fda

May 25, 2025

[70]:
```
from torch_geometric.datasets import TUDataset
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from matplotlib.colors import ListedColormap
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch_geometric.nn import GCNConv, global_mean_pool
from torch_geometric.loader import DataLoader
from sklearn.metrics import accuracy_score
import torch.optim as optim
from sklearn.linear_model import LogisticRegression
import networkx as nx
from torch_geometric.utils import to_networkx

features = np.load('features.npy')
labels = np.load('labels.npy')
```

[71]:
```
print("Features:", features.shape)
print("Labels:", labels.shape)
```

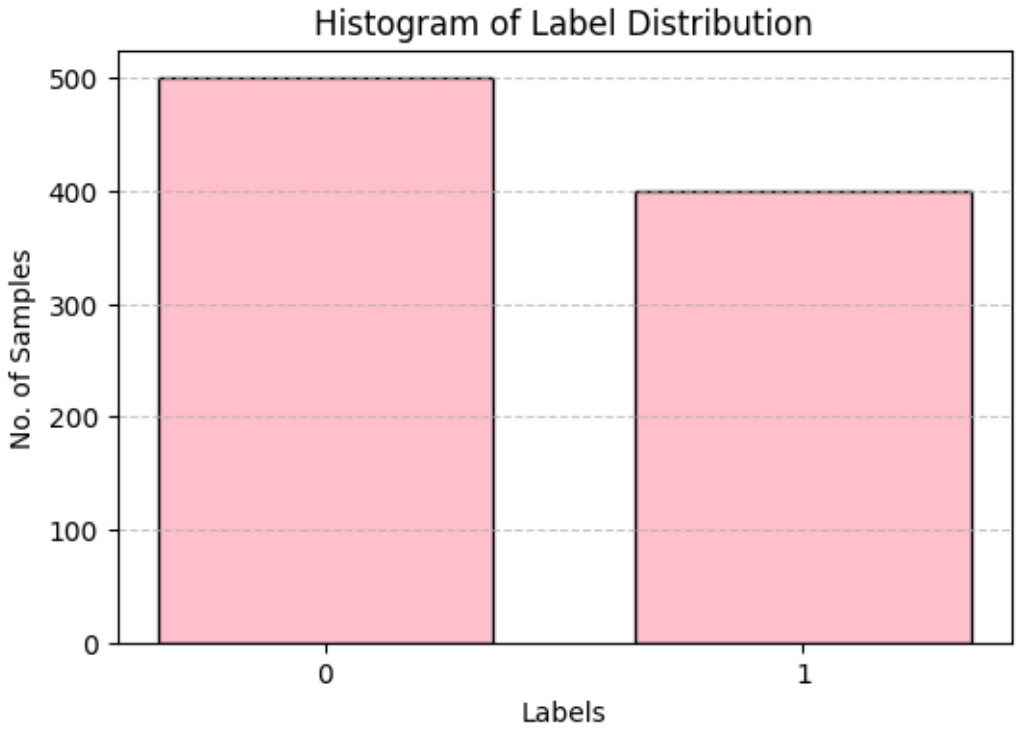
```
Features: (900, 2)
Labels: (900,)
```

[72]:
```
labels = np.load('labels.npy')
unique_labels = np.unique(labels)
print("Unique labels:", unique_labels)
print("Number of unique labels:", len(unique_labels))
```

```
Unique labels: [0. 1.]
Number of unique labels: 2
```

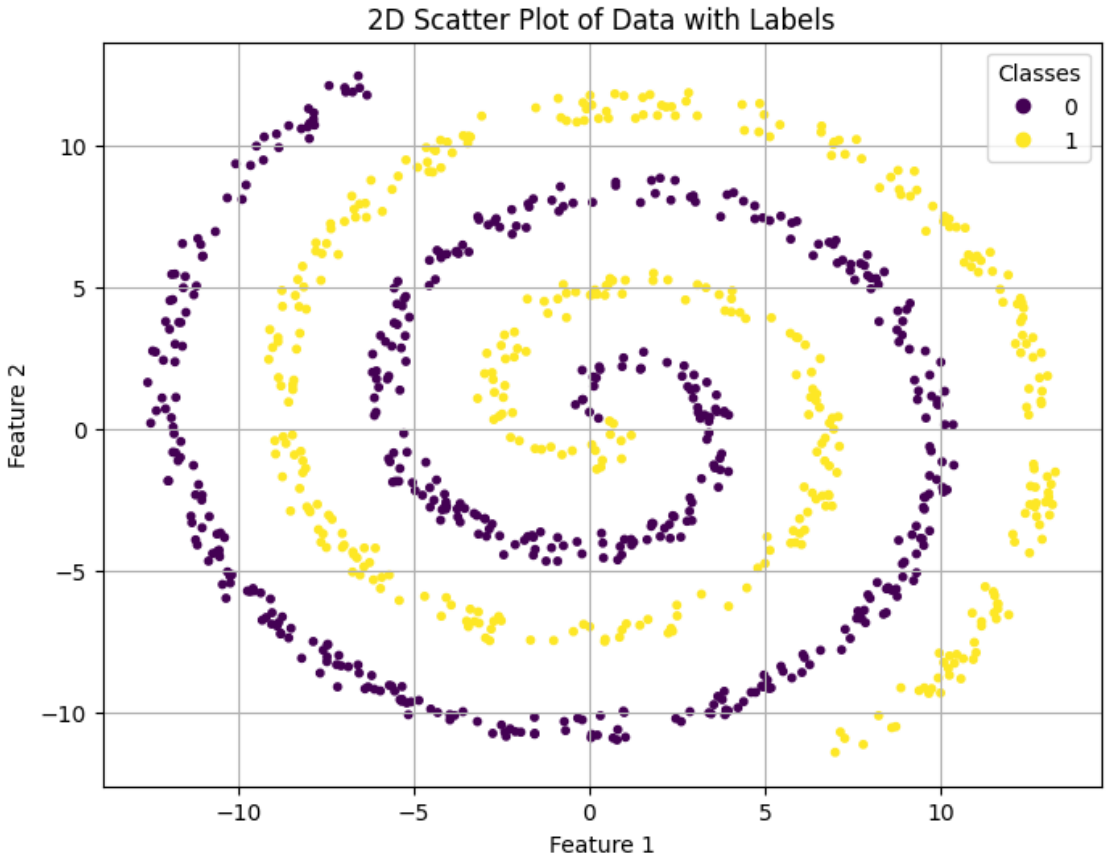
[73]:
```
labels = np.load('labels.npy')
plt.figure(figsize=(6, 4))
plt.hist(labels, bins=np.arange(-0.5, 2, 1), rwidth=0.7, color='pink',␣
 ↪edgecolor='black')
plt.xticks([0, 1])
```

```
plt.title("Histogram of Label Distribution")
plt.xlabel("Labels")
plt.ylabel("No. of Samples")
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```



[74]:
```
if features.shape[1] == 2:
    plt.figure(figsize=(8, 6))
    scatter = plt.scatter(features[:, 0], features[:, 1], c=labels,␣
 ↪cmap='viridis', s=10)
    plt.title('2D Scatter Plot')
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    legend = plt.legend(*scatter.legend_elements(), title="Classes")
    plt.grid(True)
    plt.show()
elif features.shape[1] == 3:
    fig = plt.figure(figsize=(10, 8))
    ax = fig.add_subplot(111, projection='3d')
    scatter = ax.scatter(features[:, 0], features[:, 1], features[:, 2],␣
 ↪c=labels, cmap='viridis', s=10)
    ax.set_title('3D Scatter Plot')
    ax.set_xlabel('Feature 1')
```

