heart-failure-identifier

April 16, 2023

0.1 # Heart-Failure-Identifier

CSCI 4050U, Machine Learning Professor Ken Pu. Course Final Project Faculty of Science, Ontario Tech University April 16, 2023

Our Team. Rija Baig (100746674) Preet Panchal (100707094) Eihab Syed (100707448)

Link to the Dataset. https://www.kaggle.com/datasets/fedesoriano/heart-failure-prediction

About the Dataset. The dataset is a combined heart disease dataset consisting of 11 features and 918 observations. The aim of the dataset is to predict the presence of heart disease in patients using these features. The features in the dataset include age, sex, chest pain type, resting blood pressure, serum cholesterol, fasting blood sugar, resting electrocardiogram results, maximum heart rate achieved, exercise-induced angina, oldpeak, and the slope of the peak exercise ST segment. The dataset was created by combining five different heart disease datasets, making it the largest heart disease dataset available for research purposes.

Problem. For our project, we are studying a binary classification problem, where we try to predict whether a patient has heart disease or not based on the 11 input features. The problem can be framed as training a machine learning model to accurately classify patients into two categories: those with heart disease (output class = 1) and those without heart disease (output class = 0).

Objective. The aim of the project would be to develop a machine learning model that can generalize well on unseen data and achieve a high accuracy rate on the test set.

[1]:
```
# Import libraries
import time
import torch
from torch import nn
from torch import optim
from torch import tensor
from torch.utils.data import Dataset, DataLoader, random_split, TensorDataset
import torchvision
from torchsummaryX import summary
import numpy as np
import pandas as pd
import seaborn as sns
import os
import matplotlib.pyplot as plt
from importlib import reload
```

```
import sklearn
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from scipy.stats import zscore
from sklearn.metrics import confusion_matrix,classification_report,f1_score

import warnings
warnings.filterwarnings('ignore')
```

0.2 Loading the Data

[2]:
```
# Loading the dataset
df = pd.read_csv('data/heart.csv')
df.head()
```

[2]:
```
   Age Sex ChestPainType  RestingBP  Cholesterol  FastingBS RestingECG  MaxHR  \
0   40   M           ATA        140          289          0     Normal    172
1   49   F           NAP        160          180          0     Normal    156
2   37   M           ATA        130          283          0         ST     98
3   48   F           ASY        138          214          0     Normal    108
4   54   M           NAP        150          195          0     Normal    122

  ExerciseAngina  Oldpeak ST_Slope  HeartDisease
0              N      0.0       Up             0
1              N      1.0     Flat             1
2              N      0.0       Up             0
3              Y      1.5     Flat             1
4              N      0.0       Up             0
```

[3]:
```
df.shape
```

[3]:
```
(918, 12)
```

[4]:
```
# Loading the target values (ie. the output class)
target = df.iloc[:, 11]
target.head()
```

