Homework 6 Report Code
Name: Gilberto Feliú
Student ID: 801257813
Homework Number: 6
GitHub Repository: https://github.com/gfeliu-rgb/Intro-to-ML-ECGR-4105

Problem 1:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score
import torch
import torch.nn as nn
import torch.optim as optim
import time
file_path = "diabetes.csv"
df = pd.read_csv(file_path)

inputs = ['Pregnancies','Glucose','BloodPressure','SkinThickness',
          'Insulin','BMI','DiabetesPedigreeFunction','Age']
X = df[inputs].values
Y = df['Outcome'].values
m = len(Y)

X_train, X_test, Y_train, Y_test = train_test_split(
    X, Y, test_size=0.2, random_state=42
)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
X_train_tensor = torch.tensor(X_train, dtype=torch.float32)
Y_train_tensor = torch.tensor(Y_train.reshape(-1,1), dtype=torch.float32)
X_test_tensor = torch.tensor(X_test, dtype=torch.float32)
Y_test_tensor = torch.tensor(Y_test.reshape(-1,1), dtype=torch.float32)
```

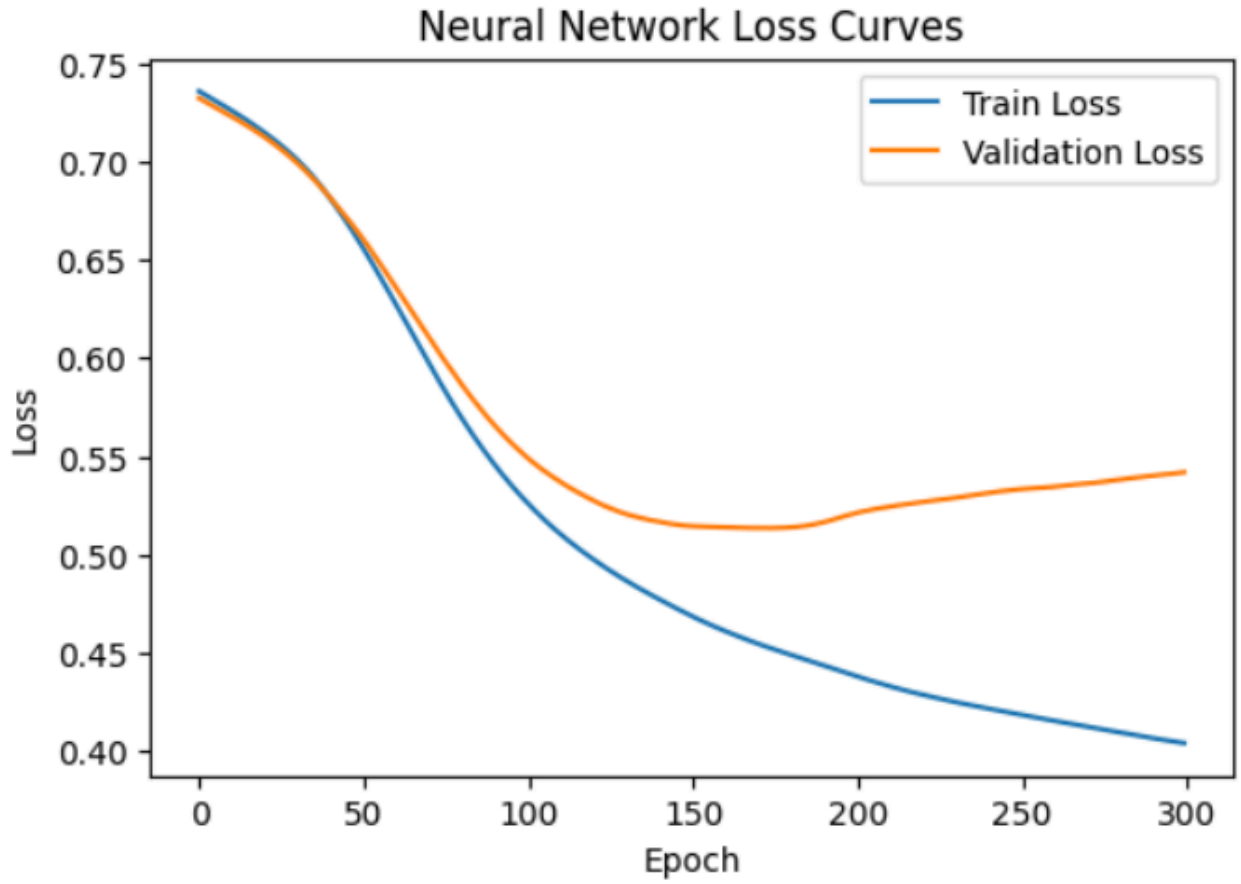
```
class NN(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(8, 16)
        self.fc2 = nn.Linear(16, 12)
        self.fc3 = nn.Linear(12, 4)
        self.fc4 = nn.Linear(4, 1)