```
    ax.set_ylabel('Feature 2')
    plt.show()
else:
    print(f"Data has {features.shape[1]} features. Cannot create .")
```



[75]:
```
X_train, X_test, y_train, y_test = train_test_split(features, labels,␣
  ↪test_size=0.3, random_state=42)
print("Data split complete.")
print(f"Training set shape (features, labels): {X_train.shape}, {y_train.
  ↪shape}")
print(f"Test set shape (features, labels): {X_test.shape}, {y_test.shape}")
```

```
Data split complete.
Training set shape (features, labels): (630, 2), (630,)
Test set shape (features, labels): (270, 2), (270,)
```

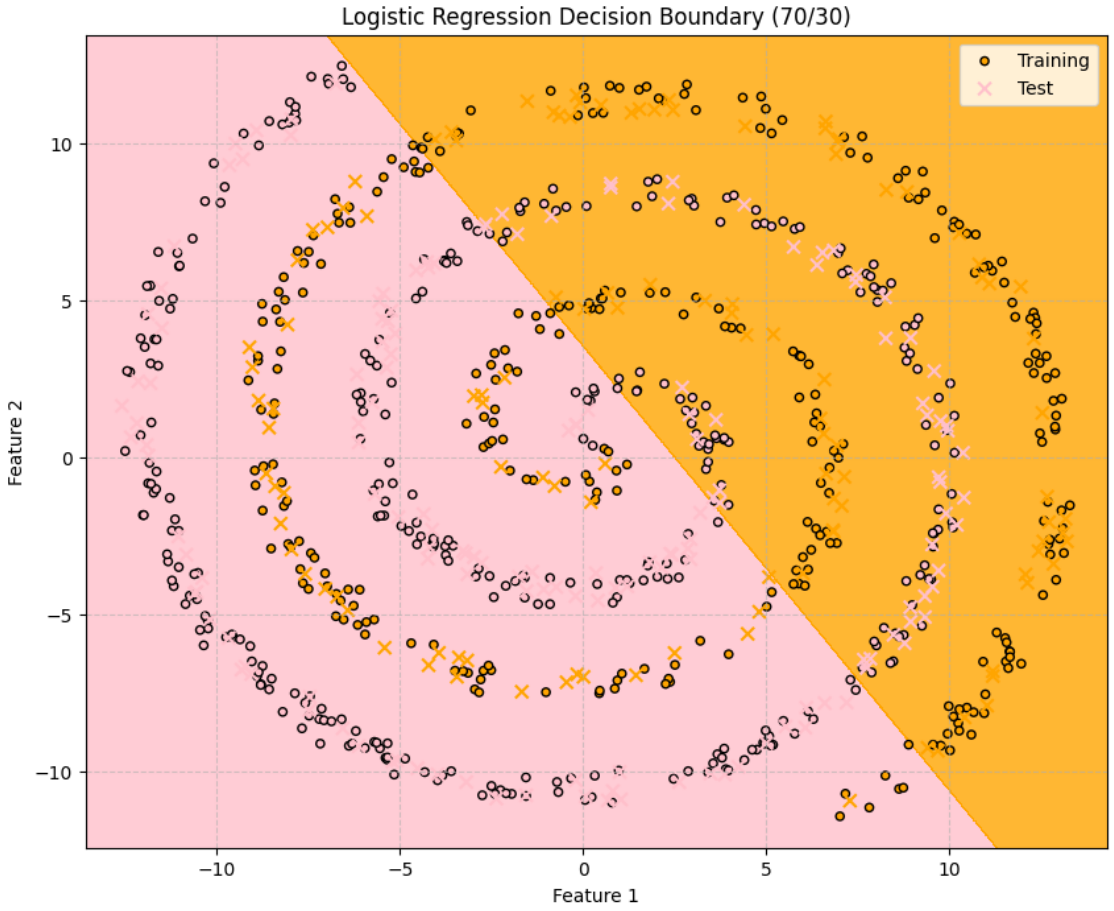
[76]:
```
logistic_classifier = LogisticRegression(random_state=42)
logistic_classifier.fit(X_train, y_train)
print("Logistic Regression classifier trained.")
```