[4]:
```
0    0
1    1
2    0
3    1
4    0
Name: HeartDisease, dtype: int64
```

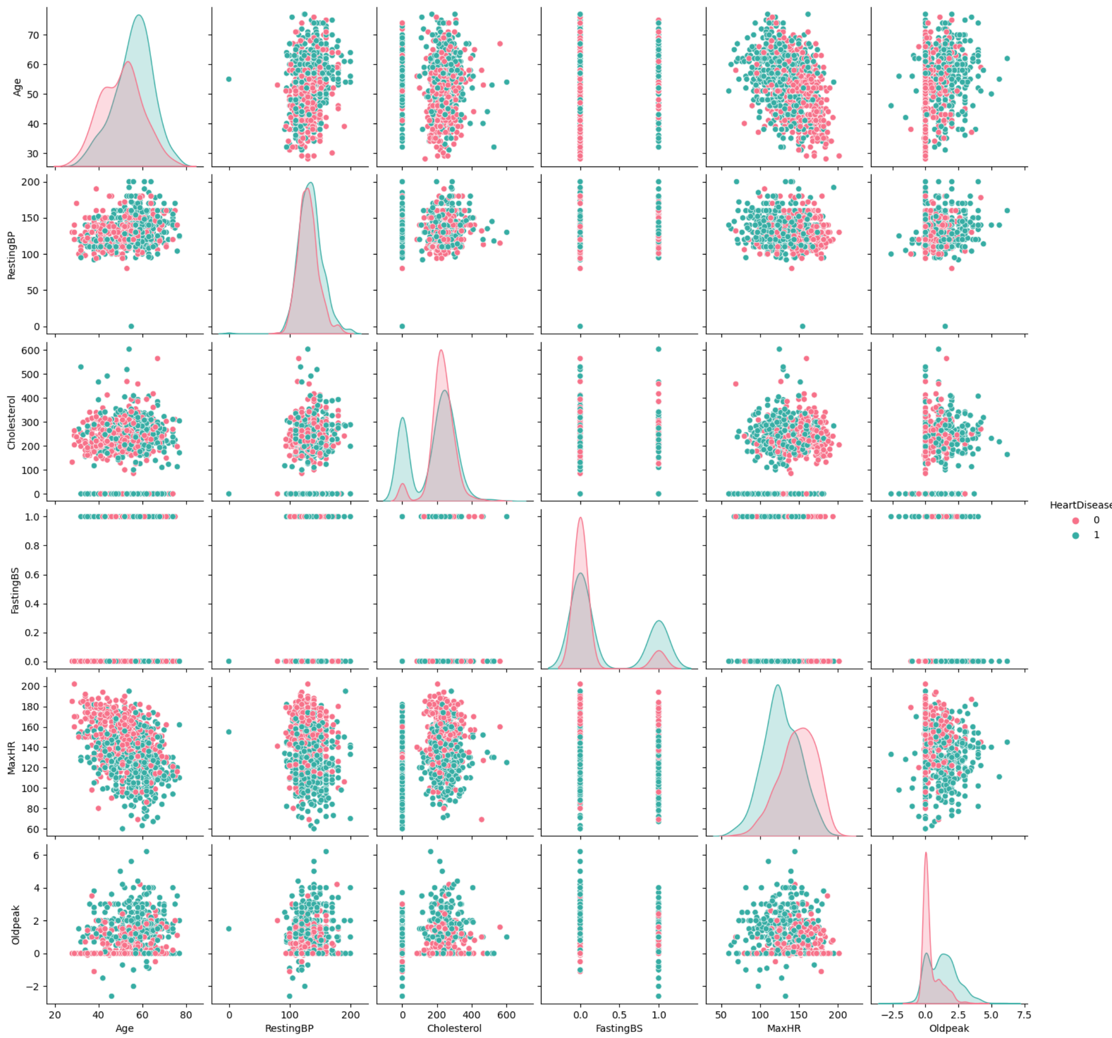
[5]:
```
# Loading the dataset with the 11 features
features = df.iloc[:, :-1]
features.head()
```

[5]:
```
   Age Sex ChestPainType  RestingBP  Cholesterol  FastingBS RestingECG  MaxHR  \
0   40   M           ATA        140          289          0     Normal    172
1   49   F           NAP        160          180          0     Normal    156
2   37   M           ATA        130          283          0         ST     98
3   48   F           ASY        138          214          0     Normal    108
4   54   M           NAP        150          195          0     Normal    122

  ExerciseAngina  Oldpeak ST_Slope
0              N      0.0       Up
1              N      1.0     Flat
2              N      0.0       Up
3              Y      1.5     Flat
4              N      0.0       Up
```