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = torch.relu(self.fc3(x))
        x = torch.sigmoid(self.fc4(x))
        return x
model = NN()

loss_fn = nn.BCELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

num_epochs = 300
train_losses = []
val_losses = []

start = time.time()

for epoch in range(1, num_epochs+1):
    # Forward pass
    y_pred = model(X_train_tensor)
    loss = loss_fn(y_pred, Y_train_tensor)
```

```
for epoch in range(1, num_epochs+1):
    # Forward pass
    y_pred = model(X_train_tensor)
    loss = loss_fn(y_pred, Y_train_tensor)
    train_losses.append(loss.item())

    # Validation loss
    with torch.no_grad():
        y_val_pred = model(X_test_tensor)
        val_loss = loss_fn(y_val_pred, Y_test_tensor)
        val_losses.append(val_loss.item())

    # Backprop
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    if epoch % 50 == 0:
        print(f"Epoch {epoch}, Loss = {loss.item():.4f}")

end = time.time()

print("Training time =", end - start, "seconds")
print("Final Training Loss =", train_losses[-1])
plt.figure(figsize=(6,4))
plt.plot(train_losses, label="Train Loss")
plt.plot(val_losses, label="Validation Loss")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.title("Cancer NN Loss Curves")
plt.legend()
plt.show()

with torch.no_grad():
    y_pred_class = (model(X_test_tensor) >= 0.5).float().numpy()

# NN Evaluation
acc_nn  = accuracy_score(Y_test, y_pred_class)
prec_nn = precision_score(Y_test, y_pred_class)
```

```
rec_nn  = recall_score(Y_test, y_pred_class)
f1_nn   = f1_score(Y_test, y_pred_class)

print("\nNN Evaluation")
print("Accuracy :", acc_nn)
print("Precision:", prec_nn)
print("Recall   :", rec_nn)
print("F1 Score :", f1_nn)

# Logistic Regression
from sklearn.linear_model import LogisticRegression
log = LogisticRegression(max_iter=500)
log.fit(X_train, Y_train)
log_pred = log.predict(X_test)

print("\nLogistic Regression")
print("Accuracy :", accuracy_score(Y_test, log_pred))
print("Precision:", precision_score(Y_test, log_pred))
print("Recall   :", recall_score(Y_test, log_pred))
print("F1 Score :", f1_score(Y_test, log_pred))

# SVM
from sklearn.svm import SVC
svm = SVC(kernel='rbf')
svm.fit(X_train, Y_train)
svm_pred = svm.predict(X_test)

print("\nSVM")
print("Accuracy :", accuracy_score(Y_test, svm_pred))
print("Precision:", precision_score(Y_test, svm_pred))
print("Recall   :", recall_score(Y_test, svm_pred))
print("F1 Score :", f1_score(Y_test, svm_pred))
```

Results (1):
Epoch 50, Loss = 0.7449
Epoch 100, Loss = 0.6699
Epoch 150, Loss = 0.5099
Epoch 200, Loss = 0.4360
Epoch 250, Loss = 0.4159
Epoch 300, Loss = 0.4034
Training time = 0.8292570114135742 seconds

Final Training Loss = 0.4034341275691986
NN Evaluation
Accuracy : 0.7207792207792207
Precision: 0.603448275862069
Recall : 0.6363636363636364
F1 Score : 0.6194690265486725
Logistic Regression
Accuracy : 0.7532467532467533
Precision: 0.6491228070175439
Recall : 0.6727272727272727
F1 Score : 0.6607142857142857
SVM
Accuracy : 0.7337662337662337Precision: 0.6458333333333334
Recall : 0.5636363636363636
F1 Score : 0.6019417475728155



Problem 2:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score
from sklearn import datasets
import torch
import torch.nn as nn
import torch.optim as optim
import time

sample = datasets.load_breast_cancer()
X = sample.data
Y = sample.target
```