```
from sklearn.metrics import accuracy_score
y_pred = logistic_classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Classification accuracy on the held-out test set: {accuracy:.4f}")
```

```
Logistic Regression classifier trained successfully.
Classification accuracy on the held-out test set: 0.6444
```

[77]:
```
x_min, x_max = features[:, 0].min() - 1, features[:, 0].max() + 1
y_min, y_max = features[:, 1].min() - 1, features[:, 1].max() + 1
h = .02
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
Z = logistic_classifier.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
colors = ['pink', 'orange']
cmap_custom = ListedColormap(colors)
plt.figure(figsize=(10, 8))
plt.contourf(xx, yy, Z, cmap=cmap_custom, alpha=0.8)
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap=cmap_custom,␣
 ↪edgecolors='k', marker='o', s=20, label='Training')
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap=cmap_custom,␣
 ↪edgecolors='k', marker='x', s=50, label='Test')
plt.title('Logistic Regression Decision Boundary (70/30)')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.grid(True, linestyle='--', alpha=0.6)
plt.show()
```

```
/tmp/ipykernel_7020/1320507588.py:14: UserWarning: You passed a
edgecolor/edgecolors ('k') for an unfilled marker ('x').  Matplotlib is ignoring
the edgecolor in favor of the facecolor.  This behavior may change in the
future.
  plt.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap=cmap_custom,
edgecolors='k', marker='x', s=50, label='Test')
```



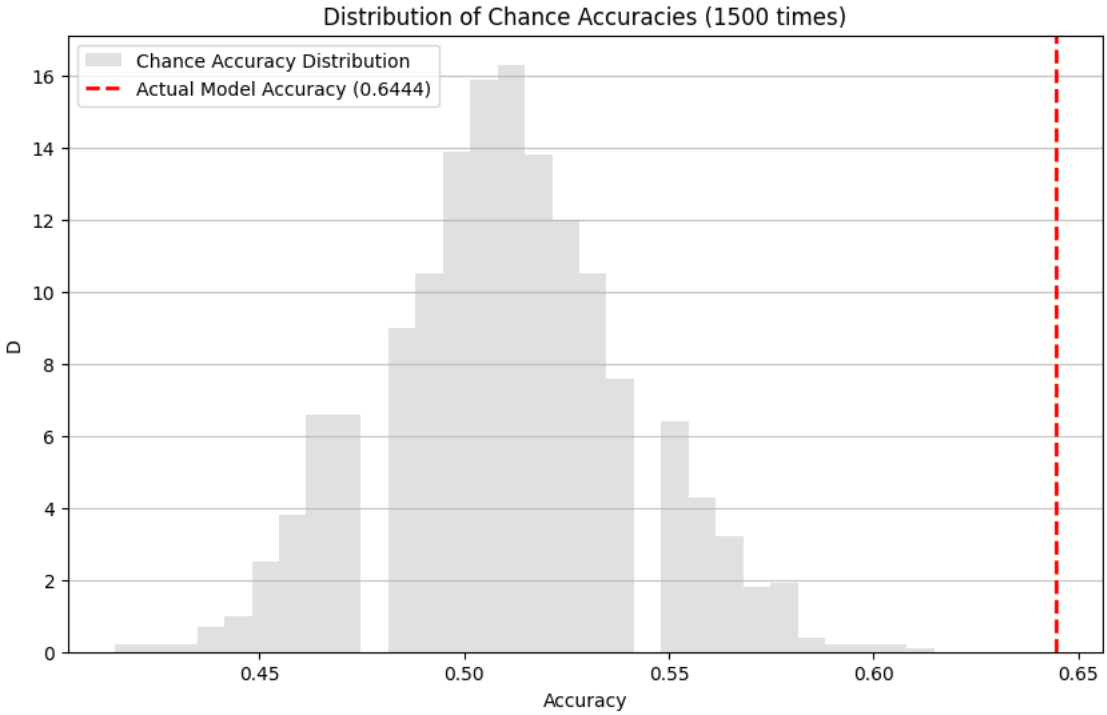
[ ]:

[78]:
```
ytest_shuffled = np.random.permutation(y_test)
chance_accuracy_single = accuracy_score(ytest_shuffled, y_pred)
print(f"Chance accuracy with a shuffle: {chance_accuracy_single:.4f}")
```

```
Chance accuracy with a shuffle: 0.5037
```

[79]:
```
X_train, X_test, y_train, y_test = train_test_split(features, labels,␣
 ↪test_size=0.3, random_state=42)
logistic_classifier = LogisticRegression(random_state=42)
logistic_classifier.fit(X_train, y_train)
y_pred = logistic_classifier.predict(X_test)
actual_test_accuracy = accuracy_score(y_test, y_pred)
n_repetitions = 1500
chance_accuracies = []
for i in range(n_repetitions):
    y_test_shuffled = np.random.permutation(y_test)
```