[6]:
```python
# Getting information of our 11 attributes
features.info()
```

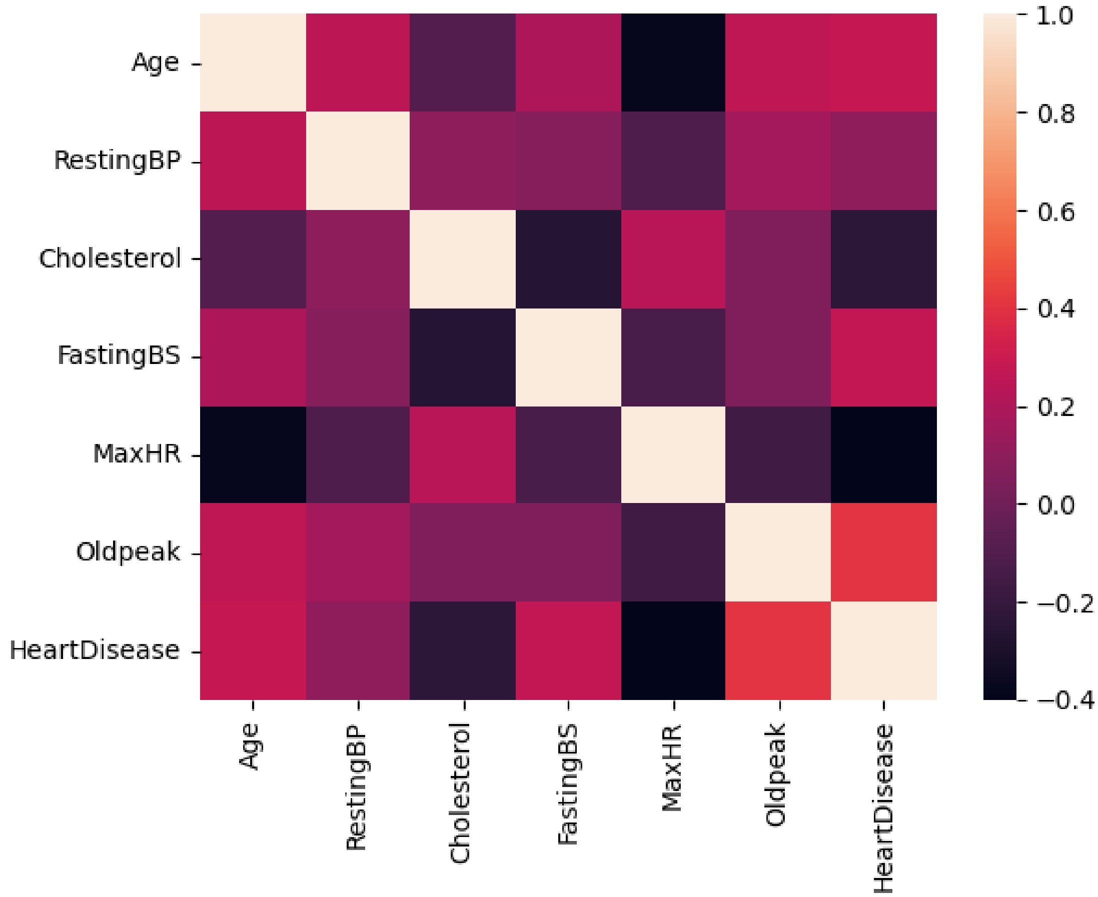
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 918 entries, 0 to 917
Data columns (total 11 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Age             918 non-null    int64
 1   Sex             918 non-null    object
 2   ChestPainType   918 non-null    object
 3   RestingBP       918 non-null    int64
 4   Cholesterol     918 non-null    int64
 5   FastingBS       918 non-null    int64
 6   RestingECG      918 non-null    object
 7   MaxHR           918 non-null    int64
 8   ExerciseAngina  918 non-null    object
 9   Oldpeak         918 non-null    float64
 10  ST_Slope        918 non-null    object
dtypes: float64(1), int64(5), object(5)
memory usage: 79.0+ KB
```

[7]:
```python
# Visualizing dataset using seaborn pairplot
sns.color_palette("flare", as_cmap=True)
sns.pairplot(df, hue='HeartDisease', palette='husl')
plt.show()
```



0.3 Preprocessing the Data

[9]:
```
# Visualizing all features with numerical values using seaborn heatmap
sns.heatmap(df.corr())
plt.show()
```



[10]:
```python
# Creating a copy of our dataframe to manipulate non-numerical values
encoded_df = df.copy()

# Mapping values of the Sex and ExerciseAngina column to numbers
encoded_df.Sex = encoded_df.Sex.map({"M":1,"F":0})
encoded_df.ExerciseAngina = encoded_df.ExerciseAngina.map({"Y":1,"N":0})
encoded_df.head()
```

[10]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	\
0	40	1	ATA	140	289	0	Normal	
1	49	0	NAP	160	180	0	Normal	
2	37	1	ATA	130	283	0	ST	
3	48	0	ASY	138	214	0	Normal	
4	54	1	NAP	150	195	0	Normal	

	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease
0	172	0	0.0	Up	0
1	156	0	1.0	Flat	1