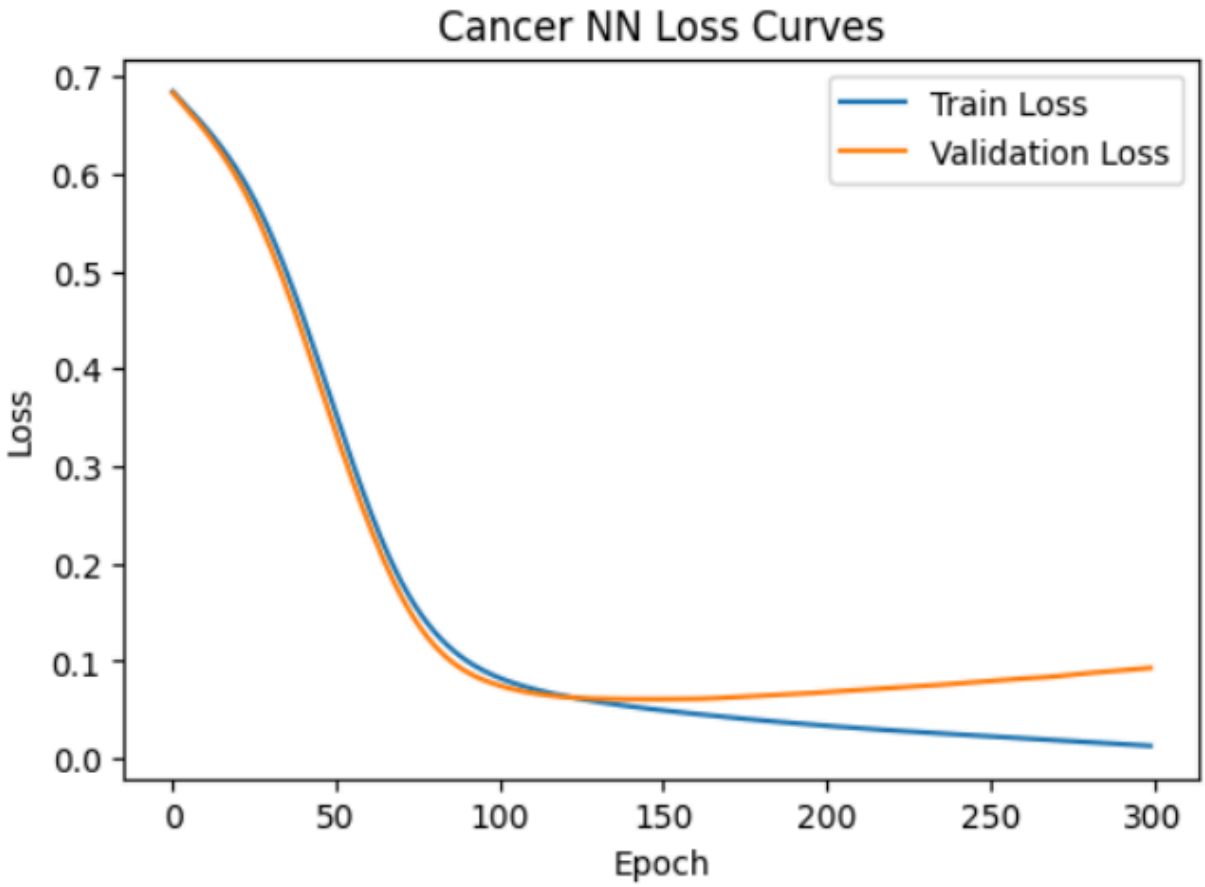
```
m = len(Y)
X_0 = np.ones((m, 1))
X = np.hstack([X_0, X])  # now shape = (569 × 31)
theta = np.zeros(X.shape[1])

X_train, X_test, Y_train, Y_test = train_test_split(
    X, Y, test_size=0.2, random_state=42
)

scaler = StandardScaler()
X_train[:, 1:] = scaler.fit_transform(X_train[:, 1:])
X_test[:, 1:] = scaler.transform(X_test[:, 1:])
X_train_tensor = torch.tensor(X_train, dtype=torch.float32)
Y_train_tensor = torch.tensor(Y_train.reshape(-1,1), dtype=torch.float32)
X_test_tensor  = torch.tensor(X_test, dtype=torch.float32)
Y_test_tensor  = torch.tensor(Y_test.reshape(-1,1), dtype=torch.float32)
class NN(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(X_train.shape[1], 32)
        self.fc2 = nn.Linear(32, 16)
        self.fc3 = nn.Linear(16, 8)
        self.fc4 = nn.Linear(8, 1)

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = torch.relu(self.fc3(x))
        x = torch.sigmoid(self.fc4(x))
        return x
```

```
model = NN()
loss_fn = nn.BCELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

num_epochs = 300
train_losses = []
val_losses = []

start = time.time()
for epoch in range(1, num_epochs+1):
```

```
    y_pred = model(X_train_tensor)
    loss = loss_fn(y_pred, Y_train_tensor)
    train_losses.append(loss.item())

    with torch.no_grad():
        y_val_pred = model(X_test_tensor)
        val_loss = loss_fn(y_val_pred, Y_test_tensor)
        val_losses.append(val_loss.item())

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    if epoch % 50 == 0:
        print(f"Epoch {epoch}, Loss = {loss.item():.4f}")
end = time.time()

print("Training Time =", end - start, "seconds")
print("Final Training Loss =", train_losses[-1])
plt.figure(figsize=(6,4))
plt.plot(train_losses, label="Train Loss")
plt.plot(val_losses, label="Validation Loss")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.title("Cancer NN Loss Curves")
plt.legend()
plt.show()
```

```
with torch.no_grad():
    y_pred_class = (model(X_test_tensor) >= 0.5).float().numpy()

# NN Evaluation
acc_nn  = accuracy_score(Y_test, y_pred_class)
prec_nn = precision_score(Y_test, y_pred_class)
rec_nn  = recall_score(Y_test, y_pred_class)
f1_nn   = f1_score(Y_test, y_pred_class)

print("\nNN Evaluation")
print("Accuracy :", acc_nn)
print("Precision:", prec_nn)
print("Recall   :", rec_nn)
print("F1 Score :", f1_nn)

# Logistic Regression
from sklearn.linear_model import LogisticRegression
log = LogisticRegression(max_iter=500)
log.fit(X_train, Y_train)
log_pred = log.predict(X_test)

print("\nLogistic Regression")
print("Accuracy :", accuracy_score(Y_test, log_pred))
print("Precision:", precision_score(Y_test, log_pred))
print("Recall   :", recall_score(Y_test, log_pred))
print("F1 Score :", f1_score(Y_test, log_pred))

# SVM
from sklearn.svm import SVC
svm = SVC(kernel='rbf')
svm.fit(X_train, Y_train)
svm_pred = svm.predict(X_test)

print("\nSVM")
print("Accuracy :", accuracy_score(Y_test, svm_pred))
print("Precision:", precision_score(Y_test, svm_pred))
print("Recall   :", recall_score(Y_test, svm_pred))
print("F1 Score :", f1_score(Y_test, svm_pred))
```