```
    chance_acc = accuracy_score(y_test_shuffled, y_pred)
    chance_accuracies.append(chance_acc)
plt.figure(figsize=(10, 6))
plt.hist(chance_accuracies, bins=30, density=True, alpha=0.7,␣
 ↪color='lightgray', label='Chance Accuracy Distribution')
plt.axvline(actual_test_accuracy, color='red', linestyle='dashed', linewidth=2,␣
 ↪label=f'Actual Model Accuracy ({actual_test_accuracy:.4f})')
plt.title(f'Distribution of Chance Accuracies ({n_repetitions} times)')
plt.xlabel('Accuracy')
plt.ylabel('D')
plt.legend()
plt.grid(axis='y', alpha=0.75)
plt.show()
mean_chance_accuracy = np.mean(chance_accuracies)
print(f"\nMean : {mean_chance_accuracy:.4f}")
```




```
Mean : 0.5096
```

[ ]:

[98]:
```
try:
    features = np.load('features.npy')
    labels = np.load('labels.npy')
```

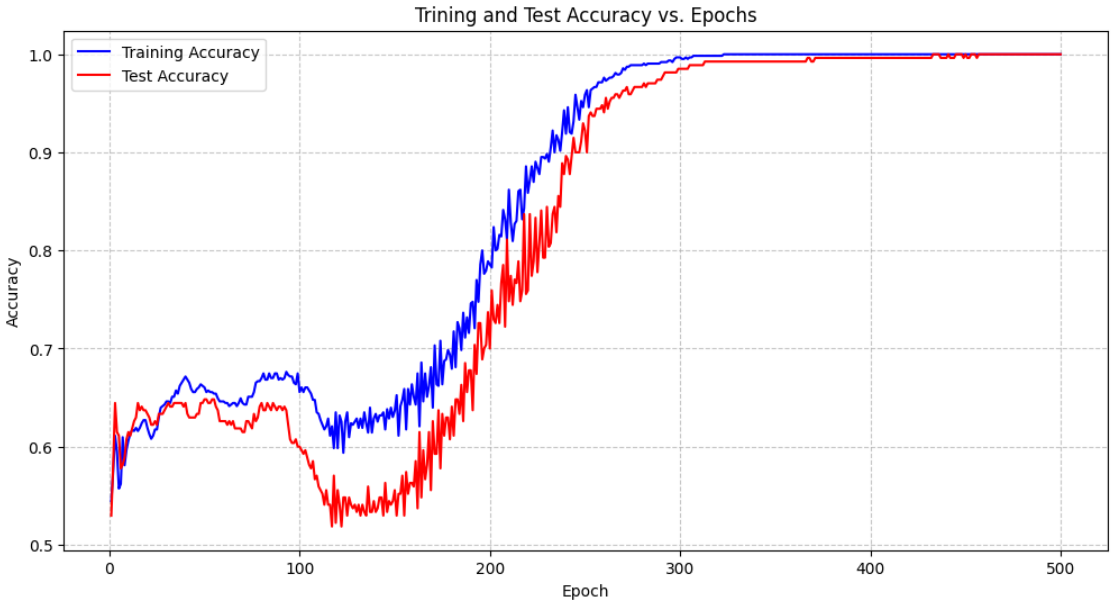
```
except FileNotFoundError:
    print("Error.")
    from sklearn.datasets import make_circles
    features, labels = make_circles(n_samples=900, factor=.5, noise=.05,␣
 ↪random_state=42)
    features = features * 10
X_train, X_test, y_train, y_test = train_test_split(features, labels,␣
 ↪test_size=0.3, random_state=42)
X_train_tensor = torch.FloatTensor(X_train)
y_train_tensor = torch.LongTensor(y_train)
X_test_tensor = torch.FloatTensor(X_test)
y_test_tensor = torch.LongTensor(y_test)
class SimpleNN(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        super(SimpleNN, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.fc2 = nn.Linear(hidden_size, hidden_size)
        self.fc3 = nn.Linear(hidden_size, num_classes)
    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
input_size = X_train.shape[1]
hidden_size = 64
num_classes = len(np.unique(labels))
learning_rate = 0.01
num_epochs = 500
model = SimpleNN(input_size, hidden_size, num_classes)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate)
print(model)
train_losses = []
train_accuracies = []
test_accuracies = []
print("\n training...")
for epoch in range(num_epochs):
    model.train()
    outputs = model(X_train_tensor)
    loss = criterion(outputs, y_train_tensor)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
    train_losses.append(loss.item())
    with torch.no_grad():
        _, predicted_train = torch.max(outputs.data, 1)
```

```
        train_acc = accuracy_score(y_train_tensor.numpy(), predicted_train.
 ↪numpy())
        train_accuracies.append(train_acc)
    model.eval()
    with torch.no_grad():
        test_outputs = model(X_test_tensor)
        _, predicted_test = torch.max(test_outputs.data, 1)
        test_acc = accuracy_score(y_test_tensor.numpy(), predicted_test.numpy())
        test_accuracies.append(test_acc)
    if (epoch + 1) % 50 == 0:
        print(f'Epoch [{epoch+1}/{num_epochs}], Train Loss: {loss.item():.4f},␣
 ↪Train Acc: {train_acc:.4f}, Test Acc: {test_acc:.4f}')
with torch.no_grad():
    test_outputs = model(X_test_tensor)
    _, predicted = torch.max(test_outputs.data, 1)
    accuracy = accuracy_score(y_test_tensor.numpy(), predicted.numpy())
print(f"\nFinal Classifcation accuracy (Neural Network): {accuracy:.4f}")
plt.figure(figsize=(12, 6))
plt.plot(range(1, num_epochs + 1), train_accuracies, label='Training Accuracy',␣
 ↪color='blue')
plt.plot(range(1, num_epochs + 1), test_accuracies, label='Test Accuracy',␣
 ↪color='red')
plt.title('Trining and Test Accuracy vs. Epochs')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True, linestyle='--', alpha=0.7)
plt.show()
```