```
2    98              0     0.0     Up          0
3   108              1     1.5   Flat          1
4   122              0     0.0     Up          0
```

[11]:
```
# Creating a copy of the encoded dataframe and dropping the output class
feature_df = encoded_df.drop(['HeartDisease'],axis=1)

# Performing one-hot encoding on dataframe to columns with non-numerical values␣
↪into categorical columns
feature_df = pd.get_dummies(feature_df, drop_first=True)
feature_df.head()
```

[11]:
```
   Age  Sex  RestingBP  Cholesterol  FastingBS  MaxHR  ExerciseAngina  \
0   40    1        140          289          0    172               0
1   49    0        160          180          0    156               0
2   37    1        130          283          0     98               0
3   48    0        138          214          0    108               1
4   54    1        150          195          0    122               0

   Oldpeak  ChestPainType_ATA  ChestPainType_NAP  ChestPainType_TA  \
0      0.0                  1                  0                 0
1      1.0                  0                  1                 0
2      0.0                  1                  0                 0
3      1.5                  0                  0                 0
4      0.0                  0                  1                 0

   RestingECG_Normal  RestingECG_ST  ST_Slope_Flat  ST_Slope_Up
0                  1              0              0            1
1                  1              0              1            0
2                  0              1              0            1
3                  1              0              1            0
4                  1              0              0            1
```

[12]:
```
# Defining our x (ie. features) and y (ie. labels=output class)
x = feature_df.values
y = encoded_df.HeartDisease.values
```

[13]:
```
# Using StandardScaler() to transform the training and testing datasets
# This allows for the distribution to have a mean value 0 and standard␣
↪deviation of 1
scaler = StandardScaler()

# Split data into a train set and a test set
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

[14]:
```python
# Initializing batch size
batch_size = 256

# Creating tensordataset for the train set
x_train = torch.tensor(x_train).float()
y_train = torch.tensor(y_train).float().reshape(-1,1)
train_dataset = TensorDataset(x_train, y_train)

# Creating tensordataset for the test set
x_test = torch.tensor(x_test).float()
y_test = torch.tensor(y_test).float().reshape(-1,1)
test_dataset = TensorDataset(x_test, y_test)

# Defining train and test dataloaders
train_dataloader = DataLoader(train_dataset, batch_size=batch_size,␣
 ↪shuffle=True, drop_last=True)
test_dataloader = DataLoader(test_dataset, batch_size=test_dataset.tensors[0].
 ↪shape[0])

print(f"Number of Training Samples = {len(train_dataloader.dataset)}\nNumber of␣
 ↪Testing Samples = {len(test_dataloader.dataset)}")
```

```
Number of Training Samples = 734
Number of Testing Samples = 184
```

[15]:
```python
train_dataset.tensors[0].size()
```

[15]:
```
torch.Size([734, 15])
```

0.4 The Model

[16]:
```python
# Setting up the neural network
class HFModel(nn.Module):
    def __init__(self):
        super().__init__()

        # Input layer
        self.input = nn.Linear(15,100)
        self.relu1 = nn.ReLU()
        self.dropout = nn.Dropout(0.2)

        # Hidden layer
        self.fc1 = nn.Linear(100,100)
        self.bnnorm1 = nn.BatchNorm1d(100)
        self.relu2 = nn.ReLU()

        # Output layer
```

```
        self.output = nn.Linear(100,1)
        self.dr = 0.2

    def forward(self,x):
        x = self.relu1(self.input(x))
        x = self.dropout(x)
        x = self.bnnorm1(x)
        x = self.relu2(self.fc1(x))
        x = self.dropout(x)

        return self.output(x)
```

