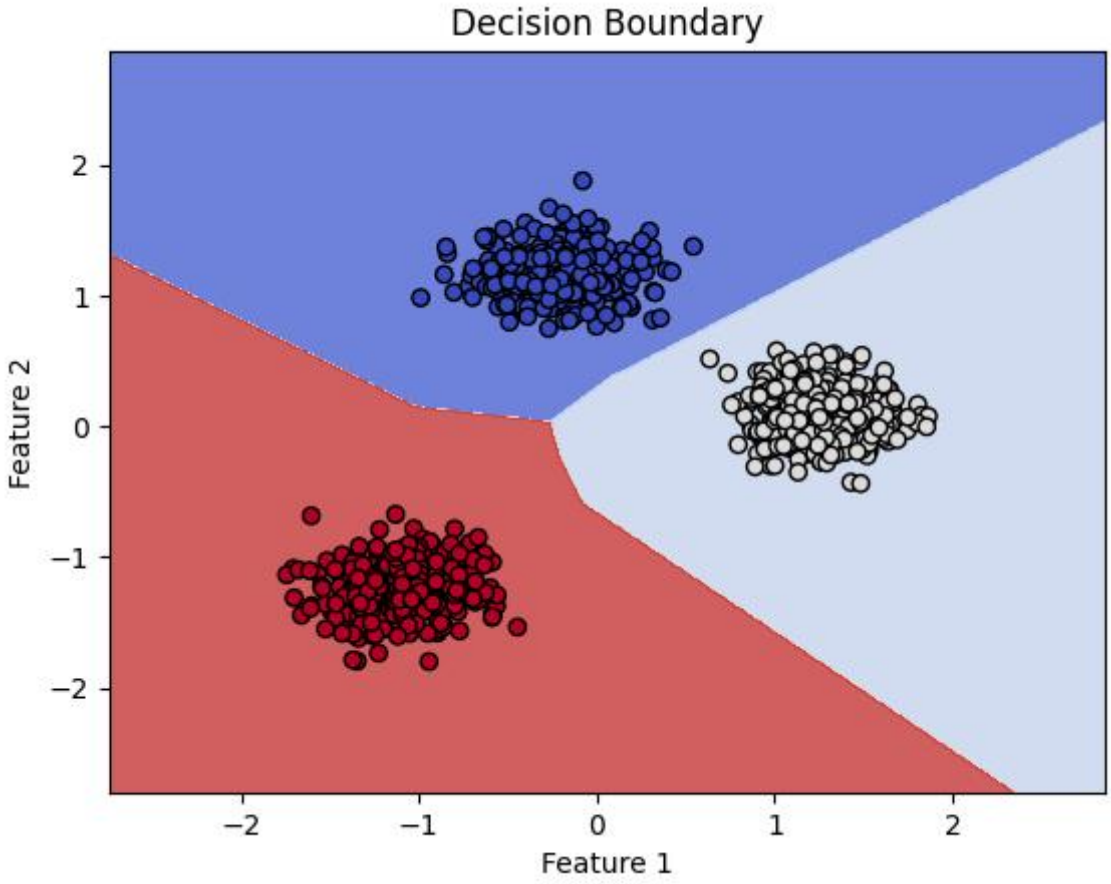
In [ ]:
```python
# 7. Vẽ quyết định ranh giới
def plot_decision_boundary(model, X, y):
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01),
                         np.arange(y_min, y_max, 0.01))
    grid = np.c_[xx.ravel(), yy.ravel()]
    grid_tensor = torch.tensor(grid, dtype=torch.float32)

    with torch.no_grad():
        preds = model(grid_tensor)
        preds = torch.argmax(preds, axis=1).numpy()

    plt.contourf(xx, yy, preds.reshape(xx.shape), alpha=0.8, cmap=plt.cm.coolwarm)
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.coolwarm, edgecolors='k')
    plt.title("Decision Boundary")
    plt.xlabel("Feature 1")
    plt.ylabel("Feature 2")
    plt.show()

#Ranh giới phân loại
plot_decision_boundary(model, np.vstack((X_train.numpy(), X_test.numpy())), np.hsta
```

Kết quả Tham khảo:



In [155…

```python
# 7. Vẽ quyết định ranh giới
def plot_decision_boundary(model, X, y):
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01),
                         np.arange(y_min, y_max, 0.01))
    grid = np.c_[xx.ravel(), yy.ravel()]
    grid_tensor = torch.tensor(grid, dtype=torch.float32)

    with torch.no_grad():
        preds = model(grid_tensor)
        preds = torch.argmax(preds, axis=1).numpy()

    plt.contourf(xx, yy, preds.reshape(xx.shape), alpha=0.8, cmap=plt.cm.coolwarm)
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.coolwarm, edgecolors='k')
    plt.title("Decision Boundary")
    plt.xlabel("Feature 1")
    plt.ylabel("Feature 2")
    plt.show()

#Ranh giới phân loại
plot_decision_boundary(model, np.vstack((X_train.numpy(), X_test.numpy())), np.hsta
```


Decision Boundary
Feature 2
Feature 1
2
1
0
−1
−2
−2
−1
0
1
2