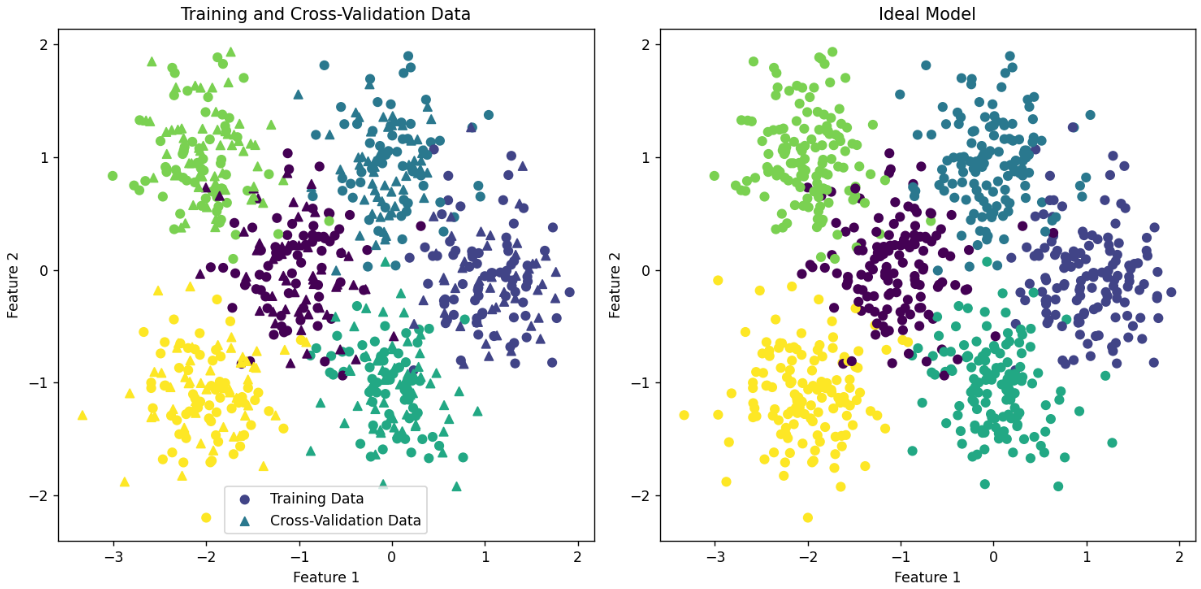
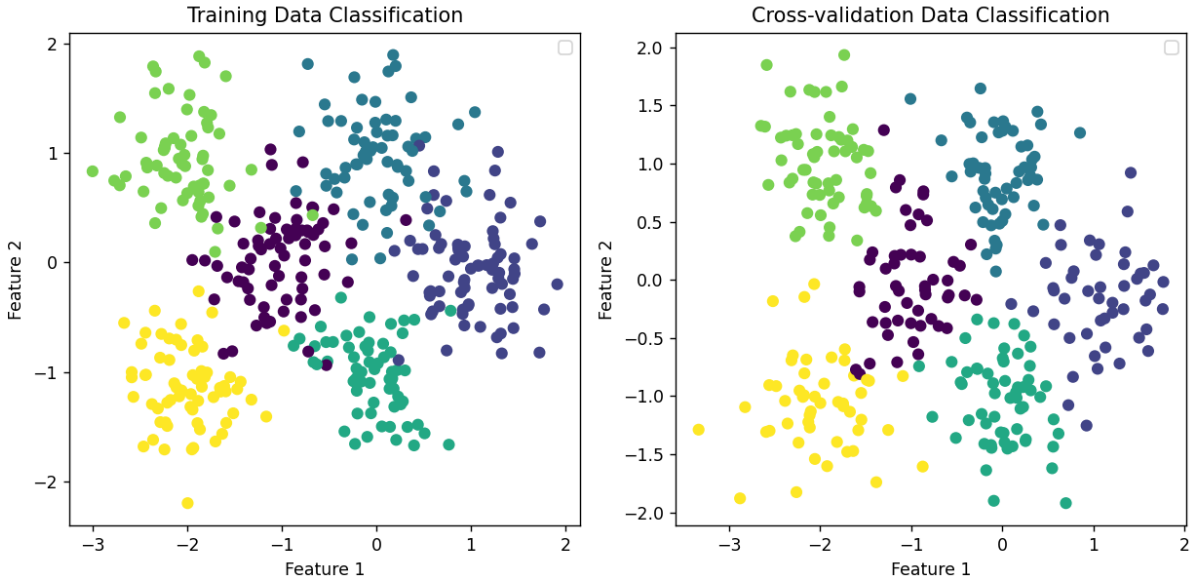
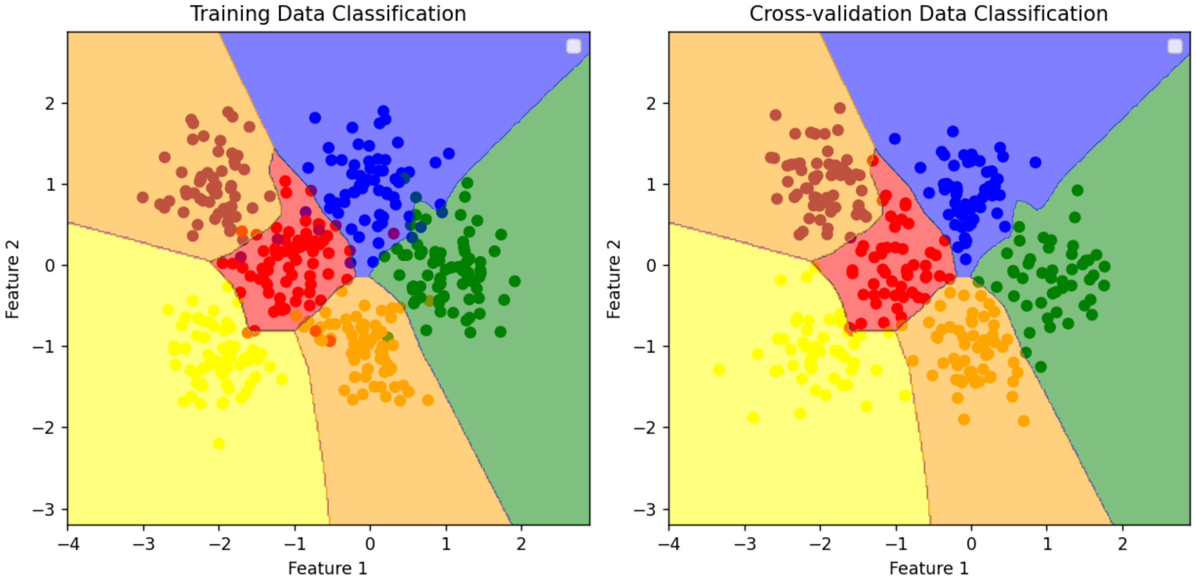
Part 1.1