0.5 Training the Model

[17]:
```
# Defining our train function
def train(model: HFModel,
          train_dataset: Dataset,
          test_dataset: Dataset,
          epoch: int,
          lr: float,
          max_batches=None):

    # Initializing train and test accuracy to store values
    train_acc = torch.zeros(epoch)
    test_acc = torch.zeros(epoch)

    # Redefining train and test dataloaders
    train_dataloader = DataLoader(train_dataset, batch_size=max_batches,␣
 ↪shuffle=True, drop_last=True)
    test_dataloader = DataLoader(test_dataset, batch_size=test_dataset.
 ↪tensors[0].shape[0])

    # Calling loss function and optimizer
    lossFunc = nn.BCEWithLogitsLoss()
    optimizer = torch.optim.Adam(model.parameters(), lr=lr)

    # Training the model
    for i in range(epoch):
        start = time.time()
        for xs, targets in train_dataloader:
            # Finding train accuracy
            acc_list = []
            model.train()

            # Forward pass
            y_out = model(xs)
```

```
            loss = lossFunc(y_out, targets) # her loss
            y_out = (y_out>0).float()
            acc_list.append(100*torch.mean((y_out==targets).float()).item())

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

        # Finding test accuracy
        train_acc[i] = np.mean(acc_list)
        model.eval()
        xs, targets = next(iter(test_dataloader))
        y_out = model(xs)
        loss = lossFunc(y_out, targets)
        y_out = (y_out>0).float()
        test_acc[i] = 100*torch.mean((targets==y_out).float()).item()

        duration = time.time() - start

        # Displaying epoch information
        print("[Epoch {} ({:.2f}s)]: "
              "Loss={:.2f}%, "
              "Train Accuracy={:.2f}%, "
              "Test Accuracy={:.2f}%"
              .format(i,
                      duration,
                      loss,
                      train_acc[i],
                      test_acc[i]
                     )
             )

    return model, test_acc, train_acc
```

[22]:
```
# Running train model
model = HFModel()
model, test_acc, train_acc = train(model, train_dataset, test_dataset,␣
 ↪epoch=50, lr=0.001, max_batches=256)
```