Results (2):
Epoch 50, Loss = 0.3624
Epoch 100, Loss = 0.0841
Epoch 150, Loss = 0.0499
Epoch 200, Loss = 0.0339
Epoch 250, Loss = 0.0232
Epoch 300, Loss = 0.0131
Training Time = 1.2786962985992432 seconds
Final Training Loss = 0.013138335198163986
NN Evaluation
Accuracy : 0.9824561403508771
Precision: 0.9859154929577465Recall : 0.9859154929577465
F1 Score : 0.9859154929577465
Logistic Regression
Accuracy : 0.9736842105263158
Precision: 0.9722222222222222
Recall : 0.9859154929577465
F1 Score : 0.9790209790209791
SVM
Accuracy : 0.9824561403508771
Precision: 0.9726027397260274
Recall : 1.0
F1 Score : 0.9861111111111112


Cancer NN Loss Curves
Train Loss
Validation Loss
Loss
Epoch
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0
50
100
150
200
250
300

Problem 3:

```
data_path = "../data-unversioned/p1ch7"
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4823, 0.4468),
                         (0.2470, 0.2435, 0.2616))
])

cifar10_train = datasets.CIFAR10(data_path, train=True, download=True,
transform=transform)
cifar10_test  = datasets.CIFAR10(data_path, train=False, download=True,
transform=transform)

cifar10_trainX = torch.stack([img.view(-1) for img, _ in cifar10_train])
cifar10_trainY = torch.tensor([label for _, label in cifar10_train],
dtype=torch.long)

cifar10_testX = torch.stack([img.view(-1) for img, _ in cifar10_test])
cifar10_testY = torch.tensor([label for _, label in cifar10_test],
dtype=torch.long)

cifar10_trainX = cifar10_trainX.to(device)
cifar10_trainY = cifar10_trainY.to(device)
cifar10_testX  = cifar10_testX.to(device)
cifar10_testY  = cifar10_testY.to(device)
```

```
train_dataset = torch.utils.data.TensorDataset(cifar10_trainX,
cifar10_trainY)
val_dataset   = torch.utils.data.TensorDataset(cifar10_testX,
cifar10_testY)

train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=64,
shuffle=True)
val_loader   = torch.utils.data.DataLoader(val_dataset, batch_size=1000,
shuffle=False)

input_size = 3 * 32 * 32
model = nn.Sequential(
    nn.Linear(input_size, 512),
    nn.Tanh(),
```

```
    nn.Linear(512, 10),
    nn.LogSoftmax(dim=1)
).to(device)

learning_rate = 1e-3
optimizer = optim.SGD(model.parameters(), lr=learning_rate)
loss_fn = nn.NLLLoss()

n_epochs = 300
start = time.time()

for epoch in range(n_epochs):
    model.train()
    running_loss = 0.0
    for batch_X, batch_Y in train_loader:
        outputs = model(batch_X)
        loss = loss_fn(outputs, batch_Y)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        running_loss += loss.item() * batch_X.size(0)
```

```
    epoch_loss = running_loss / len(train_loader.dataset)
    print(f"Epoch {epoch+1}, Loss: {epoch_loss:.4f}")

end = time.time()
print("\nTraining Time =", end - start, "seconds")
print("Final Training Loss =", epoch_loss)

model.eval()
correct = 0
total = 0
with torch.no_grad():
    for batch_X, batch_Y in val_loader:
        outputs = model(batch_X)
        preds = outputs.argmax(dim=1)
        correct += (preds == batch_Y).sum().item()
        total += batch_Y.size(0)
```

Results (3a):
Training Time = 626.65065574646 seconds
Final Training Loss = 0.49717244245529174
Evaluation Accuracy = 0.4871

3b)

```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms
import time

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

data_path = "../data-unversioned/p1ch7"
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4823, 0.4468),
                         (0.2470, 0.2435, 0.2616))
])

cifar10_train = datasets.CIFAR10(data_path, train=True, download=True,
transform=transform)
cifar10_test  = datasets.CIFAR10(data_path, train=False, download=True,
transform=transform)

cifar10_trainX = torch.stack([img.view(-1) for img, _ in cifar10_train])
cifar10_trainY = torch.tensor([label for _, label in cifar10_train],
dtype=torch.long)

cifar10_testX = torch.stack([img.view(-1) for img, _ in cifar10_test])
cifar10_testY = torch.tensor([label for _, label in cifar10_test],
dtype=torch.long)
```

```
cifar10_trainX = cifar10_trainX.to(device)
cifar10_trainY = cifar10_trainY.to(device)
cifar10_testX  = cifar10_testX.to(device)
cifar10_testY  = cifar10_testY.to(device)

train_dataset = torch.utils.data.TensorDataset(cifar10_trainX,
cifar10_trainY)
val_dataset   = torch.utils.data.TensorDataset(cifar10_testX,
cifar10_testY)

train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=64,
shuffle=True)
val_loader   = torch.utils.data.DataLoader(val_dataset, batch_size=1000,
shuffle=False)

input_size = 3 * 32 * 32
model = nn.Sequential(
    nn.Linear(input_size, 1024),
    nn.Tanh(),
    nn.Linear(1024, 512),
    nn.Tanh(),
    nn.Linear(512, 128),
    nn.Tanh(),
    nn.Linear(128, 10),
    nn.LogSoftmax(dim=1)
).to(device)
```

```
learning_rate = 1e-3
optimizer = optim.SGD(model.parameters(), lr=learning_rate)
loss_fn = nn.NLLLoss()

n_epochs = 300
start = time.time()

for epoch in range(n_epochs):
    model.train()
    running_loss = 0.0
    for batch_X, batch_Y in train_loader:
        outputs = model(batch_X)
```

```
        loss = loss_fn(outputs, batch_Y)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        running_loss += loss.item() * batch_X.size(0)

    epoch_loss = running_loss / len(train_loader.dataset)
    print(f"Epoch {epoch+1}, Loss: {epoch_loss:.4f}")

end = time.time()
print("\nTraining Time =", end - start, "seconds")
print("Final Training Loss =", epoch_loss)

model.eval()
correct = 0
total = 0
with torch.no_grad():
    for batch_X, batch_Y in val_loader:
        outputs = model(batch_X)
        preds = outputs.argmax(dim=1)
        correct += (preds == batch_Y).sum().item()
        total += batch_Y.size(0)

accuracy = correct / total
print("Evaluation Accuracy =", accuracy)
```

Results (3b):
Training Time = 1167.7478566169739 seconds
Final Training Loss = 0.015094490163475275
Evaluation Accuracy = 0.4552