```
SimpleNN(
  (fc1): Linear(in_features=2, out_features=64, bias=True)
  (fc2): Linear(in_features=64, out_features=64, bias=True)
  (fc3): Linear(in_features=64, out_features=2, bias=True)
)
```

```
 training…
Epoch [50/500], Train Loss: 0.6059, Train Acc: 0.6603, Test Acc: 0.6481
Epoch [100/500], Train Loss: 0.5692, Train Acc: 0.6556, Test Acc: 0.6000
Epoch [150/500], Train Loss: 0.5308, Train Acc: 0.6365, Test Acc: 0.5556
Epoch [200/500], Train Loss: 0.4235, Train Acc: 0.7857, Test Acc: 0.7000
Epoch [250/500], Train Loss: 0.1821, Train Acc: 0.9587, Test Acc: 0.9222
Epoch [300/500], Train Loss: 0.0494, Train Acc: 0.9968, Test Acc: 0.9852
Epoch [350/500], Train Loss: 0.0190, Train Acc: 1.0000, Test Acc: 0.9926
Epoch [400/500], Train Loss: 0.0104, Train Acc: 1.0000, Test Acc: 0.9963
Epoch [450/500], Train Loss: 0.0068, Train Acc: 1.0000, Test Acc: 1.0000
Epoch [500/500], Train Loss: 0.0049, Train Acc: 1.0000, Test Acc: 1.0000
```

```
Final Classifcation accuracy (Neural Network): 1.0000
```



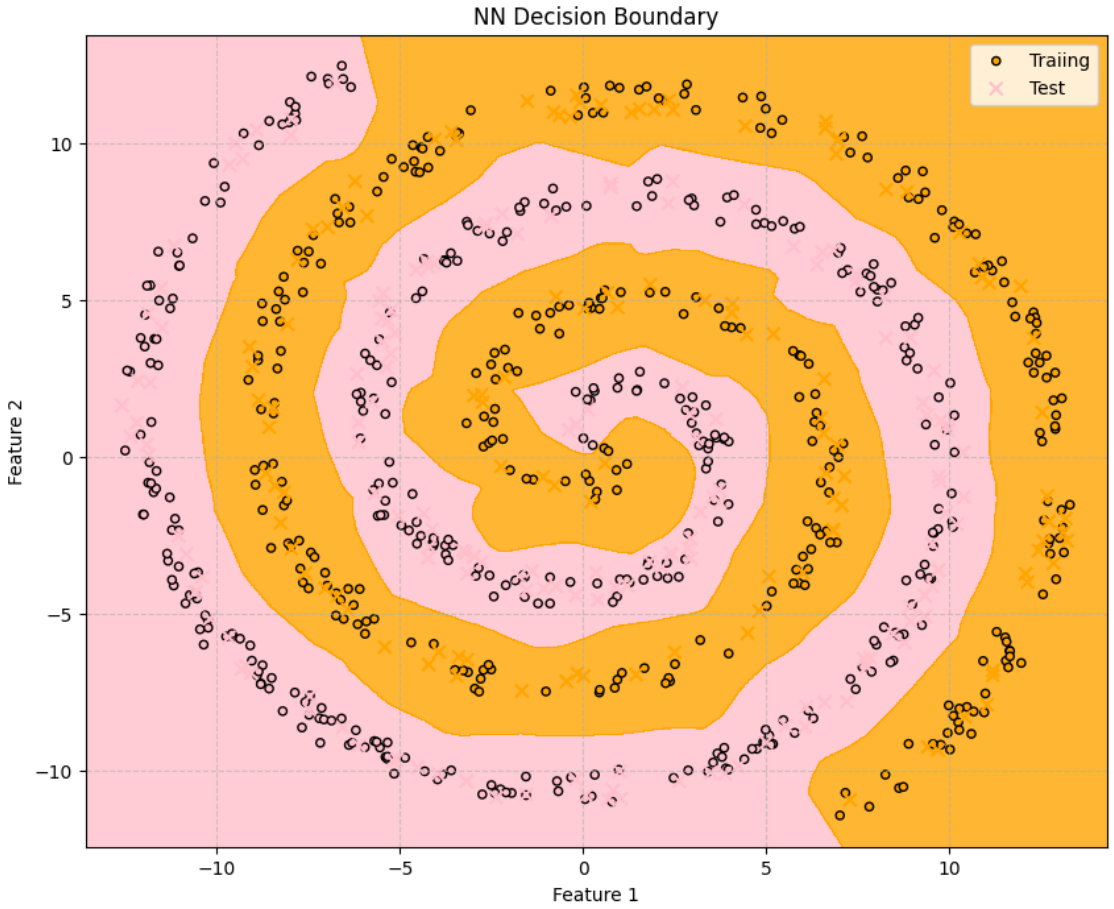
[ ]:

[81]:
```
try:
    features = np.load('features.npy')
    labels = np.load('labels.npy')
except FileNotFoundError:
    from sklearn.datasets import make_circles
    features, labels = make_circles(n_samples=900, factor=.5, noise=.05,␣
 ↪random_state=42)
    features = features * 10
X_train, X_test, y_train, y_test = train_test_split(features, labels,␣
 ↪test_size=0.3, random_state=42)
X_train_tensor = torch.FloatTensor(X_train)
y_train_tensor = torch.LongTensor(y_train)
X_test_tensor = torch.FloatTensor(X_test)
y_test_tensor = torch.LongTensor(y_test)
class SimpleNN(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        super(SimpleNN, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.fc2 = nn.Linear(hidden_size, hidden_size)
        self.fc3 = nn.Linear(hidden_size, num_classes)
    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
```