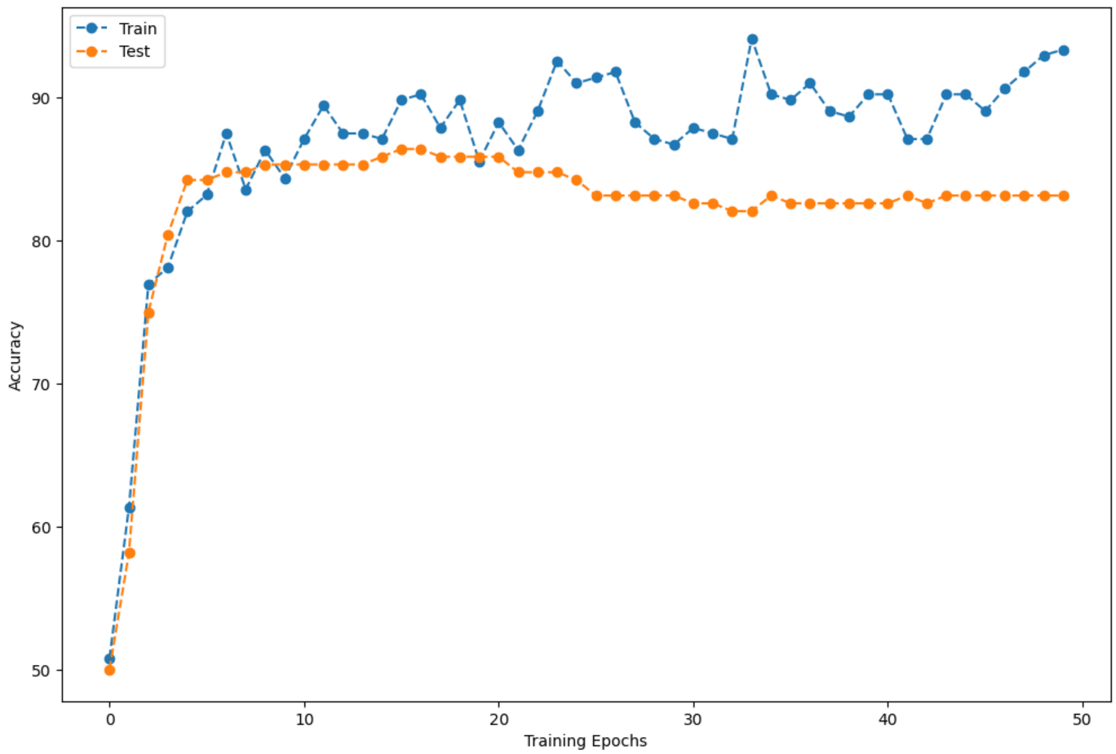
```
[Epoch 0 (0.02s)]: Loss=0.67%, Train Accuracy=50.78%, Test Accuracy=50.00%
[Epoch 1 (0.01s)]: Loss=0.65%, Train Accuracy=61.33%, Test Accuracy=58.15%
[Epoch 2 (0.01s)]: Loss=0.62%, Train Accuracy=76.95%, Test Accuracy=75.00%
[Epoch 3 (0.01s)]: Loss=0.59%, Train Accuracy=78.12%, Test Accuracy=80.43%
[Epoch 4 (0.01s)]: Loss=0.55%, Train Accuracy=82.03%, Test Accuracy=84.24%
[Epoch 5 (0.01s)]: Loss=0.52%, Train Accuracy=83.20%, Test Accuracy=84.24%
[Epoch 6 (0.01s)]: Loss=0.48%, Train Accuracy=87.50%, Test Accuracy=84.78%
```

```
[Epoch 7 (0.01s)]: Loss=0.45%, Train Accuracy=83.59%, Test Accuracy=84.78%
[Epoch 8 (0.01s)]: Loss=0.43%, Train Accuracy=86.33%, Test Accuracy=85.33%
[Epoch 9 (0.01s)]: Loss=0.40%, Train Accuracy=84.38%, Test Accuracy=85.33%
[Epoch 10 (0.01s)]: Loss=0.39%, Train Accuracy=87.11%, Test Accuracy=85.33%
[Epoch 11 (0.01s)]: Loss=0.37%, Train Accuracy=89.45%, Test Accuracy=85.33%
[Epoch 12 (0.01s)]: Loss=0.37%, Train Accuracy=87.50%, Test Accuracy=85.33%
[Epoch 13 (0.01s)]: Loss=0.36%, Train Accuracy=87.50%, Test Accuracy=85.33%
[Epoch 14 (0.01s)]: Loss=0.36%, Train Accuracy=87.11%, Test Accuracy=85.87%
[Epoch 15 (0.01s)]: Loss=0.36%, Train Accuracy=89.84%, Test Accuracy=86.41%
[Epoch 16 (0.01s)]: Loss=0.36%, Train Accuracy=90.23%, Test Accuracy=86.41%
[Epoch 17 (0.01s)]: Loss=0.37%, Train Accuracy=87.89%, Test Accuracy=85.87%
[Epoch 18 (0.01s)]: Loss=0.37%, Train Accuracy=89.84%, Test Accuracy=85.87%
[Epoch 19 (0.01s)]: Loss=0.37%, Train Accuracy=85.55%, Test Accuracy=85.87%
[Epoch 20 (0.01s)]: Loss=0.37%, Train Accuracy=88.28%, Test Accuracy=85.87%
[Epoch 21 (0.01s)]: Loss=0.38%, Train Accuracy=86.33%, Test Accuracy=84.78%
[Epoch 22 (0.01s)]: Loss=0.38%, Train Accuracy=89.06%, Test Accuracy=84.78%
[Epoch 23 (0.01s)]: Loss=0.39%, Train Accuracy=92.58%, Test Accuracy=84.78%
[Epoch 24 (0.01s)]: Loss=0.39%, Train Accuracy=91.02%, Test Accuracy=84.24%
[Epoch 25 (0.01s)]: Loss=0.39%, Train Accuracy=91.41%, Test Accuracy=83.15%
[Epoch 26 (0.01s)]: Loss=0.39%, Train Accuracy=91.80%, Test Accuracy=83.15%
[Epoch 27 (0.01s)]: Loss=0.39%, Train Accuracy=88.28%, Test Accuracy=83.15%
[Epoch 28 (0.01s)]: Loss=0.40%, Train Accuracy=87.11%, Test Accuracy=83.15%
[Epoch 29 (0.01s)]: Loss=0.40%, Train Accuracy=86.72%, Test Accuracy=83.15%
[Epoch 30 (0.01s)]: Loss=0.40%, Train Accuracy=87.89%, Test Accuracy=82.61%
[Epoch 31 (0.01s)]: Loss=0.40%, Train Accuracy=87.50%, Test Accuracy=82.61%
[Epoch 32 (0.01s)]: Loss=0.40%, Train Accuracy=87.11%, Test Accuracy=82.07%
[Epoch 33 (0.01s)]: Loss=0.40%, Train Accuracy=94.14%, Test Accuracy=82.07%
[Epoch 34 (0.01s)]: Loss=0.40%, Train Accuracy=90.23%, Test Accuracy=83.15%
[Epoch 35 (0.01s)]: Loss=0.40%, Train Accuracy=89.84%, Test Accuracy=82.61%
[Epoch 36 (0.01s)]: Loss=0.40%, Train Accuracy=91.02%, Test Accuracy=82.61%
[Epoch 37 (0.01s)]: Loss=0.40%, Train Accuracy=89.06%, Test Accuracy=82.61%
[Epoch 38 (0.01s)]: Loss=0.40%, Train Accuracy=88.67%, Test Accuracy=82.61%
[Epoch 39 (0.01s)]: Loss=0.40%, Train Accuracy=90.23%, Test Accuracy=82.61%
[Epoch 40 (0.01s)]: Loss=0.40%, Train Accuracy=90.23%, Test Accuracy=82.61%
[Epoch 41 (0.01s)]: Loss=0.40%, Train Accuracy=87.11%, Test Accuracy=83.15%
[Epoch 42 (0.01s)]: Loss=0.40%, Train Accuracy=87.11%, Test Accuracy=82.61%
[Epoch 43 (0.01s)]: Loss=0.40%, Train Accuracy=90.23%, Test Accuracy=83.15%
[Epoch 44 (0.01s)]: Loss=0.40%, Train Accuracy=90.23%, Test Accuracy=83.15%
[Epoch 45 (0.01s)]: Loss=0.40%, Train Accuracy=89.06%, Test Accuracy=83.15%
[Epoch 46 (0.01s)]: Loss=0.40%, Train Accuracy=90.62%, Test Accuracy=83.15%
[Epoch 47 (0.01s)]: Loss=0.41%, Train Accuracy=91.80%, Test Accuracy=83.15%
[Epoch 48 (0.01s)]: Loss=0.41%, Train Accuracy=92.97%, Test Accuracy=83.15%
[Epoch 49 (0.01s)]: Loss=0.41%, Train Accuracy=93.36%, Test Accuracy=83.15%
```

0.6 The Results

[23]:
```
# Visualizing the train and test accuracy
model_fig = plt.figure(figsize=(12,8))
plt.plot(train_acc, 'o--', label='Train')
plt.plot(test_acc, 'o--', label='Test')
plt.xlabel('Training Epochs')
plt.ylabel('Accuracy')
plt.legend()
model_fig
```

[23]:



[24]:
```
# Predictions for testing set
x, y = next(iter(test_dataloader))
y_out = model(x)
test_pred = (y_out>0).float()

# Predictions for training set
train_pred = (model(train_dataloader.dataset.tensors[0])>1).float()

# Initializing metrics for confusion matrix
train_conf = confusion_matrix(train_dataloader.dataset.tensors[1], train_pred)
test_conf = confusion_matrix(y, test_pred)
```

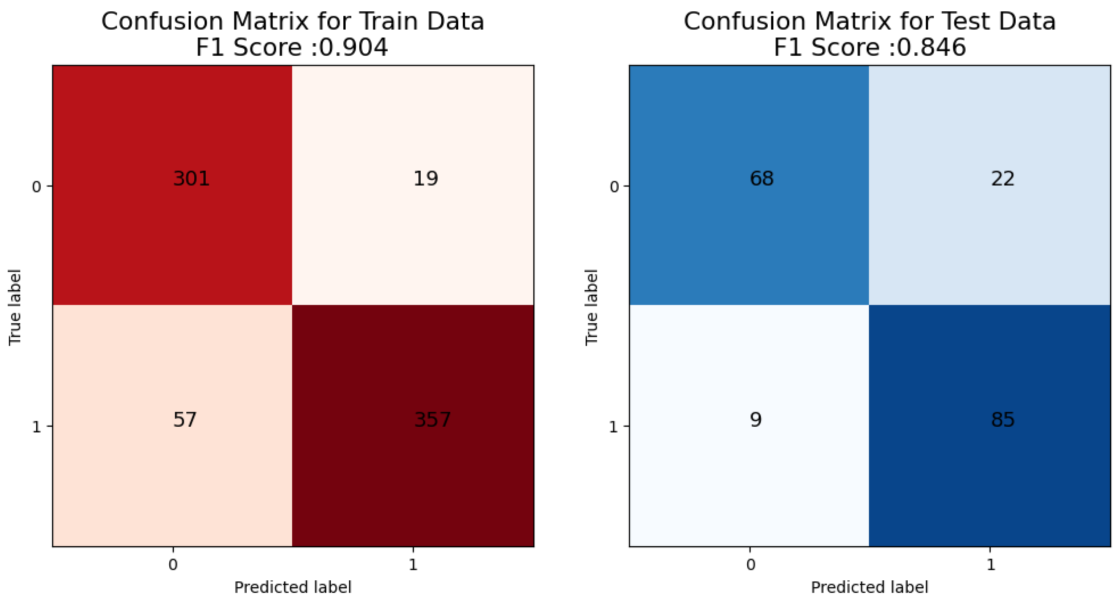
[25]:

```
# Calculating f1 score
f1_train = f1_score(train_dataloader.dataset.tensors[1], train_pred)
f1_test = f1_score(y, test_pred)
```

[26]:

```
# Plotting the confusion matrix for train and test datasets
fig, ax = plt.subplots(1, 2, figsize=(12, 8))

plt.rcParams.update({'font.size':13})

ax[0].set_title(f"Confusion Matrix for Train Data\nF1 Score :{f1_train:.3f}")
ax[0].imshow(train_conf, 'Reds', vmax=len(train_pred) / 2)
ax[0].set_xticks([0,1])
ax[0].set_yticks([0,1])
ax[0].set_xlabel("Predicted label")
ax[0].set_ylabel("True label")

ax[0].text(0, 0, train_conf[0, 0])
ax[0].text(0, 1, train_conf[1, 0])
ax[0].text(1, 0, train_conf[0, 1])
ax[0].text(1, 1, train_conf[1, 1])


ax[1].set_title(f"Confusion Matrix for Test Data\nF1 Score :{f1_test:.3f}")
ax[1].imshow(test_conf, 'Blues', vmax=len(test_pred) / 2)
ax[1].set_xticks([0, 1])
ax[1].set_yticks([0, 1])
ax[1].set_xlabel("Predicted label")
ax[1].set_ylabel("True label")

ax[1].text(0, 0, test_conf[0, 0])
ax[1].text(0, 1, test_conf[1, 0])
ax[1].text(1, 0, test_conf[0, 1])
ax[1].text(1, 1, test_conf[1, 1])
fig
```

[26]:



0.7 Final Remarks

For the confusion matrix with the training data, we can see that our model was able to accurately predict if a subject had heart disease. The true value being on the y axis and the predicted value being on the x axis, our model predicted 71 false negatives means our model has a 23% chance of producing false negatives. It produced 21 false positives giving us a 6% chance at false positives, which are very low values. For the testing dataset, our train model provided a pretty high accuracy where we only got 8 false negative cases giving our model an 11% chance of false negatives and 7 false positives giving us a 6% chance for false positives. **Note:** The numbers provided in the above statement may vary due to re-runs of the training model, however, the expectation is that the % values will be very close in correlation.

0.8 Dataset Citation & Acknowledgement

fedesoriano. (September 2021). Heart Failure Prediction Dataset. Retrieved [Apr 16, 2023] from https://www.kaggle.com/fedesoriano/heart-failure-prediction.