- This is our traning data, cv data and ideal data

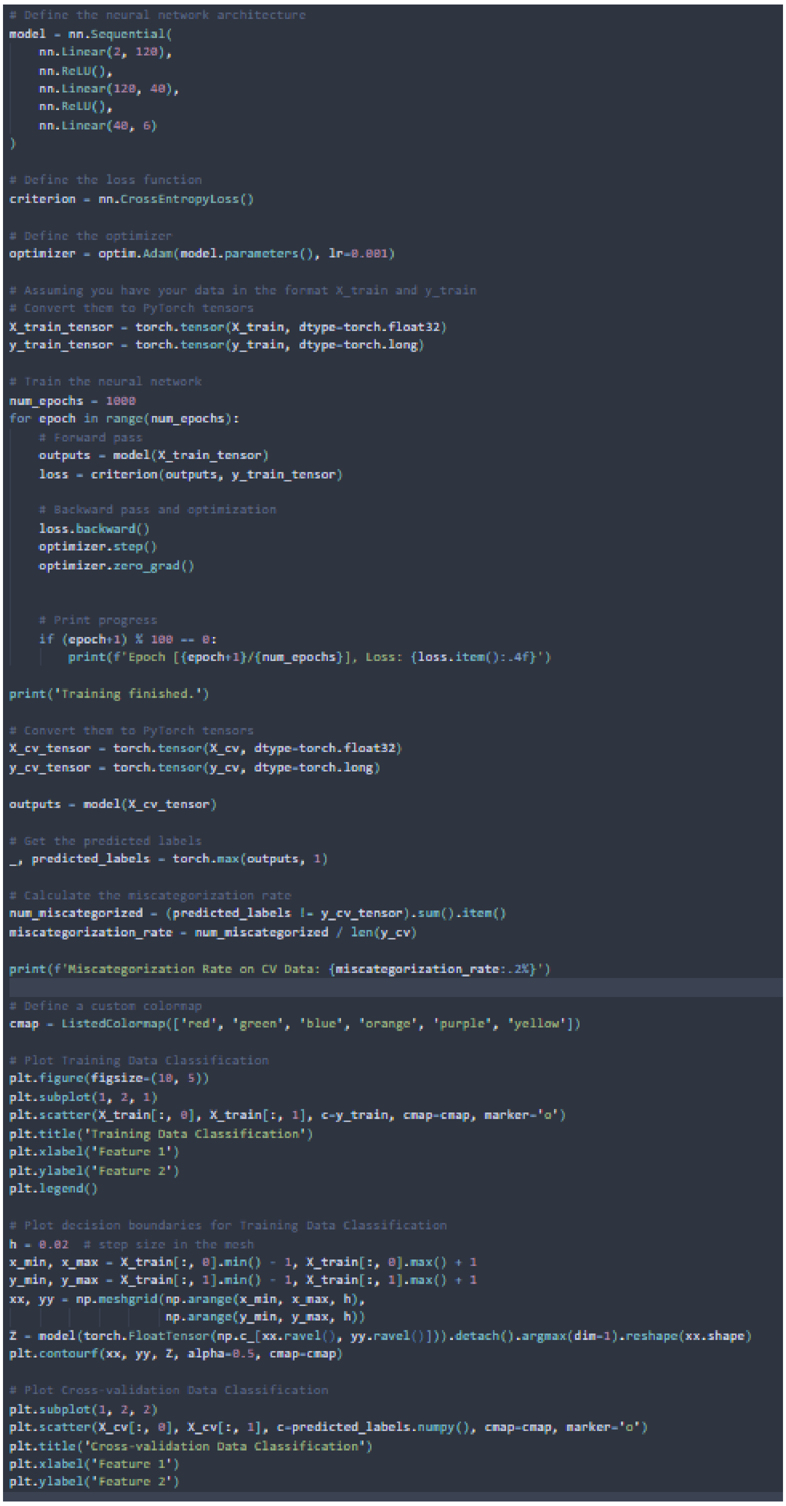




- This is how are they classified

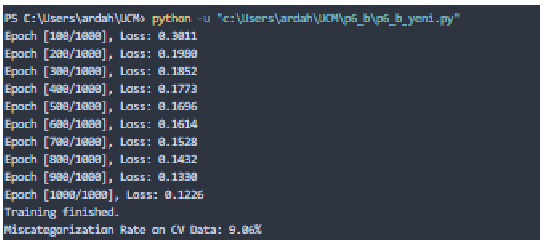


Part 1.2



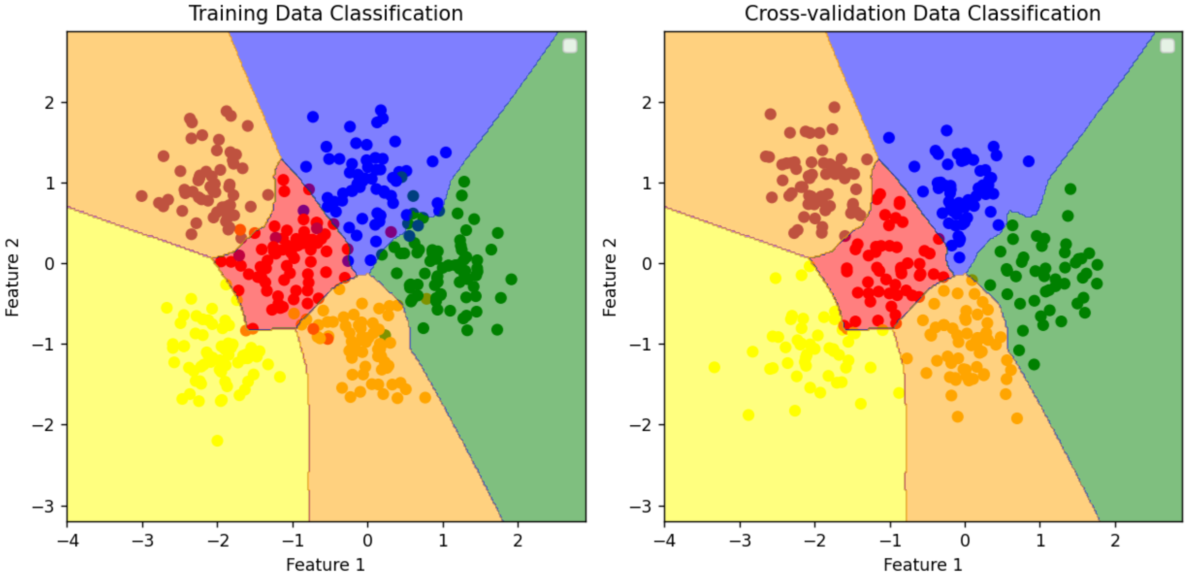
```python
# Define the neural network architecture
model = nn.Sequential(
    nn.Linear(2, 120),
    nn.ReLU(),
    nn.Linear(120, 40),
    nn.ReLU(),
    nn.Linear(40, 6)
)

# Define the loss function
criterion = nn.CrossEntropyLoss()

# Define the optimizer
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Assuming you have your data in the format X_train and y_train
# Convert them to PyTorch tensors
X_train_tensor = torch.tensor(X_train, dtype=torch.float32)
y_train_tensor = torch.tensor(y_train, dtype=torch.long)

# Train the neural network
num_epochs = 1000
for epoch in range(num_epochs):
    # Forward pass
    outputs = model(X_train_tensor)
    loss = criterion(outputs, y_train_tensor)

    # Backward pass and optimization
    loss.backward()
    optimizer.step()
    optimizer.zero_grad()


    # Print progress
    if (epoch+1) % 100 == 0:
        print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}')

print('Training finished.')

# Convert them to PyTorch tensors
X_cv_tensor = torch.tensor(X_cv, dtype=torch.float32)