```
        x = self.fc3(x)
        return x
optimal_hidden_size = 64
optimal_learning_rate = 0.01
optimal_num_epochs = 300
input_size = X_train.shape[1]
num_classes = len(np.unique(labels))
final_model = SimpleNN(input_size, optimal_hidden_size, num_classes)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(final_model.parameters(), lr=optimal_learning_rate)
final_model.train()
for epoch in range(optimal_num_epochs):
    outputs = final_model(X_train_tensor)
    loss = criterion(outputs, y_train_tensor)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
x_min, x_max = features[:, 0].min() - 1, features[:, 0].max() + 1
y_min, y_max = features[:, 1].min() - 1, features[:, 1].max() + 1
h = .02
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
mesh_tensor = torch.FloatTensor(np.c_[xx.ravel(), yy.ravel()])
final_model.eval()
with torch.no_grad():
    Z_logits = final_model(mesh_tensor)
    _, Z = torch.max(Z_logits.data, 1)
    Z = Z.numpy().reshape(xx.shape)
colors = ['pink', 'orange']
cmap_custom = ListedColormap(colors)
plt.figure(figsize=(10, 8))
plt.contourf(xx, yy, Z, cmap=cmap_custom, alpha=0.8)
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap=cmap_custom,␣
 ↪edgecolors='k', marker='o', s=20, label='Traiing')
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap=cmap_custom,␣
 ↪edgecolors='k', marker='x', s=50, label='Test')
plt.title('NN Decision Boundary')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.grid(True, linestyle='--', alpha=0.6)
plt.show()
```

```
/tmp/ipykernel_7020/3946152375.py:63: UserWarning: You passed a
edgecolor/edgecolors ('k') for an unfilled marker ('x').  Matplotlib is ignoring
the edgecolor in favor of the facecolor.  This behavior may change in the
future.
  plt.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap=cmap_custom,
```

```
edgecolors='k', marker='x', s=50, label='Test')
```



[ ]:

[82]:
```
print(f"Dataset: {dataset}")
print(f"no of graphs: {len(dataset)}")
```

```
Dataset: MUTAG(188)
no of graphs: 188
```

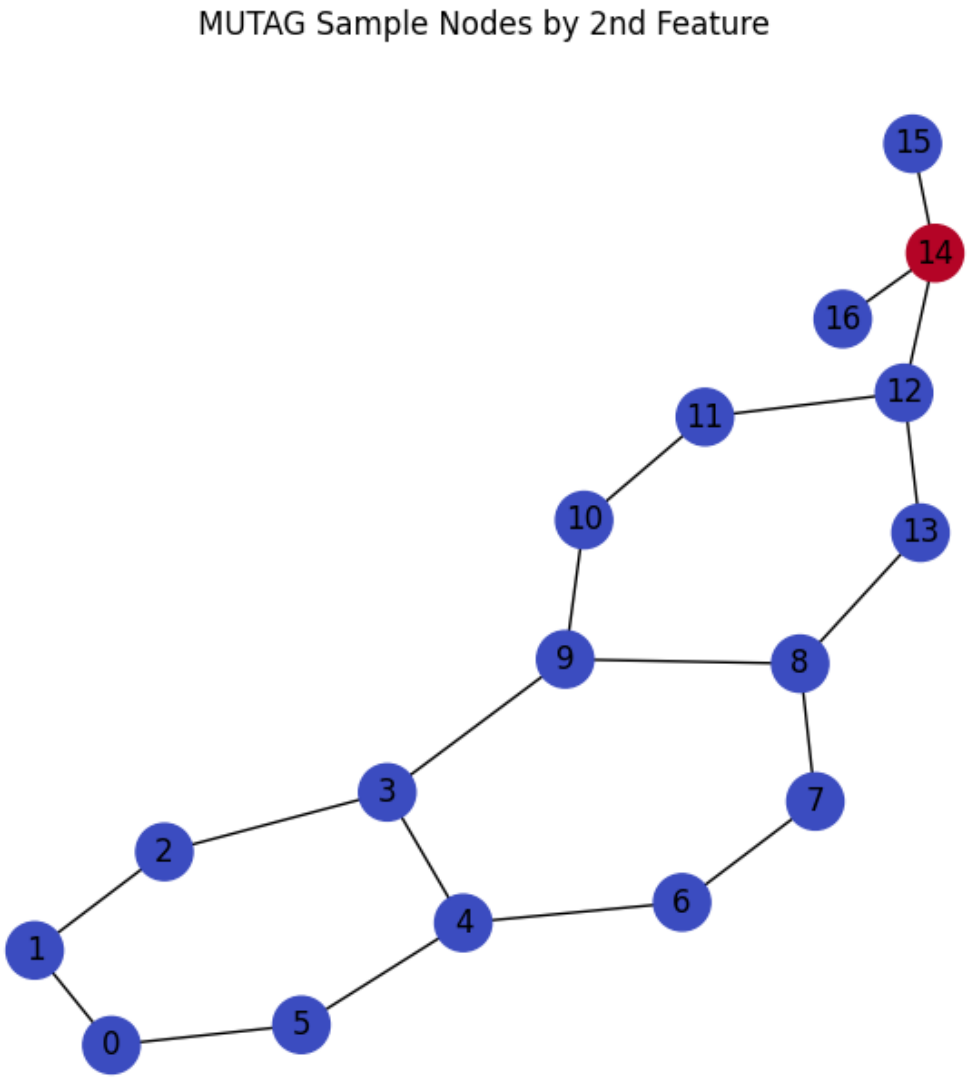
[67]:
```
samples = len(dataset)
classes = dataset.num_classes
print(samples)
print(classes)
```

```
188
2
```

[68]:
```
first_graph = dataset[0]
num_nodes_first_sample = first_graph.num_nodes
num_edges_first_sample = first_graph.num_edges
print(f"Number of nodes: {num_nodes_first_sample}")
print(f"Number of edges: {num_edges_first_sample}")
```

```
Number of nodes: 17
Number of edges: 38
```

[95]:
```
data = dataset[0]
G = to_networkx(data, to_undirected=True)
if data.x is not None and data.x.size(1) > 1:
    color_values = data.x[:, 1].tolist()
else:
    color_values = 'blue'
plt.figure(figsize=(6, 6))
nx.draw(
    G,
    with_labels=True,
    node_color=color_values,
    cmap=plt.cm.coolwarm,
    node_size=500
)
plt.title("MUTAG Sample Nodes by 2nd Feature")
plt.show()
```



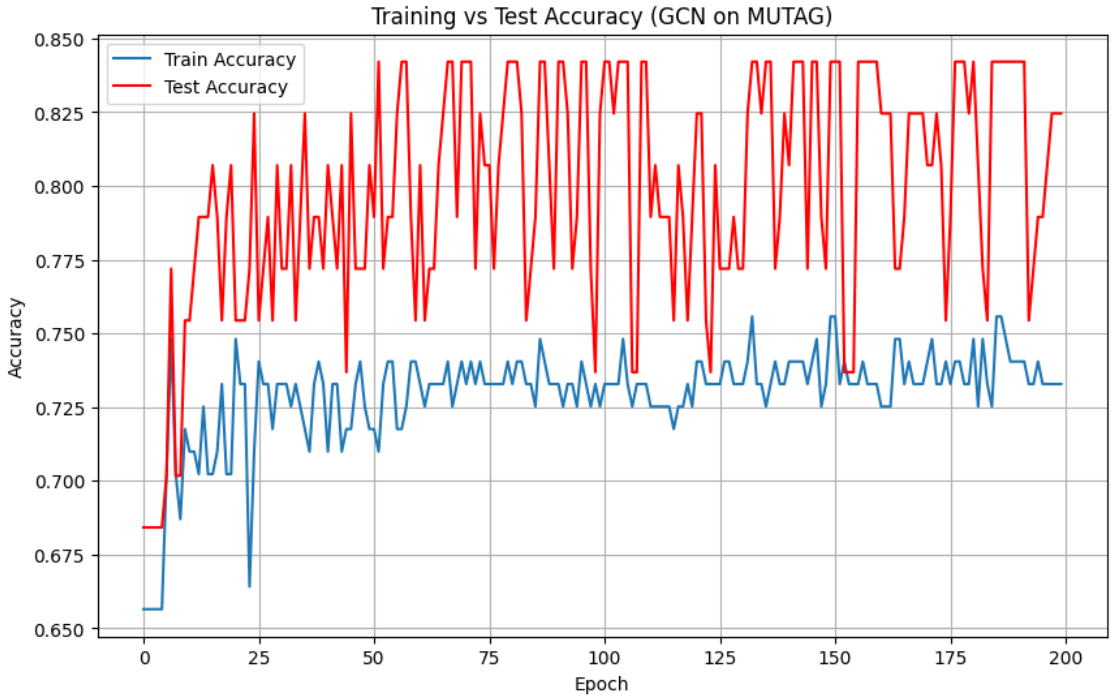
[84]:
```
train_indices, test_indices = train_test_split(range(len(dataset)), test_size=0.
↪3, random_state=42)
train_dataset = dataset[train_indices]
test_dataset = dataset[test_indices]
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)
class GCN(torch.nn.Module):
    def __init__(self, num_node_features, hidden_channels, num_classes):
        super(GCN, self).__init__()
        self.conv1 = GCNConv(num_node_features, hidden_channels)
        self.conv2 = GCNConv(hidden_channels, hidden_channels)
        self.lin = torch.nn.Linear(hidden_channels, num_classes)
```

```
    def forward(self, x, edge_index, batch):
        x = F.relu(self.conv1(x, edge_index))
        x = F.relu(self.conv2(x, edge_index))
        x = global_mean_pool(x, batch)
        x = self.lin(x)
        return x
model = GCN(num_node_features=dataset.num_node_features, hidden_channels=64,␣
 ↪num_classes=2)
optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
criterion = torch.nn.CrossEntropyLoss()
def train():
    model.train()
    total_loss = 0
    for data in train_loader:
        optimizer.zero_grad()
        out = model(data.x, data.edge_index, data.batch)
        loss = criterion(out, data.y)
        loss.backward()
        optimizer.step()
        total_loss += loss.item()
    return total_loss / len(train_loader)
def test(loader):
    model.eval()
    preds, labels = [], []
    with torch.no_grad():
        for data in loader:
            out = model(data.x, data.edge_index, data.batch)
            pred = out.argmax(dim=1)
            preds.extend(pred.tolist())
            labels.extend(data.y.tolist())
    return accuracy_score(labels, preds), preds, labels
train_accs, test_accs = [], []
for epoch in range(1, 201):
    train_loss = train()
    train_acc, _, _ = test(train_loader)
    test_acc, _, _ = test(test_loader)
    train_accs.append(train_acc)
    test_accs.append(test_acc)
    if epoch % 20 == 0:
        print(f'Epoch {epoch}, Train Acc: {train_acc:.4f}, Test Acc: {test_acc:.
 ↪4f}')
```