y_cv_tensor = torch.tensor(y_cv, dtype=torch.long)

outputs = model(X_cv_tensor)

# Get the predicted labels
_, predicted_labels = torch.max(outputs, 1)

# Calculate the miscategorization rate
num_miscategorized = (predicted_labels != y_cv_tensor).sum().item()
miscategorization_rate = num_miscategorized / len(y_cv)

print(f'Miscategorization Rate on CV Data: {miscategorization_rate:.2%}')

# Define a custom colormap
cmap = ListedColormap(['red', 'green', 'blue', 'orange', 'purple', 'yellow'])

# Plot Training Data Classification
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap=cmap, marker='o')
plt.title('Training Data Classification')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()

# Plot decision boundaries for Training Data Classification
h = 0.02  # step size in the mesh
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, h),
                     np.arange(y_min, y_max, h))
Z = model(torch.FloatTensor(np.c_[xx.ravel(), yy.ravel()])).detach().argmax(dim=1).reshape(xx.shape)
plt.contourf(xx, yy, Z, alpha=0.5, cmap=cmap)

# Plot Cross-validation Data Classification
plt.subplot(1, 2, 2)
plt.scatter(X_cv[:, 0], X_cv[:, 1], c=predicted_labels.numpy(), cmap=cmap, marker='o')
plt.title('Cross-validation Data Classification')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
```

- Error rate doesn’t go up to 12% but varies betweeb 8-10%

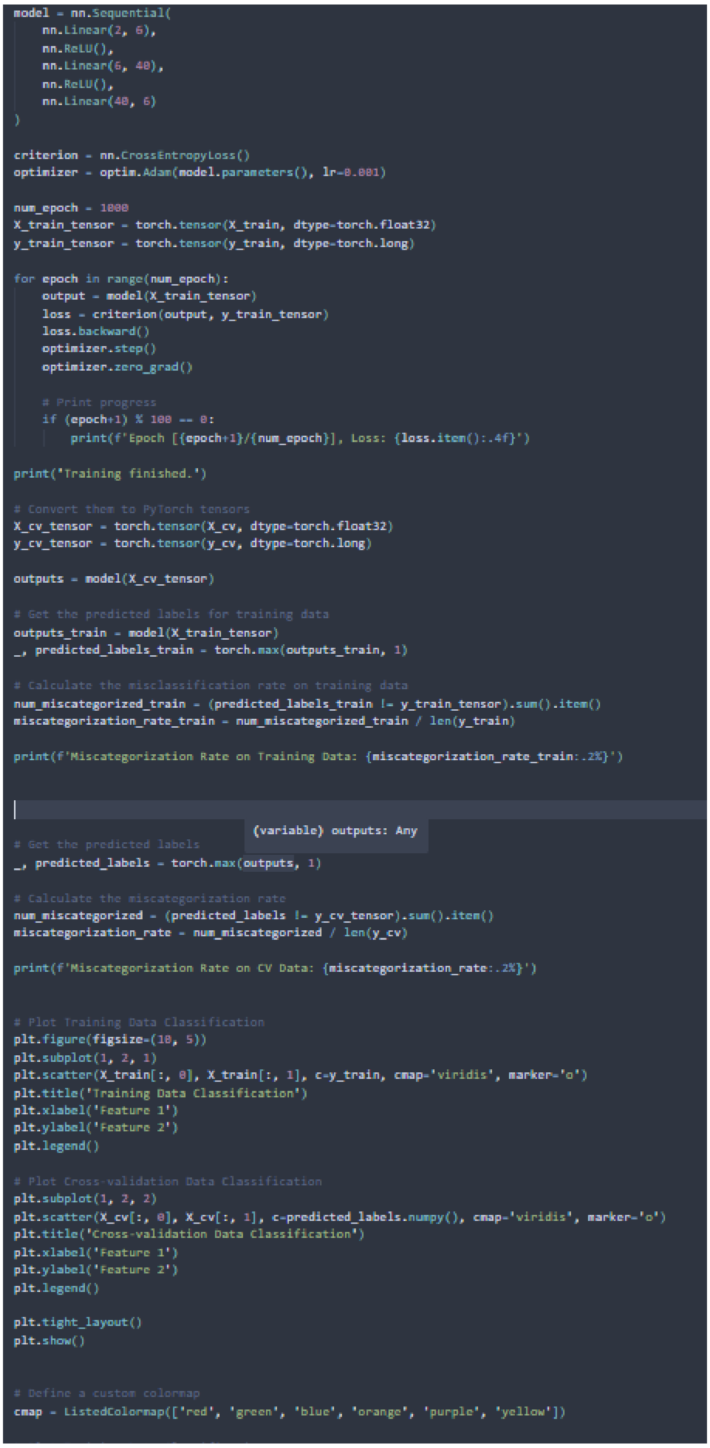


```
PS C:\Users\ardah\UCM> python -u "c:\Users\ardah\UCM\p6_b\p6_b_yeni.py"
Epoch [100/1000], Loss: 0.3011
Epoch [200/1000], Loss: 0.1980
Epoch [300/1000], Loss: 0.1852
Epoch [400/1000], Loss: 0.1773
Epoch [500/1000], Loss: 0.1696
Epoch [600/1000], Loss: 0.1614
Epoch [700/1000], Loss: 0.1528
Epoch [800/1000], Loss: 0.1432
Epoch [900/1000], Loss: 0.1330
Epoch [1000/1000], Loss: 0.1226
Training finished.
Miscategorization Rate on CV Data: 9.06%
```

- This is how ComplexModel classifies

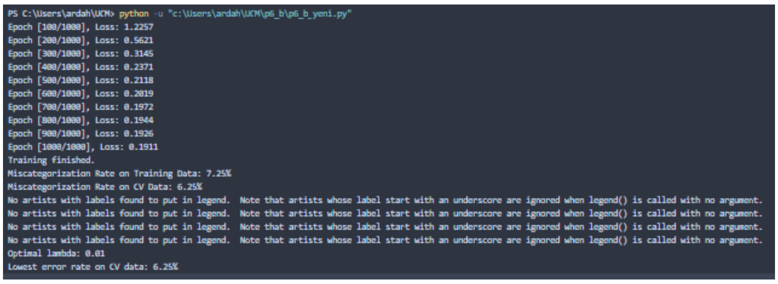


Part 1.3



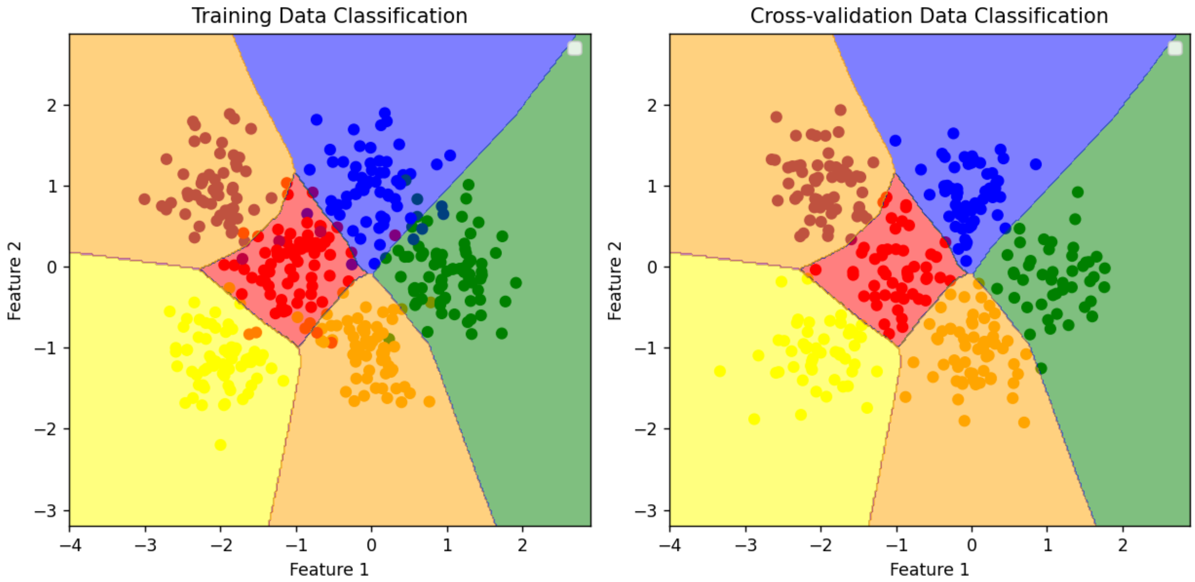
```python
model = nn.Sequential(
    nn.Linear(2, 6),
    nn.ReLU(),
    nn.Linear(6, 40),
    nn.ReLU(),
    nn.Linear(40, 6)
)

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

num_epoch = 1000
X_train_tensor = torch.tensor(X_train, dtype=torch.float32)
y_train_tensor = torch.tensor(y_train, dtype=torch.long)

for epoch in range(num_epoch):
    output = model(X_train_tensor)
    loss = criterion(output, y_train_tensor)
    loss.backward()
    optimizer.step()
    optimizer.zero_grad()

    # Print progress
    if (epoch+1) % 100 == 0:
        print(f'Epoch [{epoch+1}/{num_epoch}], Loss: {loss.item():.4f}')

print('Training finished.')

# Convert them to PyTorch tensors
X_cv_tensor = torch.tensor(X_cv, dtype=torch.float32)
y_cv_tensor = torch.tensor(y_cv, dtype=torch.long)

outputs = model(X_cv_tensor)

# Get the predicted labels for training data
outputs_train = model(X_train_tensor)
_, predicted_labels_train = torch.max(outputs_train, 1)

# Calculate the misclassification rate on training data
num_miscategorized_train = (predicted_labels_train != y_train_tensor).sum().item()
miscategorization_rate_train = num_miscategorized_train / len(y_train)

print(f'Miscategorization Rate on Training Data: {miscategorization_rate_train:.2%}')


                                     (variable) outputs: Any
# Get the predicted labels
_, predicted_labels = torch.max(outputs, 1)

# Calculate the miscategorization rate
num_miscategorized = (predicted_labels != y_cv_tensor).sum().item()
miscategorization_rate = num_miscategorized / len(y_cv)

print(f'Miscategorization Rate on CV Data: {miscategorization_rate:.2%}')


# Plot Training Data Classification
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap='viridis', marker='o')
plt.title('Training Data Classification')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()

# Plot Cross-validation Data Classification
plt.subplot(1, 2, 2)
plt.scatter(X_cv[:, 0], X_cv[:, 1], c=predicted_labels.numpy(), cmap='viridis', marker='o')
plt.title('Cross-validation Data Classification')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()

plt.tight_layout()
plt.show()


# Define a custom colormap
cmap = ListedColormap(['red', 'green', 'blue', 'orange', 'purple', 'yellow'])
```

- This is the error rates I get, which are same with the assignment

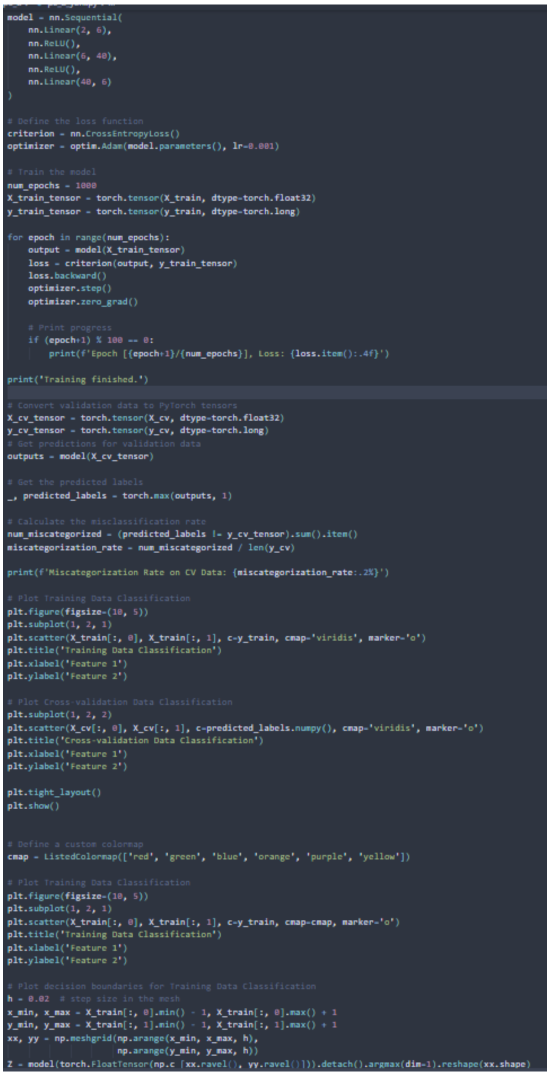


```
PS C:\Users\ardah\UCM> python -u "c:\Users\ardah\UCM\p6_b\p6_b_yeni.py"
Epoch [100/1000], Loss: 1.2257
Epoch [200/1000], Loss: 0.5621
Epoch [300/1000], Loss: 0.3145
Epoch [400/1000], Loss: 0.2371
Epoch [500/1000], Loss: 0.2118
Epoch [600/1000], Loss: 0.2019
Epoch [700/1000], Loss: 0.1972
Epoch [800/1000], Loss: 0.1944
Epoch [900/1000], Loss: 0.1926
Epoch [1000/1000], Loss: 0.1911
Training finished.
Miscategorization Rate on Training Data: 7.25%
Miscategorization Rate on CV Data: 6.25%
No artists with labels found to put in legend.  Note that artists whose label start with an underscore are ignored when legend() is called with no argument.
No artists with labels found to put in legend.  Note that artists whose label start with an underscore are ignored when legend() is called with no argument.
No artists with labels found to put in legend.  Note that artists whose label start with an underscore are ignored when legend() is called with no argument.
No artists with labels found to put in legend.  Note that artists whose label start with an underscore are ignored when legend() is called with no argument.
Optimal lambda: 0.01
Lowest error rate on CV data: 6.25%
```

- This is how simple model clasifies
- As it can be seen, it is a very close shape to ideal

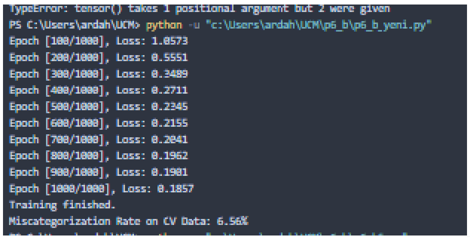


Part 1.4

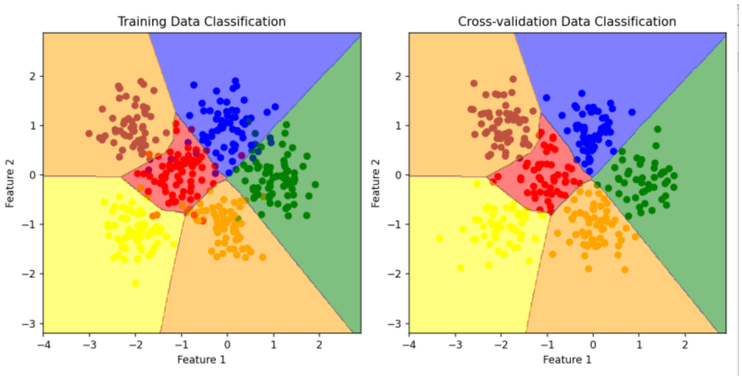


```
model = nn.Sequential(
    nn.Linear(2, 6),
    nn.ReLU(),
    nn.Linear(6, 40),
    nn.ReLU(),
    nn.Linear(40, 6)
)

# Define the loss function
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Train the model
num_epochs = 1000
X_train_tensor = torch.tensor(X_train, dtype=torch.float32)
y_train_tensor = torch.tensor(y_train, dtype=torch.long)

for epoch in range(num_epochs):
    output = model(X_train_tensor)
    loss = criterion(output, y_train_tensor)
    loss.backward()
    optimizer.step()
    optimizer.zero_grad()

    # Print progress
    if (epoch+1) % 100 == 0:
        print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}')

print('Training finished.')

# Convert validation data to PyTorch tensors
X_cv_tensor = torch.tensor(X_cv, dtype=torch.float32)
y_cv_tensor = torch.tensor(y_cv, dtype=torch.long)
# Get predictions for validation data
outputs = model(X_cv_tensor)

# Get the predicted labels
_, predicted_labels = torch.max(outputs, 1)

# Calculate the misclassification rate
num_miscategorized = (predicted_labels != y_cv_tensor).sum().item()
miscategorization_rate = num_miscategorized / len(y_cv)

print(f'Miscategorization Rate on CV Data: {miscategorization_rate:.2%}')

# Plot Training Data Classification
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap='viridis', marker='o')
plt.title('Training Data Classification')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')

# Plot Cross-validation Data Classification
plt.subplot(1, 2, 2)
plt.scatter(X_cv[:, 0], X_cv[:, 1], c=predicted_labels.numpy(), cmap='viridis', marker='o')
plt.title('Cross-validation Data Classification')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')

plt.tight_layout()
plt.show()


# Define a custom colormap
cmap = ListedColormap(['red', 'green', 'blue', 'orange', 'purple', 'yellow'])

# Plot Training Data Classification
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap=cmap, marker='o')
plt.title('Training Data Classification')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')

# Plot decision boundaries for Training Data Classification
h = 0.02  # step size in the mesh
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, h),
                     np.arange(y_min, y_max, h))
Z = model(torch.FloatTensor(np.c_[xx.ravel(), yy.ravel()])).detach().argmax(dim=1).reshape(xx.shape)
```

- The complex model also looks much better and much close to the ideal model

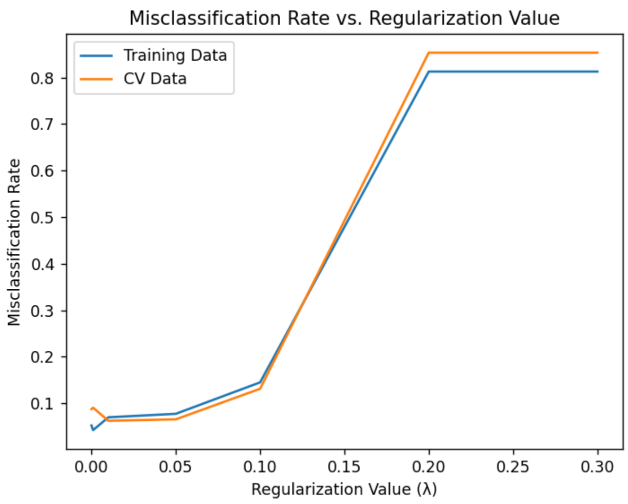


```
TypeError: tensor() takes 1 positional argument but 2 were given
PS C:\Users\ardah\UCM> python -u "c:\Users\ardah\UCM\p6_b\p6_b_yeni.py"
Epoch [100/1000], Loss: 1.0573
Epoch [200/1000], Loss: 0.5551
Epoch [300/1000], Loss: 0.3489
Epoch [400/1000], Loss: 0.2711
Epoch [500/1000], Loss: 0.2345
Epoch [600/1000], Loss: 0.2155
Epoch [700/1000], Loss: 0.2041
Epoch [800/1000], Loss: 0.1962
Epoch [900/1000], Loss: 0.1901
Epoch [1000/1000], Loss: 0.1857
Training finished.
Miscategorization Rate on CV Data: 6.56%
```



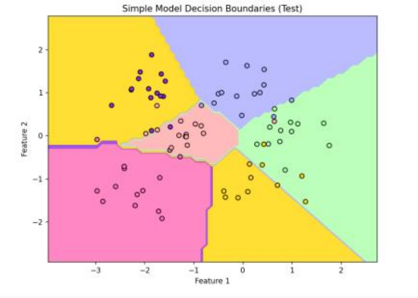
Part 1.5

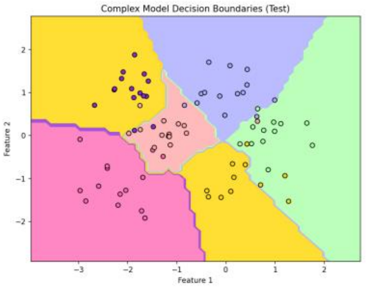
- The graph I got is much different than how it should be, I wasn't able to solve it

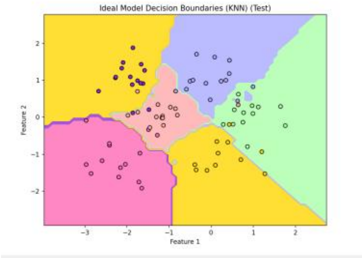


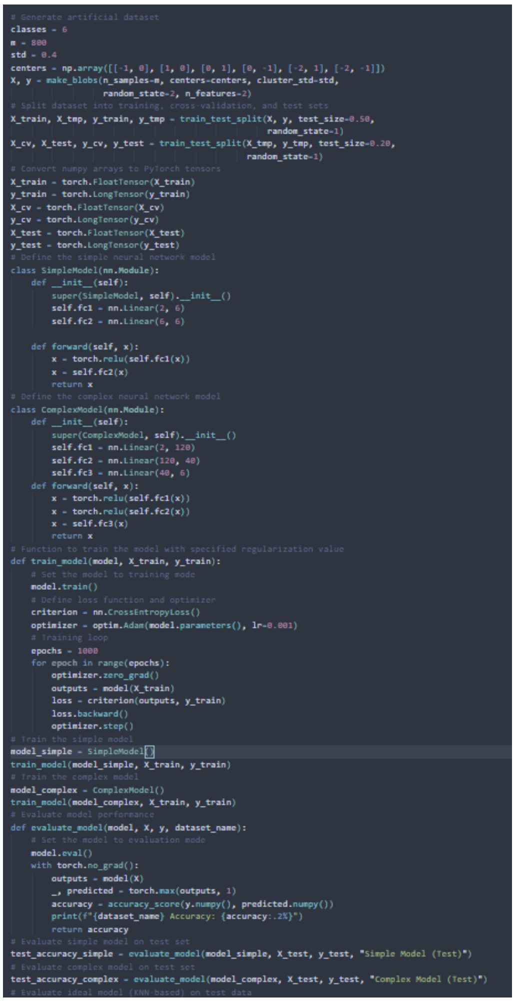
Part 1.6

- Here are the classifiactions I got from the test class



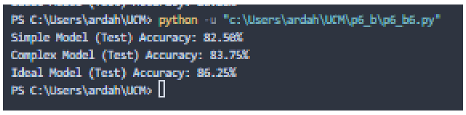






```
# Generate artificial dataset
classes = 6
m = 800
std = 0.4
centers = np.array([[-1, 0], [1, 0], [0, 1], [0, -1], [-2, 1], [-2, -1]])
X, y = make_blobs(n_samples=m, centers=centers, cluster_std=std,
                  random_state=2, n_features=2)
# Split dataset into training, cross-validation, and test sets
X_train, X_tmp, y_train, y_tmp = train_test_split(X, y, test_size=0.50,
                                                  random_state=1)
X_cv, X_test, y_cv, y_test = train_test_split(X_tmp, y_tmp, test_size=0.20,
                                              random_state=1)
# Convert numpy arrays to PyTorch tensors
X_train = torch.FloatTensor(X_train)
y_train = torch.LongTensor(y_train)
X_cv = torch.FloatTensor(X_cv)
y_cv = torch.LongTensor(y_cv)
X_test = torch.FloatTensor(X_test)
y_test = torch.LongTensor(y_test)
# Define the simple neural network model
class SimpleModel(nn.Module):
    def __init__(self):
        super(SimpleModel, self).__init__()
        self.fc1 = nn.Linear(2, 6)
        self.fc2 = nn.Linear(6, 6)

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = self.fc2(x)
        return x
# Define the complex neural network model
class ComplexModel(nn.Module):
    def __init__(self):
        super(ComplexModel, self).__init__()
        self.fc1 = nn.Linear(2, 120)
        self.fc2 = nn.Linear(120, 40)
        self.fc3 = nn.Linear(40, 6)
    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = self.fc3(x)
        return x
# Function to train the model with specified regularization value
def train_model(model, X_train, y_train):
    # Set the model to training mode
    model.train()
    # Define loss function and optimizer
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=0.001)
    # Training loop
    epochs = 1000
    for epoch in range(epochs):
        optimizer.zero_grad()
        outputs = model(X_train)
        loss = criterion(outputs, y_train)
        loss.backward()
        optimizer.step()
# Train the simple model
model_simple = SimpleModel()
train_model(model_simple, X_train, y_train)
# Train the complex model
model_complex = ComplexModel()
train_model(model_complex, X_train, y_train)
# Evaluate model performance
def evaluate_model(model, X, y, dataset_name):
    # Set the model to evaluation mode
    model.eval()
    with torch.no_grad():
        outputs = model(X)
        _, predicted = torch.max(outputs, 1)
        accuracy = accuracy_score(y.numpy(), predicted.numpy())
        print(f"{dataset_name} Accuracy: {accuracy:.2%}")
        return accuracy
# Evaluate simple model on test set
test_accuracy_simple = evaluate_model(model_simple, X_test, y_test, "Simple Model (Test)")
# Evaluate complex model on test set
test_accuracy_complex = evaluate_model(model_complex, X_test, y_test, "Complex Model (Test)")
# Evaluate ideal model (KNN-based) on test data
```

- Error rates are also asme with the assignment document



```
PS C:\Users\ardah\UCM> python -u "c:\Users\ardah\UCM\p6_b\p6_b6.py"
Simple Model (Test) Accuracy: 82.50%
Complex Model (Test) Accuracy: 83.75%
Ideal Model (Test) Accuracy: 86.25%
PS C:\Users\ardah\UCM> 
```