```
Epoch 20, Train Acc: 0.7328, Test Acc: 0.7193
Epoch 40, Train Acc: 0.7405, Test Acc: 0.7895
Epoch 60, Train Acc: 0.7252, Test Acc: 0.8070
Epoch 80, Train Acc: 0.7328, Test Acc: 0.8070
Epoch 100, Train Acc: 0.7405, Test Acc: 0.8421
```

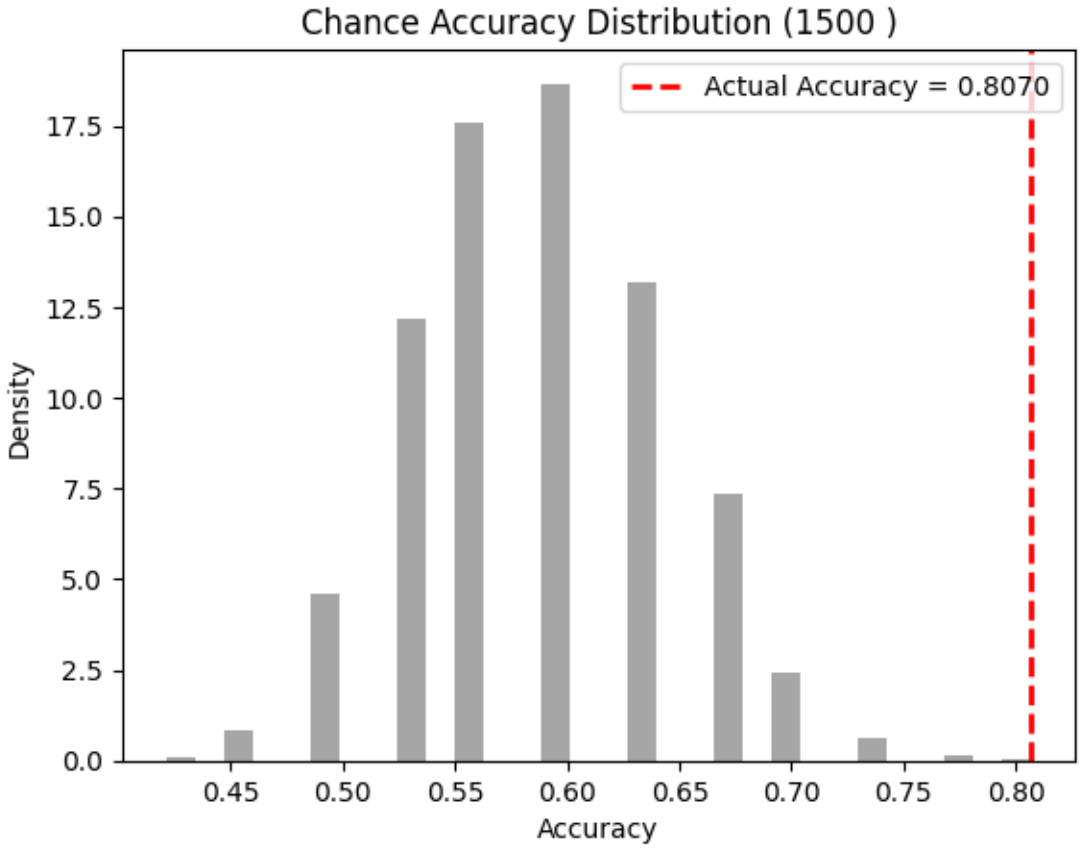
```
Epoch 120, Train Acc: 0.7252, Test Acc: 0.7719
Epoch 140, Train Acc: 0.7405, Test Acc: 0.8421
Epoch 160, Train Acc: 0.7405, Test Acc: 0.8421
Epoch 180, Train Acc: 0.7405, Test Acc: 0.7895
Epoch 200, Train Acc: 0.7557, Test Acc: 0.8246
```

[44]:
```python
plt.figure(figsize=(10, 6))
plt.plot(train_accs, label='Train Accuracy')
plt.plot(test_accs, label='Test Accuracy', color='red')
plt.title('Training vs Test Accuracy (GCN on MUTAG)')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)
plt.show()
```



[94]:
```python
_, y_pred, y_true = test(test_loader)
y_true = np.array(y_true)
y_pred = np.array(y_pred)
n_repetitions = 1500
chance_accuracies = []
for _ in range(n_repetitions):
    shuffled = np.random.permutation(y_true)
    acc = accuracy_score(shuffled, y_pred)
```

```
    chance_accuracies.append(acc)
actual_acc = accuracy_score(y_true, y_pred)
plt.hist(chance_accuracies, bins=30, color='gray', alpha=0.7, density=True)
plt.axvline(actual_acc, color='red', linestyle='--', linewidth=2,␣
 ↪label=f'Actual Accuracy = {actual_acc:.4f}')
plt.xlabel('Accuracy')
plt.ylabel('Density')
plt.title(f'Chance Accuracy Distribution ({n_repetitions} )')
plt.legend()
plt.show()
```



[93]:
```
train_indices, test_indices = train_test_split(range(len(dataset)), test_size=0.
 ↪3, random_state=42)
train_dataset = dataset[train_indices]
test_dataset = dataset[test_indices]
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)
def get_graph_embeddings(dataset):
    features = []
```

```
    labels = []
    for data in dataset:
        x_mean = data.x.mean(dim=0).numpy()
        features.append(x_mean)
        labels.append(data.y.item())
    return np.array(features), np.array(labels)

X_train, y_train = get_graph_embeddings(train_dataset)
X_test, y_test = get_graph_embeddings(test_dataset)

logreg = LogisticRegression(max_iter=1000)
logreg.fit(X_train, y_train)
baseline_test_acc = logreg.score(X_test, y_test)
print(f"Baseline Test Accuracy: {baseline_test_acc:.4f}")
```

```
Baseline Test Accuracy: 0.6842
```

[ ]: