diabetes-nn

April 14, 2025

Zavier Andrianarivo 4/11/25 Professor Wallisch Homework #4 Artificial Neural Networks on Predicting Diabetes

1 Overview

Last homework, we used different types of models such as logistic regression, support vector machines, and different tree based models to predict diabetes. Now, given the same dataset, we'll be using different neural networks architectures (single-layer perceptron, feedforward, a "deep" neural network, CNN, and a network of our choice) to predict diabetes.

1.1 Question #1

Build and train a Perceptron (one input layer, one output layer, no hidden layers and no activation functions) to classify diabetes from the rest of the dataset. What is the AUC of this model?

[1]:
```
pip install torch torchvision
```

```
Requirement already satisfied: torch in /opt/anaconda3/lib/python3.11/site-
packages (2.6.0)
Requirement already satisfied: torchvision in
/opt/anaconda3/lib/python3.11/site-packages (0.21.0)
Requirement already satisfied: filelock in /opt/anaconda3/lib/python3.11/site-
packages (from torch) (3.13.1)
Requirement already satisfied: typing-extensions>=4.10.0 in
/opt/anaconda3/lib/python3.11/site-packages (from torch) (4.12.2)
Requirement already satisfied: networkx in /opt/anaconda3/lib/python3.11/site-
packages (from torch) (3.1)
Requirement already satisfied: jinja2 in /opt/anaconda3/lib/python3.11/site-
packages (from torch) (3.1.3)
Requirement already satisfied: fsspec in /opt/anaconda3/lib/python3.11/site-
packages (from torch) (2023.10.0)
Requirement already satisfied: sympy==1.13.1 in
/opt/anaconda3/lib/python3.11/site-packages (from torch) (1.13.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in
/opt/anaconda3/lib/python3.11/site-packages (from sympy==1.13.1->torch) (1.3.0)
Requirement already satisfied: numpy in /opt/anaconda3/lib/python3.11/site-
packages (from torchvision) (1.24.4)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in
/opt/anaconda3/lib/python3.11/site-packages (from torchvision) (10.2.0)
```

```
Requirement already satisfied: MarkupSafe>=2.0 in
/opt/anaconda3/lib/python3.11/site-packages (from jinja2->torch) (2.1.3)
Note: you may need to restart the kernel to use updated packages.
```

[2]:
```python
import numpy as np
import pandas as pd
import torch
from sklearn.linear_model import Perceptron
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
```

[3]:
```python
# load dataset
data = pd.read_csv('./diabetes.csv')
df = pd.DataFrame(data)
data.head(n=10)
```

[3]:
```
   Diabetes  HighBP  HighChol  BMI  Smoker  Stroke  Myocardial  PhysActivity  \
0         0       1         1   40       1       0           0             0
1         0       0         0   25       1       0           0             1
2         0       1         1   28       0       0           0             0
3         0       1         0   27       0       0           0             1
4         0       1         1   24       0       0           0             1
5         0       1         1   25       1       0           0             1
6         0       1         0   30       1       0           0             0
7         0       1         1   25       1       0           0             1
8         1       1         1   30       1       0           1             0
9         0       0         0   24       0       0           0             0

   Fruit  Vegetables  ...  NotAbleToAffordDoctor  GeneralHealth  MentalHealth  \
0      0           1  ...                      0              5            18
1      0           0  ...                      1              3             0
2      1           0  ...                      1              5            30
3      1           1  ...                      0              2             0
4      1           1  ...                      0              2             3
5      1           1  ...                      0              2             0
6      0           0  ...                      0              3             0
7      0           1  ...                      0              3             0
8      1           1  ...                      0              5            30
9      0           1  ...                      0              2             0

   PhysicalHealth  HardToClimbStairs  BiologicalSex  AgeBracket  \
0              15                  1              1           9
1               0                  0              1           7
2              30                  1              1           9
3               0                  0              1          11
4               0                  0              1          11
```

```
5                 2                     0                 2            10
6                14                     0                 1             9
7                 0                     1                 1            11
8                30                     1                 1             9
9                 0                     0                 2             8

   EducationBracket  IncomeBracket  Zodiac
0                 4              3      10
1                 6              1      11
2                 4              8       2
3                 3              6      11
4                 5              4       8
5                 6              8       2
6                 6              7       4
7                 4              4       7
8                 5              1      12
9                 4              3      12

[10 rows x 22 columns]
```

[4]:

```
# set X and y
X = df.drop(columns=['Diabetes'])
y = df['Diabetes']

print(f'X: \n{X}')
print(f'y: \n{y}')
```

```
X:
        HighBP  HighChol  BMI  Smoker  Stroke  Myocardial  PhysActivity  \
0            1         1   40       1       0           0             0
1            0         0   25       1       0           0             1
2            1         1   28       0       0           0             0
3            1         0   27       0       0           0             1
4            1         1   24       0       0           0             1
...        ...       ...  ...     ...     ...         ...           ...
253675       1         1   45       0       0           0             0
253676       1         1   18       0       0           0             0
253677       0         0   28       0       0           0             1
253678       1         0   23       0       0           0             0
253679       1         1   25       0       0           1             1

        Fruit  Vegetables  HeavyDrinker  ...  NotAbleToAffordDoctor  \
0           0           1             0  ...                      0
1           0           0             0  ...                      1
2           1           0             0  ...                      1
3           1           1             0  ...                      0
4           1           1             0  ...                      0
...       ...         ...           ...  ...                    ...
```

```
253675       1            1             0  ...                    0
253676       0            0             0  ...                    0
253677       1            0             0  ...                    0
253678       1            1             0  ...                    0
253679       1            0             0  ...                    0

        GeneralHealth  MentalHealth  PhysicalHealth  HardToClimbStairs  \
0                   5            18              15                  1
1                   3             0               0                  0
2                   5            30              30                  1
3                   2             0               0                  0
4                   2             3               0                  0
...               ...           ...             ...                ...
253675              3             0               5                  0
253676              4             0               0                  1
253677              1             0               0                  0
253678              3             0               0                  0
253679              2             0               0                  0

        BiologicalSex  AgeBracket  EducationBracket  IncomeBracket  Zodiac
0                   1           9                 4              3      10
1                   1           7                 6              1      11
2                   1           9                 4              8       2
3                   1          11                 3              6      11
4                   1          11                 5              4       8
...               ...         ...               ...            ...     ...
253675              2           5                 6              7      11
253676              1          11                 2              4       6
253677              1           2                 5              2       5
253678              2           7                 5              1       6
253679              1           9                 6              2       6

[253680 rows x 21 columns]
y:
0         0
1         0
2         0
3         0
4         0
         ..
253675    0
253676    1
253677    0
253678    0
253679    1
Name: Diabetes, Length: 253680, dtype: int64
```

[5]:
```python
# define train and test sets
from sklearn.model_selection import train_test_split
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5,␣
 ↪random_state=42)

clf = Perceptron(tol=1e-3, shuffle=True)

# train model
clf.fit(X_train, y_train)

score = clf.score(X_train, y_train)
print(f'Training Accuracy: {score:2f}')
```

```
Training Accuracy: 0.860311
```

Seeing the loss at 0.86, we can now test the model on the training data and see how the single layer perceptron performs

[99]:
```python
# import metrics to analyze model performance on the test set
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
 ↪f1_score, matthews_corrcoef, confusion_matrix, roc_auc_score,␣
 ↪average_precision_score

y_val_pred = clf.predict(X_val)

# calc metrics
perc_accuracy = accuracy_score(y_val, y_val_pred)
perc_confMatrix = confusion_matrix(y_val, y_val_pred)
perc_precision = precision_score(y_val, y_val_pred, zero_division=1)
perc_recall = recall_score(y_val, y_val_pred)
perc_f1 = f1_score(y_val, y_val_pred)
perc_mcc = matthews_corrcoef(y_val, y_val_pred)

# print metrics
print('******************** Perceptron Test Metrics ********************')
print(f'Perceptron Test Accuracy: {perc_accuracy}')
print(f'Perceptron Test Confusion Matrix: \n{perc_confMatrix}')
print(f'Perceptron Test Precision: {perc_precision}')
print(f'Perceptron Test Recall: {perc_recall}')
print(f'Perceptron Test F1: {perc_f1}')
print(f'Perceptron Test MCC: {perc_mcc}')
```

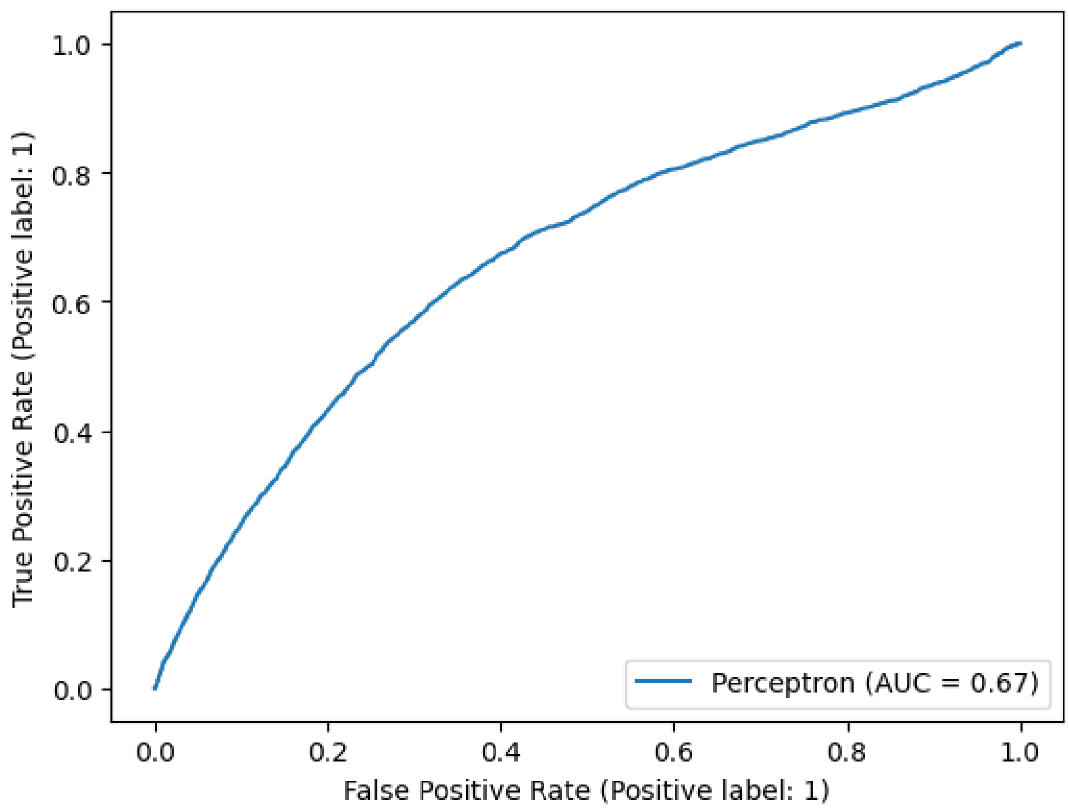
```
******************** Perceptron Test Metrics ********************
Perceptron Test Accuracy: 0.8615184484389783
Perceptron Test Confusion Matrix:
[[21855     0]
 [ 3513     0]]
```

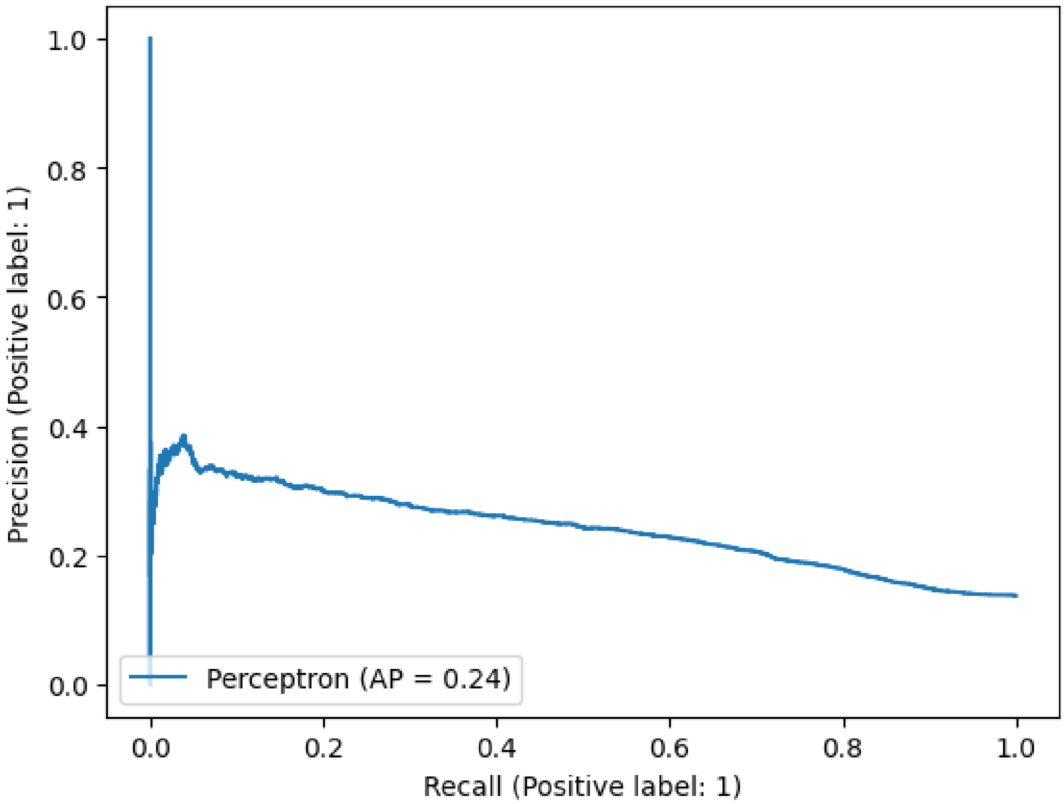
```
Perceptron Test Precision: 1.0
Perceptron Test Recall: 0.0
Perceptron Test F1: 0.0
Perceptron Test MCC: 0.0
```

[7]:
```python
# import roc and prc
from sklearn.metrics import PrecisionRecallDisplay, RocCurveDisplay

ROC = RocCurveDisplay
PRC = PrecisionRecallDisplay
```

[8]:
```python
# print AUROC and AUPRC
ROC.from_estimator(clf, X_test, y_test)
PRC.from_estimator(clf, X_test, y_test)
```

[8]:
```
<sklearn.metrics._plot.precision_recall_curve.PrecisionRecallDisplay at
0x29ac5dc50>
```





1.2 Question #2

Build and train a feedforward neural network with at least one hidden layer to classify diabetes from the rest of the dataset. Make sure to try different numbers of hidden layers and different activation functions (at a minimum reLU and sigmoid). Doing so: How does AUC vary as a function of the number of hidden layers and is it dependent on the kind of activation function used (make sure to include "no activation function" in your comparison). How does this network perform relative to the Perceptron?

1.2.1 Solution

First, lets start by defining the model.

[9]:
```
# import nn module from torch
from torch import nn

# define our input dimensions (dimension of data)
numFeatures = df.shape[1] - 1 # 21
numClasses = 2
dense_nodes = 46 # keep things simple

# define our weights
```

```
# w = torch.Tensor(numFeatures)

learning_rate = 1e-2

model = nn.Sequential(
    nn.Linear(numFeatures, dense_nodes),
    nn.ReLU(),
    nn.Linear(dense_nodes, numClasses),
    nn.Softmax(dim=1) # return 1d-array with probability for either class
)
```

[10]:
```
print(model)
```

```
Sequential(
  (0): Linear(in_features=21, out_features=46, bias=True)
  (1): ReLU()
  (2): Linear(in_features=46, out_features=2, bias=True)
  (3): Softmax(dim=1)
)
```

[11]:
```
# let's process data and initialize our loss functions
numRows = df.shape[0]
X_train_nn = torch.as_tensor(X_train.values, dtype=torch.float)
X_test_nn = torch.as_tensor(X_test.values, dtype=torch.float)

# print to log data before training
print(X_train_nn)
print(X_test_nn)
print(f'X shape for forward feeding nn training: {X_train_nn.shape}')
print(f'X shape for forward feeding nn testing: {X_test_nn.shape}')
```

```
tensor([[ 0.,  1., 20.,  ...,  6.,  8., 12.],
        [ 0.,  0., 34.,  ...,  5.,  8.,  1.],
        [ 1.,  1., 24.,  ...,  5.,  6.,  5.],
        ...,
        [ 0.,  1., 25.,  ...,  6.,  8.,  7.],
        [ 0.,  0., 23.,  ...,  6.,  6., 12.],
        [ 1.,  0., 35.,  ...,  5.,  6.,  6.]])
tensor([[ 1.,  1., 28.,  ...,  6.,  8.,  8.],
        [ 0.,  1., 33.,  ...,  6.,  5., 11.],
        [ 0.,  1., 18.,  ...,  6.,  7., 11.],
        ...,
        [ 1.,  0., 28.,  ...,  5.,  8.,  2.],
        [ 0.,  0., 20.,  ...,  4.,  4., 12.],
        [ 0.,  1., 38.,  ...,  4.,  8.,  7.]])
X shape for forward feeding nn training: torch.Size([202944, 21])
X shape for forward feeding nn testing: torch.Size([25368, 21])
```

[12]:
```
# set labels
labels = y_train.unique()

y_train_nn = torch.as_tensor(y_train.values, dtype=torch.long)
y_test_nn = torch.as_tensor(y_test.values, dtype=torch.long)

# once again, print processed data to make sure everything was processed properly
print(labels)
print(y_train_nn)
print(y_train_nn)

# some quick validation
if X_train_nn.shape[0] == y_train_nn.shape[0]:
    print(f'validation success:\nX: {X_train_nn.shape[0]}, y: {y_train_nn.
 ↪shape}')
else:
    print(f'mismatch in data size:\nX{X_train_nn.shape[0]}, y: {y_train_nn.
 ↪shape}')
```

```
[0 1]
tensor([0, 0, 1,  ..., 0, 0, 1])
tensor([0, 0, 1,  ..., 0, 0, 1])
validation success:
X: 202944, y: torch.Size([202944])
```

[13]:
```
# import optim for gd
from torch import optim

print(torch.__version__)

# define loss function and optimizer
criterion = nn.CrossEntropyLoss() # cross entropy
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate) # stochastic␣
 ↪gradient descent
```

```
2.6.0
```

Now, we can define our forward feeding function

[95]:
```
# import IPython for display during training loop
from IPython import display
import time

# training function
def train(model, X, y, criterion, optimizer, num_epochs=200):
    start_time = time.time()

    for epoch in range(num_epochs):  # run for num_epochs epochs
```

```
        epoch_start_time = time.time()  # Start the timer for each epoch

        model.train()  # Ensure the model is in training mode

        # Pass training data through the model
        y_pred = model(X)

        # Compute loss
        loss = criterion(y_pred, y)

        # Compute model predictions and accuracy
        _, predicted = torch.max(y_pred, 1)  # Get predicted class indices
        acc = (predicted == y).sum().item() / y.size(0)  # Accuracy calculation

        # Calculate epoch time
        epoch_time = time.time() - epoch_start_time

        # Calculate estimated time remaining
        elapsed_time = time.time() - start_time
        avg_epoch_time = elapsed_time / (epoch + 1)  # Average time per epoch so far
        remaining_epochs = num_epochs - (epoch + 1)
        estimated_time_remaining = avg_epoch_time * remaining_epochs

        # Convert the estimated time remaining to minutes and seconds
        minutes_remaining = int(estimated_time_remaining // 60)
        seconds_remaining = int(estimated_time_remaining % 60)

        # Print current epoch, loss, accuracy, and estimated time remaining
        print(f'[EPOCH]: {epoch + 1}/{num_epochs}, [LOSS]: {loss.item():.6f}, [ACCURACY]: {acc:.3f}')
        print(f'Estimated time remaining: {minutes_remaining}m {seconds_remaining}s')

        # Optionally clear output for progress visualization (if running in a notebook)
        display.clear_output(wait=True)

        # Zero the gradients before the backward pass
        optimizer.zero_grad()

        # Backpropagation: compute gradients
        loss.backward()

        # Update the model parameters
        optimizer.step()
```

```
    total_time = time.time() - start_time
    total_minutes = total_time / 60
    print(f'\nTraining completed in {total_minutes:.2f} minutes')

# testing function
def test(model, X, y, criterion):
    model.eval()

    with torch.no_grad():
        y_pred = model(X)
        loss = criterion(y_pred, y)

        # predicted class labels
        _, predicted = torch.max(y_pred, 1)

        # convert to numpy for sklearn
        y_true_np = y.cpu().numpy()
        y_pred_np = predicted.cpu().numpy()

        # calc class probabilities
        y_score = torch.softmax(y_pred, dim=1)  # default to multi-class softmax
        y_score_np = y_score.cpu().numpy()

        # Basic metrics
        acc = accuracy_score(y_true_np, y_pred_np)
        precision = precision_score(y_true_np, y_pred_np, average='macro')
        recall = recall_score(y_true_np, y_pred_np, average='macro')
        f1 = f1_score(y_true_np, y_pred_np, average='macro')

        # Detect binary vs multi-class
        is_binary = y_score_np.shape[1] == 2

        try:
            if is_binary:
                # for binary classification, use score of positive class (class␣
↪1)
                auroc = roc_auc_score(y_true_np, y_score_np[:, 1])
                auprc = average_precision_score(y_true_np, y_score_np[:, 1])

                RocCurveDisplay.from_predictions(y_true_np, y_score_np[:, 1])
                plt.title("ROC Curve")
                plt.show()

                PrecisionRecallDisplay.from_predictions(y_true_np, y_score_np[:,␣
↪1])
                plt.title("Precision-Recall Curve")
                plt.show()
```

```
        else:
            # Multi-class case: One-vs-Rest
            auroc = roc_auc_score(y_true_np, y_score_np, multi_class='ovr')
            auprc = average_precision_score(y_true_np, y_score_np,␣
↪average='macro')

            # Binarize labels for plotting
            y_true_bin = label_binarize(y_true_np,␣
↪classes=list(range(y_score_np.shape[1])))

            for i in range(y_score_np.shape[1]):
                RocCurveDisplay.from_predictions(y_true_bin[:, i],␣
↪y_score_np[:, i])
                plt.title(f"ROC Curve - Class {i}")
                plt.show()

                PrecisionRecallDisplay.from_predictions(y_true_bin[:, i],␣
↪y_score_np[:, i])
                plt.title(f"Precision-Recall Curve - Class {i}")
                plt.show()

    except ValueError as e:
        auroc = float('nan')
        auprc = float('nan')
        print(f"AUROC/AUPRC could not be computed: {e}")

    # Print metrics
    print(f'Test Loss: {loss.item():.6f}')
    print(f'Accuracy: {100 * acc:.2f}%')
    print(f'Precision: {precision:.4f}, Recall: {recall:.4f}, F1 Score: {f1:.
↪4f}')
    print(f'AUROC: {auroc:.4f}, AUPRC: {auprc:.4f}')

    return acc, precision, recall, f1, auroc, auprc
```

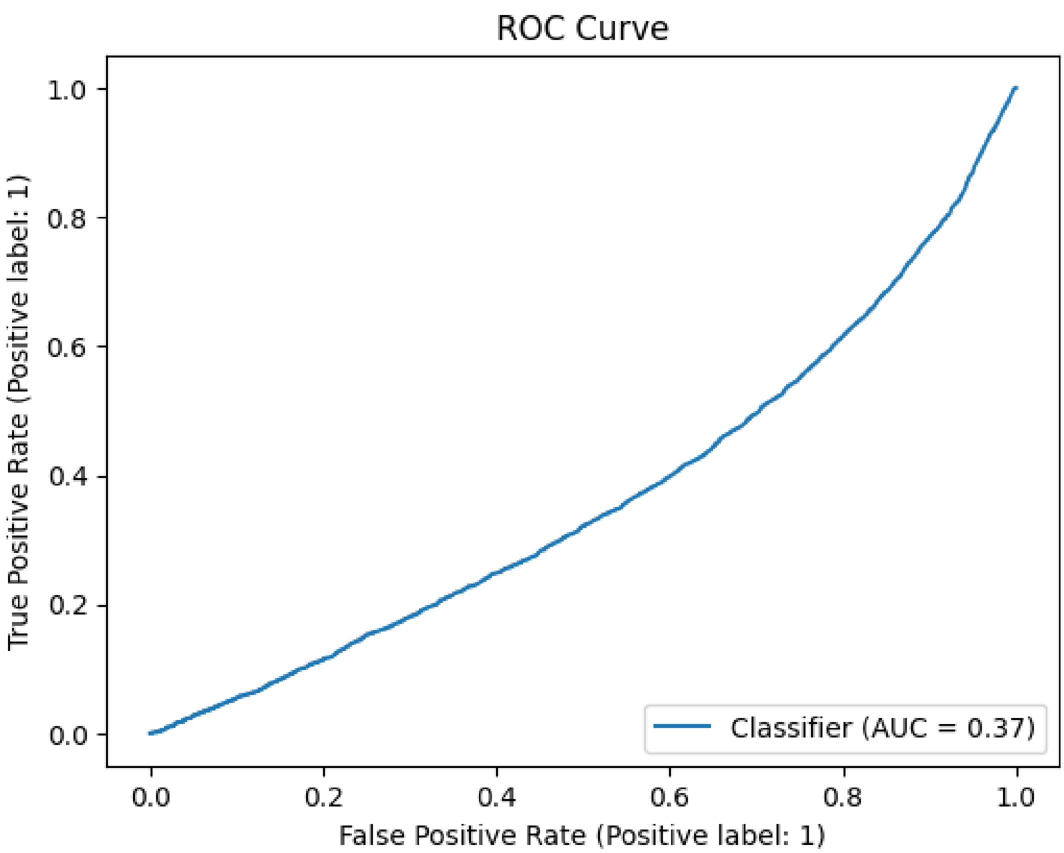
[96]:
```
# train the model
train(model, X_train_nn, y_train_nn, criterion, optimizer)
```

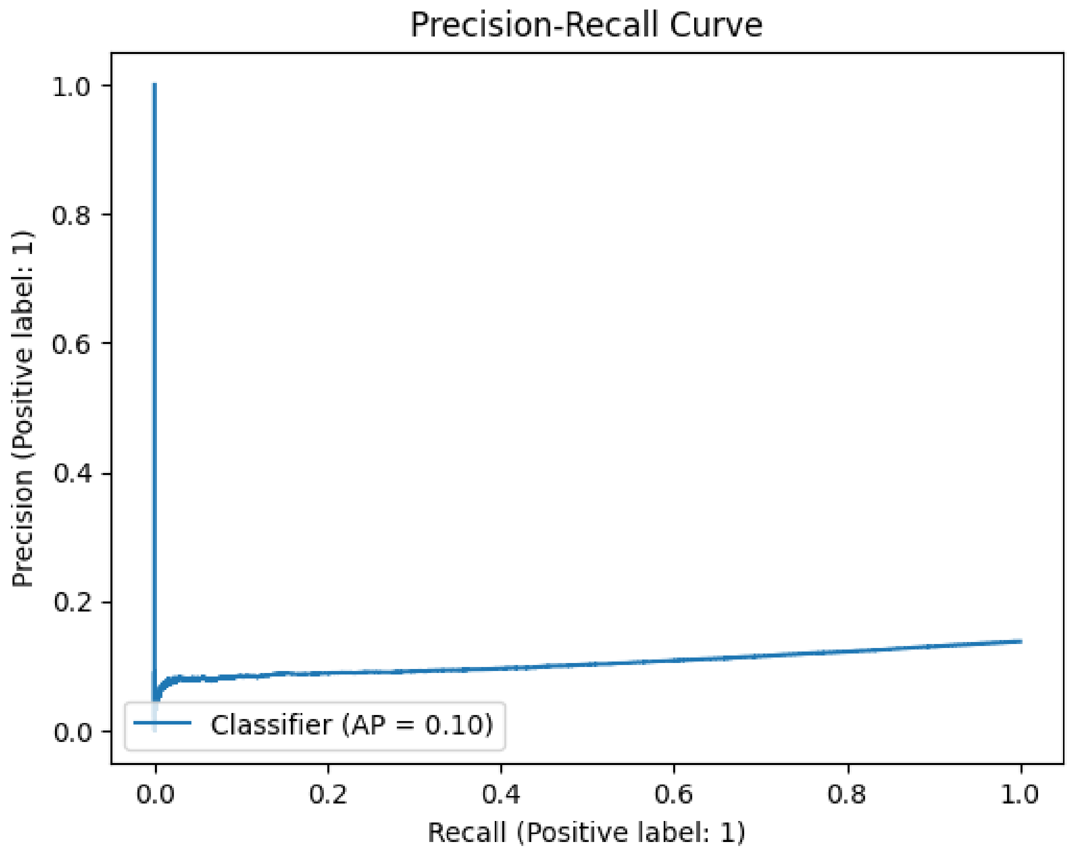
```
Training completed in 0.14 minutes
```

[100]:
```
# test the model
acc, prec, recall, f1, auroc, auprc = test(model, X_test_nn, y_test_nn,␣
↪criterion)

print(f'********** FNN Metrics **********')
```

```python
print(f'MLP Accuracy: {acc}')
print(f'MLP Precision: {prec}')
print(f'MLP Recall: {recall}')
print(f'MLP F1-Score: {f1}')
```

```
/opt/anaconda3/lib/python3.11/site-
packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```






```
Test Loss: 0.450737
Accuracy: 86.27%
Precision: 0.4313, Recall: 0.5000, F1 Score: 0.4631
AUROC: 0.3675, AUPRC: 0.1040
********** FNN Metrics **********
MLP Accuracy: 0.8626616209397666
MLP Precision: 0.4313308104698833
MLP Recall: 0.5
MLP F1-Score: 0.4631338356048421
```

Calculate the metrics of the model's performance using sklearn.metrics

[18]:
```
from binary_nn_model import BinaryFNN

X_train_bnn = torch.as_tensor(X_train.values, dtype=torch.float32)
X_test_bnn = torch.as_tensor(X_train.values, dtype=torch.float32)

y_train_bnn = torch.as_tensor(y_train.values, dtype=torch.long)
y_test_bnn = torch.as_tensor(y_test.values, dtype=torch.long)
```

```python
bnn = BinaryFNN(X_train_bnn, y_train_bnn, activation_func = 'sig')

bnn.X = X_train_bnn
bnn.y = y_train_bnn

bnn._train(X_train_bnn, num_epochs=200)
print()
bnn.test(X_test_bnn, y_test_bnn)
```

```
Epoch [1/200], Loss: 0.5284
Epoch [2/200], Loss: 0.5163
Epoch [3/200], Loss: 0.5056
Epoch [4/200], Loss: 0.4961
Epoch [5/200], Loss: 0.4875
Epoch [6/200], Loss: 0.4799
Epoch [7/200], Loss: 0.4731
Epoch [8/200], Loss: 0.4669
Epoch [9/200], Loss: 0.4614
Epoch [10/200], Loss: 0.4565
Epoch [11/200], Loss: 0.4520
Epoch [12/200], Loss: 0.4479
Epoch [13/200], Loss: 0.4443
Epoch [14/200], Loss: 0.4410
Epoch [15/200], Loss: 0.4380
Epoch [16/200], Loss: 0.4352
Epoch [17/200], Loss: 0.4327
Epoch [18/200], Loss: 0.4305
Epoch [19/200], Loss: 0.4284
Epoch [20/200], Loss: 0.4265
Epoch [21/200], Loss: 0.4248
Epoch [22/200], Loss: 0.4232
Epoch [23/200], Loss: 0.4217
Epoch [24/200], Loss: 0.4204
Epoch [25/200], Loss: 0.4192
Epoch [26/200], Loss: 0.4181
Epoch [27/200], Loss: 0.4170
Epoch [28/200], Loss: 0.4161
Epoch [29/200], Loss: 0.4152
Epoch [30/200], Loss: 0.4144
Epoch [31/200], Loss: 0.4137
Epoch [32/200], Loss: 0.4130
Epoch [33/200], Loss: 0.4123
Epoch [34/200], Loss: 0.4118
Epoch [35/200], Loss: 0.4112
Epoch [36/200], Loss: 0.4107
Epoch [37/200], Loss: 0.4103
Epoch [38/200], Loss: 0.4098
Epoch [39/200], Loss: 0.4094
```

```
Epoch [40/200], Loss: 0.4091
Epoch [41/200], Loss: 0.4087
Epoch [42/200], Loss: 0.4084
Epoch [43/200], Loss: 0.4081
Epoch [44/200], Loss: 0.4078
Epoch [45/200], Loss: 0.4076
Epoch [46/200], Loss: 0.4073
Epoch [47/200], Loss: 0.4071
Epoch [48/200], Loss: 0.4069
Epoch [49/200], Loss: 0.4067
Epoch [50/200], Loss: 0.4065
Epoch [51/200], Loss: 0.4064
Epoch [52/200], Loss: 0.4062
Epoch [53/200], Loss: 0.4061
Epoch [54/200], Loss: 0.4059
Epoch [55/200], Loss: 0.4058
Epoch [56/200], Loss: 0.4057
Epoch [57/200], Loss: 0.4056
Epoch [58/200], Loss: 0.4055
Epoch [59/200], Loss: 0.4054
Epoch [60/200], Loss: 0.4053
Epoch [61/200], Loss: 0.4052
Epoch [62/200], Loss: 0.4051
Epoch [63/200], Loss: 0.4051
Epoch [64/200], Loss: 0.4050
Epoch [65/200], Loss: 0.4049
Epoch [66/200], Loss: 0.4049
Epoch [67/200], Loss: 0.4048
Epoch [68/200], Loss: 0.4048
Epoch [69/200], Loss: 0.4047
Epoch [70/200], Loss: 0.4047
Epoch [71/200], Loss: 0.4046
Epoch [72/200], Loss: 0.4046
Epoch [73/200], Loss: 0.4046
Epoch [74/200], Loss: 0.4045
Epoch [75/200], Loss: 0.4045
Epoch [76/200], Loss: 0.4045
Epoch [77/200], Loss: 0.4044
Epoch [78/200], Loss: 0.4044
Epoch [79/200], Loss: 0.4044
Epoch [80/200], Loss: 0.4043
Epoch [81/200], Loss: 0.4043
Epoch [82/200], Loss: 0.4043
Epoch [83/200], Loss: 0.4043
Epoch [84/200], Loss: 0.4043
Epoch [85/200], Loss: 0.4042
Epoch [86/200], Loss: 0.4042
Epoch [87/200], Loss: 0.4042
```

```
Epoch [88/200], Loss: 0.4042
Epoch [89/200], Loss: 0.4042
Epoch [90/200], Loss: 0.4042
Epoch [91/200], Loss: 0.4041
Epoch [92/200], Loss: 0.4041
Epoch [93/200], Loss: 0.4041
Epoch [94/200], Loss: 0.4041
Epoch [95/200], Loss: 0.4041
Epoch [96/200], Loss: 0.4041
Epoch [97/200], Loss: 0.4041
Epoch [98/200], Loss: 0.4041
Epoch [99/200], Loss: 0.4041
Epoch [100/200], Loss: 0.4041
Epoch [101/200], Loss: 0.4040
Epoch [102/200], Loss: 0.4040
Epoch [103/200], Loss: 0.4040
Epoch [104/200], Loss: 0.4040
Epoch [105/200], Loss: 0.4040
Epoch [106/200], Loss: 0.4040
Epoch [107/200], Loss: 0.4040
Epoch [108/200], Loss: 0.4040
Epoch [109/200], Loss: 0.4040
Epoch [110/200], Loss: 0.4040
Epoch [111/200], Loss: 0.4040
Epoch [112/200], Loss: 0.4040
Epoch [113/200], Loss: 0.4040
Epoch [114/200], Loss: 0.4040
Epoch [115/200], Loss: 0.4040
Epoch [116/200], Loss: 0.4040
Epoch [117/200], Loss: 0.4039
Epoch [118/200], Loss: 0.4039
Epoch [119/200], Loss: 0.4039
Epoch [120/200], Loss: 0.4039
Epoch [121/200], Loss: 0.4039
Epoch [122/200], Loss: 0.4039
Epoch [123/200], Loss: 0.4039
Epoch [124/200], Loss: 0.4039
Epoch [125/200], Loss: 0.4039
Epoch [126/200], Loss: 0.4039
Epoch [127/200], Loss: 0.4039
Epoch [128/200], Loss: 0.4039
Epoch [129/200], Loss: 0.4039
Epoch [130/200], Loss: 0.4039
Epoch [131/200], Loss: 0.4039
Epoch [132/200], Loss: 0.4039
Epoch [133/200], Loss: 0.4039
Epoch [134/200], Loss: 0.4039
Epoch [135/200], Loss: 0.4039
```

```
Epoch [136/200], Loss: 0.4039
Epoch [137/200], Loss: 0.4039
Epoch [138/200], Loss: 0.4039
Epoch [139/200], Loss: 0.4039
Epoch [140/200], Loss: 0.4039
Epoch [141/200], Loss: 0.4039
Epoch [142/200], Loss: 0.4039
Epoch [143/200], Loss: 0.4039
Epoch [144/200], Loss: 0.4039
Epoch [145/200], Loss: 0.4039
Epoch [146/200], Loss: 0.4038
Epoch [147/200], Loss: 0.4038
Epoch [148/200], Loss: 0.4038
Epoch [149/200], Loss: 0.4038
Epoch [150/200], Loss: 0.4038
Epoch [151/200], Loss: 0.4038
Epoch [152/200], Loss: 0.4038
Epoch [153/200], Loss: 0.4038
Epoch [154/200], Loss: 0.4038
Epoch [155/200], Loss: 0.4038
Epoch [156/200], Loss: 0.4038
Epoch [157/200], Loss: 0.4038
Epoch [158/200], Loss: 0.4038
Epoch [159/200], Loss: 0.4038
Epoch [160/200], Loss: 0.4038
Epoch [161/200], Loss: 0.4038
Epoch [162/200], Loss: 0.4038
Epoch [163/200], Loss: 0.4038
Epoch [164/200], Loss: 0.4038
Epoch [165/200], Loss: 0.4038
Epoch [166/200], Loss: 0.4038
Epoch [167/200], Loss: 0.4038
Epoch [168/200], Loss: 0.4038
Epoch [169/200], Loss: 0.4038
Epoch [170/200], Loss: 0.4038
Epoch [171/200], Loss: 0.4038
Epoch [172/200], Loss: 0.4038
Epoch [173/200], Loss: 0.4038
Epoch [174/200], Loss: 0.4038
Epoch [175/200], Loss: 0.4038
Epoch [176/200], Loss: 0.4038
Epoch [177/200], Loss: 0.4038
Epoch [178/200], Loss: 0.4038
Epoch [179/200], Loss: 0.4038
Epoch [180/200], Loss: 0.4038
Epoch [181/200], Loss: 0.4038
Epoch [182/200], Loss: 0.4038
Epoch [183/200], Loss: 0.4038
```

```
Epoch [184/200], Loss: 0.4038
Epoch [185/200], Loss: 0.4037
Epoch [186/200], Loss: 0.4037
Epoch [187/200], Loss: 0.4037
Epoch [188/200], Loss: 0.4037
Epoch [189/200], Loss: 0.4037
Epoch [190/200], Loss: 0.4037
Epoch [191/200], Loss: 0.4037
Epoch [192/200], Loss: 0.4037
Epoch [193/200], Loss: 0.4037
Epoch [194/200], Loss: 0.4037
Epoch [195/200], Loss: 0.4037
Epoch [196/200], Loss: 0.4037
Epoch [197/200], Loss: 0.4037
Epoch [198/200], Loss: 0.4037
Epoch [199/200], Loss: 0.4037
Epoch [200/200], Loss: 0.4037

Test Loss: 0.403708
```

1.3 Question #3

Build and train a "deep" network (at least 2 hidden layers) to classify diabetes from the rest of the dataset. Given the nature of this dataset, is there a benefit of using a CNN for the classification?

1.4 Functions to implement and what they tell us:

1.4.1 Z-Score:

$$z = \frac{x - \mu_x}{\sigma}$$

Transforms the distribution of a vector into the normal distribution with $\mathbb{E}(X) = 0$ and $\sigma = 1$. ### Min-Max Scaling:

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

Scales our data from 0 to 1, which can be helpful given working with a binary class distribution of output labels. ### Cosine-Similarity:

$$\theta = \arccos(\frac{\vec{u} \cdot \vec{v}}{||\vec{u}|| \cdot ||\vec{v}||})$$

Computes the angle between two vectors. Given $\vec{v} \cdot \vec{u} = 0$ tells if the vectors are orthogonal, if $\cos(\theta) = 0$, then we know the vectors are orthogonal. Vice-versa, if $\cos(\theta) \approx 1$, then we know they lie within the same span - indicating some linear relationship between the vectors. This can help tremendously with dimensionality reduction with Principal Component Analysis before feeding the data into our CNN.

[19]:
```
# let's visualize the data first, once again - visualizing the relationships␣
↪between the features
# we'll try something different to analyze the input features
```

```
# z-score function
def z_score(x):
    return (x - np.mean(x))/np.std(x)

# min-max scaling function
def min_max_scaling(x):
    return (x - np.min(x))/(np.max(x) - np.min(x))

# cosine similarity == arccos((u * v)/(||u|| * ||v||))
def cosine_similarity(u, v):
    '''
    we'll convert everything to numpy arrays to make
    the computations easier
    '''
    # compute dot product
    u = np.array(u)
    v = np.array(v)
    dot = np.dot(u, v)

    # calc magnitudes of u, v
    norm_u = np.linalg.norm(u)
    norm_v = np.linalg.norm(v)

    # return theta
    return np.arccos((dot)/(norm_u * norm_v))
```

[20]:

```
# test our cosine similarity function on general and mental health
gh_mh_theta = cosine_similarity(X.iloc[:, 12], X.iloc[:, 13])
print(f'Cosine Similarity of General Health, Mental Health Features:
 ↪\n{gh_mh_theta:.2f}')

print()

# test on general health, physical health
gh_ph_theta = cosine_similarity(X.iloc[:, 12], X.iloc[:, 14])
print(f'Cosine Similarity of General Health, Physical Health Features:
 ↪\n{gh_ph_theta:.2f}')
```

```
Cosine Similarity of General Health, Mental Health Features:
1.08

Cosine Similarity of General Health, Physical Health Features:
0.94
```

Let's do a little more pre-processing before testing all of the functions on our features.

It makes sense to center each vector of the input matrix around the origin so we get accurate cosine similarity computations between each vector.

Let's first start with checking the mean of each vector and outputting it to the notebook.

[21]:
```
# check means of each vector
X_arr = np.array(X)
for feature in range(X_arr.shape[1]):
    print(f"Mean of {df.columns[feature]} feature: {np.mean(X_arr[:, feature]):.
 ↪3f}")
```

```
Mean of Diabetes feature: 0.429
Mean of HighBP feature: 0.424
Mean of HighChol feature: 28.382
Mean of BMI feature: 0.443
Mean of Smoker feature: 0.041
Mean of Stroke feature: 0.094
Mean of Myocardial feature: 0.757
Mean of PhysActivity feature: 0.634
Mean of Fruit feature: 0.811
Mean of Vegetables feature: 0.056
Mean of HeavyDrinker feature: 0.951
Mean of HasHealthcare feature: 0.084
Mean of NotAbleToAffordDoctor feature: 2.511
Mean of GeneralHealth feature: 3.185
Mean of MentalHealth feature: 4.242
Mean of PhysicalHealth feature: 0.168
Mean of HardToClimbStairs feature: 1.440
Mean of BiologicalSex feature: 8.032
Mean of AgeBracket feature: 5.050
Mean of EducationBracket feature: 6.054
Mean of IncomeBracket feature: 6.504
```

Seeing the means, we can now make some assumptions about the dataset:

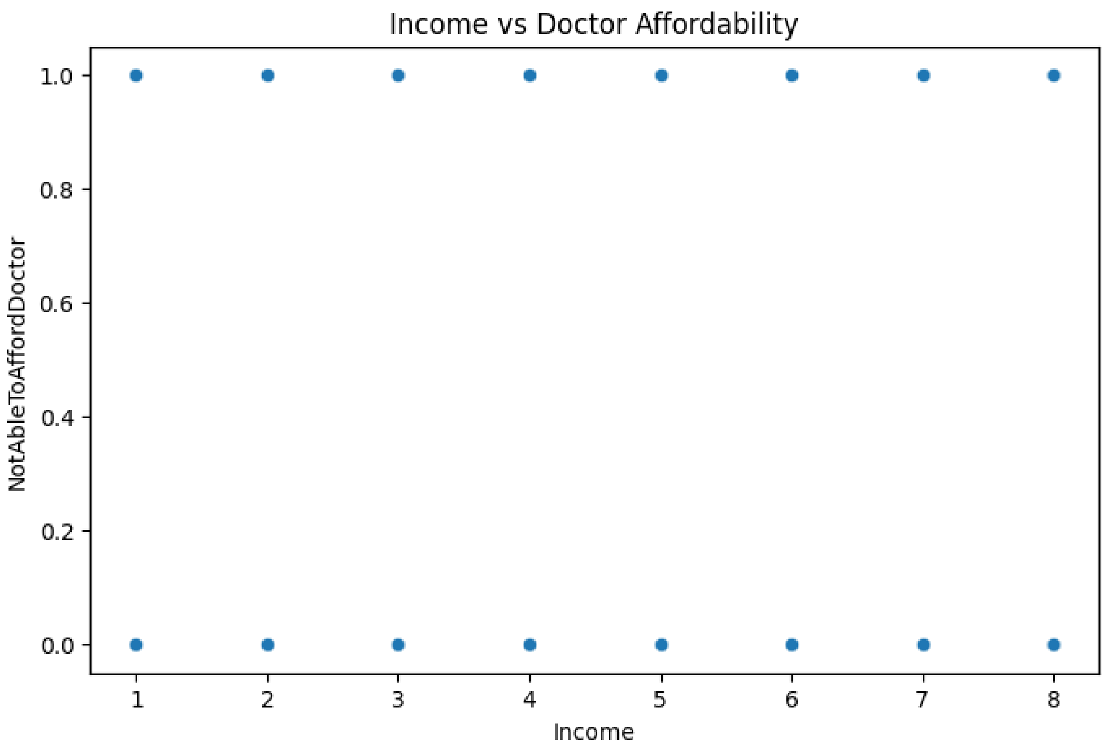
```
*Diabetes mean of 0.429 indicates majority of samples are 0, indicating class imbalance (roughly
*BP mean ~ Diabetes mean, could be a direct contributor to indicating
*Biological Sex feature indicates the majority label for this feature is 2? We can check that.
```

[22]:
```
print(df['BiologicalSex'].value_counts())
```

```
1    141974
2    111706
Name: BiologicalSex, dtype: int64
```

[23]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# Add small random noise (jitter) to diabetes
# jittered_diabetes = df['Diabetes'] + np.random.normal(0, 0.05, size=len(df))
```

```
# plot BMI vs Diabetes
plt.figure(figsize=(8, 5))
sns.scatterplot(x='IncomeBracket', y='NotAbleToAffordDoctor', data=df, alpha=0.4)
plt.title('Income vs Doctor Affordability')
plt.xlabel('Income')
plt.ylabel('NotAbleToAffordDoctor')
plt.show()
```



Since the scatter doesn't provide much at all, let's analyze the data using cosine-similarity among the vectors.

[24]:
```
X_unprocessed = df.drop(columns=['Diabetes'])
y_unprocessed = df['Diabetes']

# print to make sure
print(f'X: {X_unprocessed}')
print(f'y: {y_unprocessed}')
```

```
X:          HighBP  HighChol  BMI  Smoker  Stroke  Myocardial  PhysActivity  \
0                1         1   40       1       0           0             0
1                0         0   25       1       0           0             1
2                1         1   28       0       0           0             0
3                1         0   27       0       0           0             1
4                1         1   24       0       0           0             1
```

```
...        ...       ...  ...      ...      ...         ...           ...
253675       1         1   45        0        0           0             0
253676       1         1   18        0        0           0             0
253677       0         0   28        0        0           0             1
253678       1         0   23        0        0           0             0
253679       1         1   25        0        0           1             1

        Fruit  Vegetables  HeavyDrinker  ...  NotAbleToAffordDoctor  \
0           0           1             0  ...                      0
1           0           0             0  ...                      1
2           1           0             0  ...                      1
3           1           1             0  ...                      0
4           1           1             0  ...                      0
...       ...         ...           ...  ...                    ...
253675      1           1             0  ...                      0
253676      0           0             0  ...                      0
253677      1           0             0  ...                      0
253678      1           1             0  ...                      0
253679      1           0             0  ...                      0

        GeneralHealth  MentalHealth  PhysicalHealth  HardToClimbStairs  \
0                   5            18              15                  1
1                   3             0               0                  0
2                   5            30              30                  1
3                   2             0               0                  0
4                   2             3               0                  0
...               ...           ...             ...                ...
253675              3             0               5                  0
253676              4             0               0                  1
253677              1             0               0                  0
253678              3             0               0                  0
253679              2             0               0                  0

        BiologicalSex  AgeBracket  EducationBracket  IncomeBracket  Zodiac
0                   1           9                 4              3      10
1                   1           7                 6              1      11
2                   1           9                 4              8       2
3                   1          11                 3              6      11
4                   1          11                 5              4       8
...               ...         ...               ...            ...     ...
253675              2           5                 6              7      11
253676              1          11                 2              4       6
253677              1           2                 5              2       5
253678              2           7                 5              1       6
253679              1           9                 6              2       6

[253680 rows x 21 columns]
y: 0        0
```

```
1         0
2         0
3         0
4         0
         ..
253675    0
253676    1
253677    0
253678    0
253679    1
Name: Diabetes, Length: 253680, dtype: int64
```

[25]:
```
# convert to np arrays
X_matrix = np.array(X_unprocessed)
y_vec = np.array(y_unprocessed)

print(f'Matrix X: \n{X_matrix}')
print(f'Vector y: \n{y_vec}')
```

```
Matrix X:
[[ 1  1 40 ...  4  3 10]
 [ 0  0 25 ...  6  1 11]
 [ 1  1 28 ...  4  8  2]
 ...
 [ 0  0 28 ...  5  2  5]
 [ 1  0 23 ...  5  1  6]
 [ 1  1 25 ...  6  2  6]]
Vector y:
[0 0 0 ... 0 0 1]
```

Let's scale and then normalize before performing cosine-similarity on the vectors of X

[26]:
```
# scale our continuous data (looking at mean of these vectors)
continuous_features = ['HighChol', 'BMI', 'NotAbleToAffordDoctor',␣
 ↪'GeneralHealth', 'MentalHealth', 'PhysicalHealth',␣
 ↪'HardToClimbStairs','AgeBracket', 'EducationBracket', 'IncomeBracket']

# scale features
for feature in continuous_features:
    X_unprocessed[feature] = min_max_scaling(X_unprocessed[feature])

print(f'Data after scaling: {X_unprocessed}')
```

```
Data after scaling:         HighBP  HighChol       BMI  Smoker  Stroke
Myocardial  PhysActivity  \
0             1       1.0  0.325581       1       0          0            0
1             0       0.0  0.151163       1       0          0            1
2             1       1.0  0.186047       0       0          0            0
3             1       0.0  0.174419       0       0          0            1
```

```
4             1        1.0  0.139535        0        0            0            1
...         ...        ...       ...      ...      ...          ...          ...
253675        1        1.0  0.383721        0        0            0            0
253676        1        1.0  0.069767        0        0            0            0
253677        0        0.0  0.186047        0        0            0            1
253678        1        0.0  0.127907        0        0            0            0
253679        1        1.0  0.151163        0        0            1            1

        Fruit  Vegetables  HeavyDrinker  ...  NotAbleToAffordDoctor  \
0           0           1             0  ...                    0.0
1           0           0             0  ...                    1.0
2           1           0             0  ...                    1.0
3           1           1             0  ...                    0.0
4           1           1             0  ...                    0.0
...       ...         ...           ...  ...                    ...
253675      1           1             0  ...                    0.0
253676      0           0             0  ...                    0.0
253677      1           0             0  ...                    0.0
253678      1           1             0  ...                    0.0
253679      1           0             0  ...                    0.0

        GeneralHealth  MentalHealth  PhysicalHealth  HardToClimbStairs  \
0                1.00           0.6        0.500000                1.0
1                0.50           0.0        0.000000                0.0
2                1.00           1.0        1.000000                1.0
3                0.25           0.0        0.000000                0.0
4                0.25           0.1        0.000000                0.0
...               ...           ...             ...                ...
253675           0.50           0.0        0.166667                0.0
253676           0.75           0.0        0.000000                1.0
253677           0.00           0.0        0.000000                0.0
253678           0.50           0.0        0.000000                0.0
253679           0.25           0.0        0.000000                0.0

        BiologicalSex  AgeBracket  EducationBracket  IncomeBracket  Zodiac
0                   1    0.666667               0.6       0.285714      10
1                   1    0.500000               1.0       0.000000      11
2                   1    0.666667               0.6       1.000000       2
3                   1    0.833333               0.4       0.714286      11
4                   1    0.833333               0.8       0.428571       8
...               ...         ...               ...            ...     ...
253675              2    0.333333               1.0       0.857143      11
253676              1    0.833333               0.2       0.428571       6
253677              1    0.083333               0.8       0.142857       5
253678              2    0.500000               0.8       0.000000       6
253679              1    0.666667               1.0       0.142857       6

[253680 rows x 21 columns]
```

[27]:
```
# check means, standardize, then check means once again
X_matrix = np.array(X_unprocessed)
for feature in range(X_arr.shape[1]):
    print(f"Mean of {df.columns[feature]} feature: {np.mean(X_matrix[:,␣
↪feature]):.3f}")

# standardize
print()
for feature in range(X_matrix.shape[1]):
    X_matrix[:, feature] = z_score(X_matrix[:, feature])

# check again
for feature in range(X_matrix.shape[1]):
    print(f'Standardized Mean and Std of {df.columns[feature]} feature: {np.
↪mean(X_matrix[:, feature]):.3f} | Std: {np.std(X_matrix[:, feature]):.2f}')
```

```
Mean of Diabetes feature: 0.429
Mean of HighBP feature: 0.424
Mean of HighChol feature: 0.190
Mean of BMI feature: 0.443
Mean of Smoker feature: 0.041
Mean of Stroke feature: 0.094
Mean of Myocardial feature: 0.757
Mean of PhysActivity feature: 0.634
Mean of Fruit feature: 0.811
Mean of Vegetables feature: 0.056
Mean of HeavyDrinker feature: 0.951
Mean of HasHealthcare feature: 0.084
Mean of NotAbleToAffordDoctor feature: 0.378
Mean of GeneralHealth feature: 0.106
Mean of MentalHealth feature: 0.141
Mean of PhysicalHealth feature: 0.168
Mean of HardToClimbStairs feature: 1.440
Mean of BiologicalSex feature: 0.586
Mean of AgeBracket feature: 0.810
Mean of EducationBracket feature: 0.722
Mean of IncomeBracket feature: 6.504

Standardized Mean and Std of Diabetes feature: -0.000 | Std: 1.00
Standardized Mean and Std of HighBP feature: 0.000 | Std: 1.00
Standardized Mean and Std of HighChol feature: 0.000 | Std: 1.00
Standardized Mean and Std of BMI feature: 0.000 | Std: 1.00
Standardized Mean and Std of Smoker feature: -0.000 | Std: 1.00
Standardized Mean and Std of Stroke feature: 0.000 | Std: 1.00
Standardized Mean and Std of Myocardial feature: 0.000 | Std: 1.00
Standardized Mean and Std of PhysActivity feature: 0.000 | Std: 1.00
Standardized Mean and Std of Fruit feature: -0.000 | Std: 1.00
Standardized Mean and Std of Vegetables feature: 0.000 | Std: 1.00
```

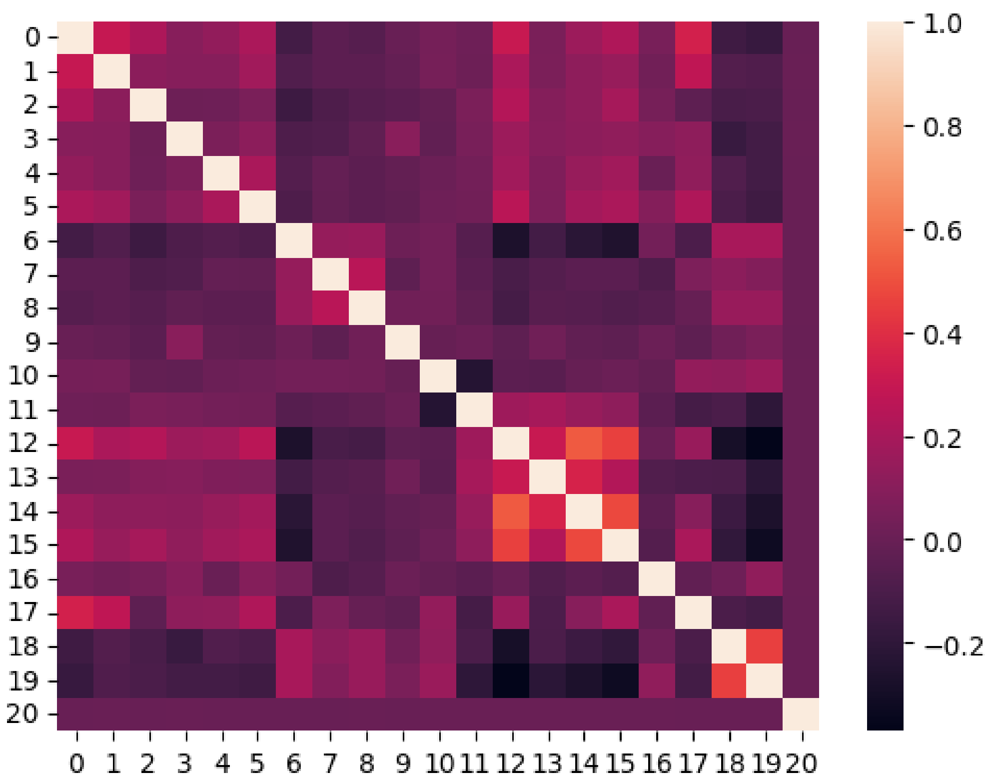
```
Standardized Mean and Std of HeavyDrinker feature: 0.000 | Std: 1.00
Standardized Mean and Std of HasHealthcare feature: -0.000 | Std: 1.00
Standardized Mean and Std of NotAbleToAffordDoctor feature: -0.000 | Std: 1.00
Standardized Mean and Std of GeneralHealth feature: 0.000 | Std: 1.00
Standardized Mean and Std of MentalHealth feature: -0.000 | Std: 1.00
Standardized Mean and Std of PhysicalHealth feature: 0.000 | Std: 1.00
Standardized Mean and Std of HardToClimbStairs feature: 0.000 | Std: 1.00
Standardized Mean and Std of BiologicalSex feature: -0.000 | Std: 1.00
Standardized Mean and Std of AgeBracket feature: -0.000 | Std: 1.00
Standardized Mean and Std of EducationBracket feature: 0.000 | Std: 1.00
Standardized Mean and Std of IncomeBracket feature: -0.000 | Std: 1.00
```

We can now see the matrix is mean-centered around 0.

Let's calculate the cosine similarity between the vectors to see what components we can turn into prinicpal components.

[28]:
```python
# get our covariance matrix
cov = np.cov(X_matrix, rowvar=False)
print(f'Covariance Matrix Shape: {cov.shape}')

# plot heatmap of covariance
sns.heatmap(cov)
plt.show()
```

```
Covariance Matrix Shape: (21, 21)
```



Since we don't know how many principal components we want yet, we'll extract the principal components first and then use cosine-similarity to essentially validate PCA on our covariance matrix.

We can use our covariance matrix as the "input" for SVD, where we will work to decompose the matrix into the following:

$$A = U\Sigma V^T$$

or for our case:

$$Cov(X) = U\Sigma V^T$$

[29]:
```
# compute pca using covariance matrix
def PCA(A):
    eigenvalues, eigenvectors = np.linalg.eig(A)

    # index the eigenvalues in descending order
    idx = eigenvalues.argsort()[::-1]

    # Sort the eigenvalues in descending order
    eigenvalues = eigenvalues[idx]

    # sort the corresponding eigenvectors accordingly
    eigenvectors = eigenvectors[:, idx]

    explained_var = np.cumsum(eigenvalues) / np.sum(eigenvalues)
    n_components = np.argmax(explained_var >= 0.60) + 1

    # return our optimal # principal components
    return n_components, eigenvectors, explained_var
```

[30]:
```
numPC, eigenvectors, explained_var = PCA(cov)
print(numPC)
```

```
9
```

[31]:
```
principal_components = eigenvectors[:, :numPC]
print(f'Shape of principal components: {principal_components.shape}')
X_pca = np.dot(X_matrix, principal_components)

print(f'Shape of X_pca: {X_pca.shape}')
print(f'X_pca: \n{X_pca}')
```

```
Shape of principal components: (21, 9)
Shape of X_pca: (253680, 9)
X_pca:
[[-4.77269689  0.49396039 -0.26774082 ... -1.75722652 -1.06815029
   0.58818862]
```

```
 [-0.79262428  4.25294799  1.25190507 ...  0.65922029 -1.26349008
  -1.79592832]
 [-4.93947004  1.92849307 -1.90017037 ... -0.44131509  1.30724296
  -1.74182326]
 ...
 [ 1.30839857  1.85667356  0.27076275 ...  0.73871767  0.49055108
  -0.39166565]
 [-0.45784354  0.15712272  0.15751728 ...  0.04223485  0.12727493
   0.95056459]
 [-0.77306394 -1.59863759  0.25510217 ...  1.04505378  0.18712582
  -1.89498704]]
```

[32]:
```
# double check the # of samples matches w # labels
if (X_pca.shape[0] == y.shape[0]):
    print(f'Shapes match: X = {X_pca.shape[0]}, y = {y.shape[0]}')
else:
    raise ValueError(f'Shape mismatch: X = {X_pca.shape[0]}, y = {y.shape[0]}')
```

```
Shapes match: X = 253680, y = 253680
```

Now that we the original input matrix transformed after performing PCA, we can import our CNN class and test it on the PCA transformed X (I'm hoping so bad that this works lol)

[33]:
```
# split dataset
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_pca, y, test_size = 0.2)

# convert data to pytorch tensors
# training data
X_train_tensor = torch.as_tensor(X_train, dtype=torch.float32)
y_train_tensor = torch.as_tensor(y_train.values, dtype=torch.long)
X_train_tensor = X_train_tensor.unsqueeze(1)

# test data
X_test_tensor = torch.as_tensor(X_test, dtype=torch.float32)
y_test_tensor = torch.as_tensor(y_test.values, dtype=torch.long)
X_test_tensor = X_test_tensor.unsqueeze(1)

print("Input shape:", X_train_tensor.shape)
print("Test shape:", X_test_tensor.shape)
```

```
Input shape: torch.Size([202944, 1, 9])
Test shape: torch.Size([50736, 1, 9])
```

[34]:
```
# import our model
from cnn_model import ConvolutionalNeuralNetwork

in_channel = X_train.shape[1]
```

```
out_channel = 2

cnn = ConvolutionalNeuralNetwork(X_train_tensor, y_train_tensor, in_channel,␣
 ↪out_channel)

cnn.X = X_train_tensor
cnn.y = y_train_tensor
```

[35]:
```
print("Input shape:", X_train_tensor.shape)
```

```
Input shape: torch.Size([202944, 1, 9])
```

[36]:
```
# train the model
cnn._train(num_epochs=200)
```

```
Epoch [1/200], Loss: 0.6823
Epoch [2/200], Loss: 0.5980
Epoch [3/200], Loss: 0.5039
Epoch [4/200], Loss: 0.4120
Epoch [5/200], Loss: 0.4884
Epoch [6/200], Loss: 0.4483
Epoch [7/200], Loss: 0.4019
Epoch [8/200], Loss: 0.3971
Epoch [9/200], Loss: 0.4069
Epoch [10/200], Loss: 0.4111
Epoch [11/200], Loss: 0.4062
Epoch [12/200], Loss: 0.3943
Epoch [13/200], Loss: 0.3804
Epoch [14/200], Loss: 0.3713
Epoch [15/200], Loss: 0.3715
Epoch [16/200], Loss: 0.3750
Epoch [17/200], Loss: 0.3713
Epoch [18/200], Loss: 0.3608
Epoch [19/200], Loss: 0.3515
Epoch [20/200], Loss: 0.3476
Epoch [21/200], Loss: 0.3470
Epoch [22/200], Loss: 0.3458
Epoch [23/200], Loss: 0.3423
Epoch [24/200], Loss: 0.3378
Epoch [25/200], Loss: 0.3354
Epoch [26/200], Loss: 0.3362
Epoch [27/200], Loss: 0.3375
Epoch [28/200], Loss: 0.3361
Epoch [29/200], Loss: 0.3334
Epoch [30/200], Loss: 0.3322
Epoch [31/200], Loss: 0.3326
Epoch [32/200], Loss: 0.3327
Epoch [33/200], Loss: 0.3313
```

```
Epoch [34/200], Loss: 0.3295
Epoch [35/200], Loss: 0.3292
Epoch [36/200], Loss: 0.3299
Epoch [37/200], Loss: 0.3292
Epoch [38/200], Loss: 0.3279
Epoch [39/200], Loss: 0.3278
Epoch [40/200], Loss: 0.3282
Epoch [41/200], Loss: 0.3277
Epoch [42/200], Loss: 0.3268
Epoch [43/200], Loss: 0.3265
Epoch [44/200], Loss: 0.3267
Epoch [45/200], Loss: 0.3265
Epoch [46/200], Loss: 0.3259
Epoch [47/200], Loss: 0.3256
Epoch [48/200], Loss: 0.3257
Epoch [49/200], Loss: 0.3256
Epoch [50/200], Loss: 0.3251
Epoch [51/200], Loss: 0.3249
Epoch [52/200], Loss: 0.3249
Epoch [53/200], Loss: 0.3248
Epoch [54/200], Loss: 0.3245
Epoch [55/200], Loss: 0.3244
Epoch [56/200], Loss: 0.3245
Epoch [57/200], Loss: 0.3244
Epoch [58/200], Loss: 0.3242
Epoch [59/200], Loss: 0.3242
Epoch [60/200], Loss: 0.3242
Epoch [61/200], Loss: 0.3240
Epoch [62/200], Loss: 0.3239
Epoch [63/200], Loss: 0.3239
Epoch [64/200], Loss: 0.3239
Epoch [65/200], Loss: 0.3237
Epoch [66/200], Loss: 0.3237
Epoch [67/200], Loss: 0.3237
Epoch [68/200], Loss: 0.3236
Epoch [69/200], Loss: 0.3235
Epoch [70/200], Loss: 0.3235
Epoch [71/200], Loss: 0.3234
Epoch [72/200], Loss: 0.3233
Epoch [73/200], Loss: 0.3233
Epoch [74/200], Loss: 0.3233
Epoch [75/200], Loss: 0.3232
Epoch [76/200], Loss: 0.3231
Epoch [77/200], Loss: 0.3231
Epoch [78/200], Loss: 0.3231
Epoch [79/200], Loss: 0.3230
Epoch [80/200], Loss: 0.3230
Epoch [81/200], Loss: 0.3229
```

```
Epoch [82/200], Loss: 0.3229
Epoch [83/200], Loss: 0.3228
Epoch [84/200], Loss: 0.3228
Epoch [85/200], Loss: 0.3228
Epoch [86/200], Loss: 0.3227
Epoch [87/200], Loss: 0.3227
Epoch [88/200], Loss: 0.3226
Epoch [89/200], Loss: 0.3226
Epoch [90/200], Loss: 0.3226
Epoch [91/200], Loss: 0.3225
Epoch [92/200], Loss: 0.3225
Epoch [93/200], Loss: 0.3225
Epoch [94/200], Loss: 0.3224
Epoch [95/200], Loss: 0.3224
Epoch [96/200], Loss: 0.3224
Epoch [97/200], Loss: 0.3223
Epoch [98/200], Loss: 0.3223
Epoch [99/200], Loss: 0.3223
Epoch [100/200], Loss: 0.3222
Epoch [101/200], Loss: 0.3222
Epoch [102/200], Loss: 0.3222
Epoch [103/200], Loss: 0.3221
Epoch [104/200], Loss: 0.3221
Epoch [105/200], Loss: 0.3220
Epoch [106/200], Loss: 0.3220
Epoch [107/200], Loss: 0.3220
Epoch [108/200], Loss: 0.3219
Epoch [109/200], Loss: 0.3219
Epoch [110/200], Loss: 0.3219
Epoch [111/200], Loss: 0.3218
Epoch [112/200], Loss: 0.3218
Epoch [113/200], Loss: 0.3218
Epoch [114/200], Loss: 0.3217
Epoch [115/200], Loss: 0.3217
Epoch [116/200], Loss: 0.3216
Epoch [117/200], Loss: 0.3216
Epoch [118/200], Loss: 0.3216
Epoch [119/200], Loss: 0.3215
Epoch [120/200], Loss: 0.3215
Epoch [121/200], Loss: 0.3214
Epoch [122/200], Loss: 0.3214
Epoch [123/200], Loss: 0.3214
Epoch [124/200], Loss: 0.3213
Epoch [125/200], Loss: 0.3213
Epoch [126/200], Loss: 0.3212
Epoch [127/200], Loss: 0.3212
Epoch [128/200], Loss: 0.3212
Epoch [129/200], Loss: 0.3211
```

```
Epoch [130/200], Loss: 0.3211
Epoch [131/200], Loss: 0.3211
Epoch [132/200], Loss: 0.3210
Epoch [133/200], Loss: 0.3210
Epoch [134/200], Loss: 0.3210
Epoch [135/200], Loss: 0.3209
Epoch [136/200], Loss: 0.3209
Epoch [137/200], Loss: 0.3209
Epoch [138/200], Loss: 0.3208
Epoch [139/200], Loss: 0.3208
Epoch [140/200], Loss: 0.3208
Epoch [141/200], Loss: 0.3207
Epoch [142/200], Loss: 0.3207
Epoch [143/200], Loss: 0.3207
Epoch [144/200], Loss: 0.3206
Epoch [145/200], Loss: 0.3206
Epoch [146/200], Loss: 0.3206
Epoch [147/200], Loss: 0.3206
Epoch [148/200], Loss: 0.3206
Epoch [149/200], Loss: 0.3205
Epoch [150/200], Loss: 0.3205
Epoch [151/200], Loss: 0.3205
Epoch [152/200], Loss: 0.3204
Epoch [153/200], Loss: 0.3204
Epoch [154/200], Loss: 0.3204
Epoch [155/200], Loss: 0.3203
Epoch [156/200], Loss: 0.3203
Epoch [157/200], Loss: 0.3203
Epoch [158/200], Loss: 0.3203
Epoch [159/200], Loss: 0.3202
Epoch [160/200], Loss: 0.3202
Epoch [161/200], Loss: 0.3202
Epoch [162/200], Loss: 0.3201
Epoch [163/200], Loss: 0.3201
Epoch [164/200], Loss: 0.3201
Epoch [165/200], Loss: 0.3201
Epoch [166/200], Loss: 0.3200
Epoch [167/200], Loss: 0.3200
Epoch [168/200], Loss: 0.3200
Epoch [169/200], Loss: 0.3200
Epoch [170/200], Loss: 0.3200
Epoch [171/200], Loss: 0.3199
Epoch [172/200], Loss: 0.3199
Epoch [173/200], Loss: 0.3199
Epoch [174/200], Loss: 0.3199
Epoch [175/200], Loss: 0.3199
Epoch [176/200], Loss: 0.3199
Epoch [177/200], Loss: 0.3199
```

```
Epoch [178/200], Loss: 0.3199
Epoch [179/200], Loss: 0.3199
Epoch [180/200], Loss: 0.3198
Epoch [181/200], Loss: 0.3197
Epoch [182/200], Loss: 0.3197
Epoch [183/200], Loss: 0.3197
Epoch [184/200], Loss: 0.3197
Epoch [185/200], Loss: 0.3197
Epoch [186/200], Loss: 0.3196
Epoch [187/200], Loss: 0.3196
Epoch [188/200], Loss: 0.3196
Epoch [189/200], Loss: 0.3196
Epoch [190/200], Loss: 0.3196
Epoch [191/200], Loss: 0.3195
Epoch [192/200], Loss: 0.3195
Epoch [193/200], Loss: 0.3194
Epoch [194/200], Loss: 0.3194
Epoch [195/200], Loss: 0.3195
Epoch [196/200], Loss: 0.3195
Epoch [197/200], Loss: 0.3195
Epoch [198/200], Loss: 0.3194
Epoch [199/200], Loss: 0.3194
Epoch [200/200], Loss: 0.3193
```

[37]:

```
# test cnn
acc, y_pred, y_true, y_pred_proba = cnn.test(X_test_tensor, y_test_tensor)
```

```
Test Accuracy: 86.58%
```

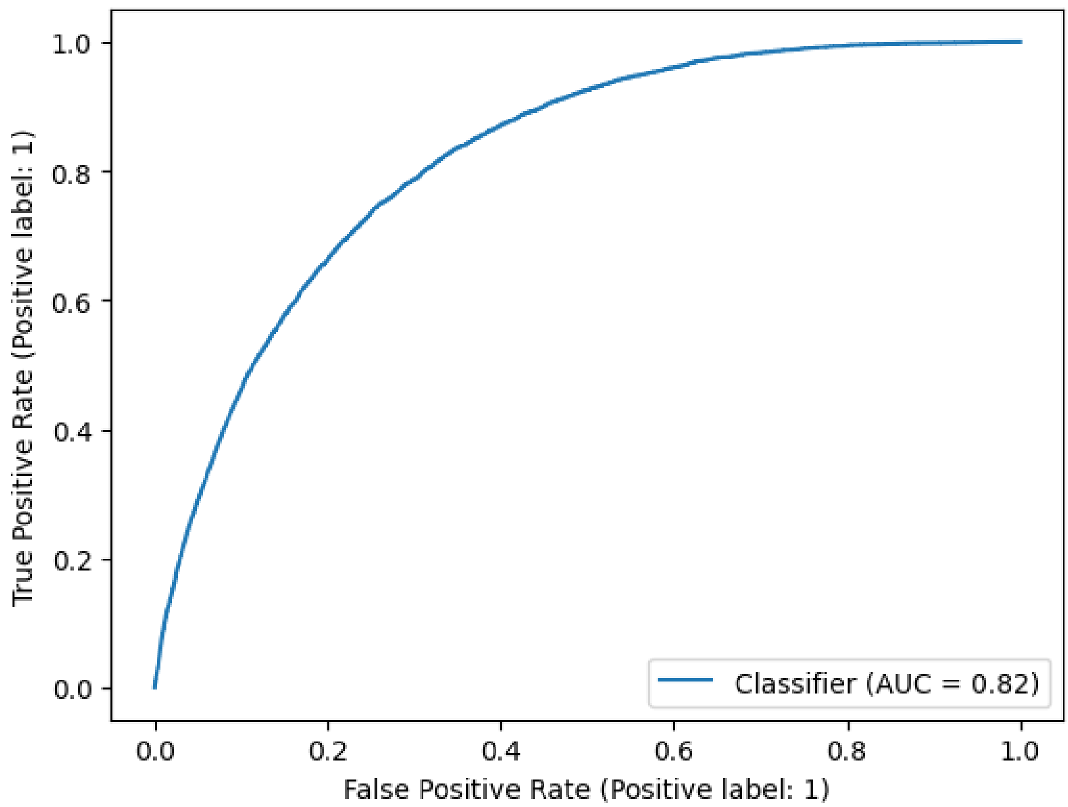
[38]:

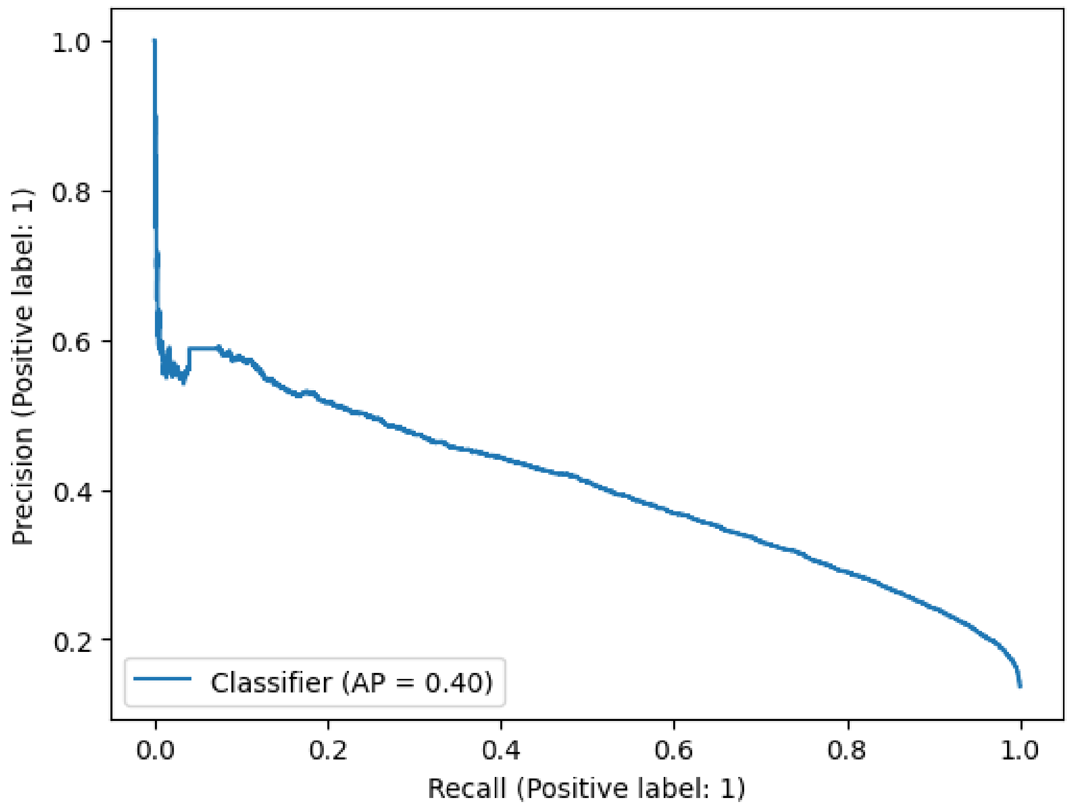
```
# print metrics
from sklearn.metrics import precision_score, recall_score, f1_score,␣
↪matthews_corrcoef, PrecisionRecallDisplay, RocCurveDisplay

# cow compute the metrics
cnn_precision = precision_score(y_true, y_pred)
cnn_recall = recall_score(y_true, y_pred)
cnn_f1 = f1_score(y_true, y_pred)
cnn_mcc = matthews_corrcoef(y_true, y_pred)

# print results
print(f'********** CNN Performance Metrics **********')
print(f'Precision: {cnn_precision:.4f}')
print(f'Recall: {cnn_recall:.4f}')
print(f'F1 Score: {cnn_f1:.4f}')
print(f'MCC: {cnn_mcc:.4f}')

# print AUC
# print AUROC and AUPRC
```

```
RocCurveDisplay.from_predictions(y_test_tensor, y_pred_proba)
PrecisionRecallDisplay.from_predictions(y_test_tensor, y_pred_proba)
```

```
********** CNN Performance Metrics **********
Precision: 0.5442
Recall: 0.1372
F1 Score: 0.2191
MCC: 0.2238
```

[38]:
```
<sklearn.metrics._plot.precision_recall_curve.PrecisionRecallDisplay at
0x2c652f1d0>
```





1.5 Question #4

Build and train a feedforward neural network with one hidden layer to predict BMI from the rest of the dataset. Use RMSE to assess the accuracy of your model. Does the RMSE depend on the activation function used?

[65]:
```
# we can set y = BMI and drop the diabetes (based on the fact we're using the␣
↪rest of the dataset)
# set our input matrix and target vectors
bmi_data = pd.read_csv('./diabetes.csv')
df = pd.DataFrame(data = bmi_data)

df.head()
```

[65]:

	Diabetes	HighBP	HighChol	BMI	Smoker	Stroke	Myocardial	PhysActivity	\
0	0	1	1	40	1	0	0	0	
1	0	0	0	25	1	0	0	1	
2	0	1	1	28	0	0	0	0	
3	0	1	0	27	0	0	0	1	
4	0	1	1	24	0	0	0	1	

```
   Fruit  Vegetables  ...  NotAbleToAffordDoctor  GeneralHealth  MentalHealth  \
0      0           1  ...                      0              5            18
1      0           0  ...                      1              3             0
2      1           0  ...                      1              5            30
3      1           1  ...                      0              2             0
4      1           1  ...                      0              2             3

   PhysicalHealth  HardToClimbStairs  BiologicalSex  AgeBracket  \
0              15                  1              1           9
1               0                  0              1           7
2              30                  1              1           9
3               0                  0              1          11
4               0                  0              1          11

   EducationBracket  IncomeBracket  Zodiac
0                 4              3      10
1                 6              1      11
2                 4              8       2
3                 3              6      11
4                 5              4       8

[5 rows x 22 columns]
```

[66]:
```python
df.drop(columns = ['Diabetes'])
```

[66]:
```
        HighBP  HighChol  BMI  Smoker  Stroke  Myocardial  PhysActivity  \
0            1         1   40       1       0           0             0
1            0         0   25       1       0           0             1
2            1         1   28       0       0           0             0
3            1         0   27       0       0           0             1
4            1         1   24       0       0           0             1
...        ...       ...  ...     ...     ...         ...           ...
253675       1         1   45       0       0           0             0
253676       1         1   18       0       0           0             0
253677       0         0   28       0       0           0             1
253678       1         0   23       0       0           0             0
253679       1         1   25       0       0           1             1

        Fruit  Vegetables  HeavyDrinker  ...  NotAbleToAffordDoctor  \
0           0           1             0  ...                      0
1           0           0             0  ...                      1
2           1           0             0  ...                      1
3           1           1             0  ...                      0
4           1           1             0  ...                      0
...       ...         ...           ...  ...                    ...
253675      1           1             0  ...                      0
253676      0           0             0  ...                      0
```

```
253677       1          0           0  ...                0
253678       1          1           0  ...                0
253679       1          0           0  ...                0

        GeneralHealth  MentalHealth  PhysicalHealth  HardToClimbStairs  \
0                   5            18              15                  1
1                   3             0               0                  0
2                   5            30              30                  1
3                   2             0               0                  0
4                   2             3               0                  0
...               ...           ...             ...                ...
253675              3             0               5                  0
253676              4             0               0                  1
253677              1             0               0                  0
253678              3             0               0                  0
253679              2             0               0                  0

        BiologicalSex  AgeBracket  EducationBracket  IncomeBracket  Zodiac
0                   1           9                 4              3      10
1                   1           7                 6              1      11
2                   1           9                 4              8       2
3                   1          11                 3              6      11
4                   1          11                 5              4       8
...               ...         ...               ...            ...     ...
253675              2           5                 6              7      11
253676              1          11                 2              4       6
253677              1           2                 5              2       5
253678              2           7                 5              1       6
253679              1           9                 6              2       6

[253680 rows x 21 columns]
```

[67]:
```python
y_bmi = df['BMI']
X_bmi = df.drop(columns = ['Diabetes', 'BMI'])

print(f'X: \n{X_bmi}')
print(f'y: \n{y_bmi}')
```

```
X:
        HighBP  HighChol  Smoker  Stroke  Myocardial  PhysActivity  Fruit  \
0            1         1       1       0           0             0      0
1            0         0       1       0           0             1      0
2            1         1       0       0           0             0      1
3            1         0       0       0           0             1      1
4            1         1       0       0           0             1      1
...        ...       ...     ...     ...         ...           ...    ...
253675       1         1       0       0           0             0      1
253676       1         1       0       0           0             0      0
```

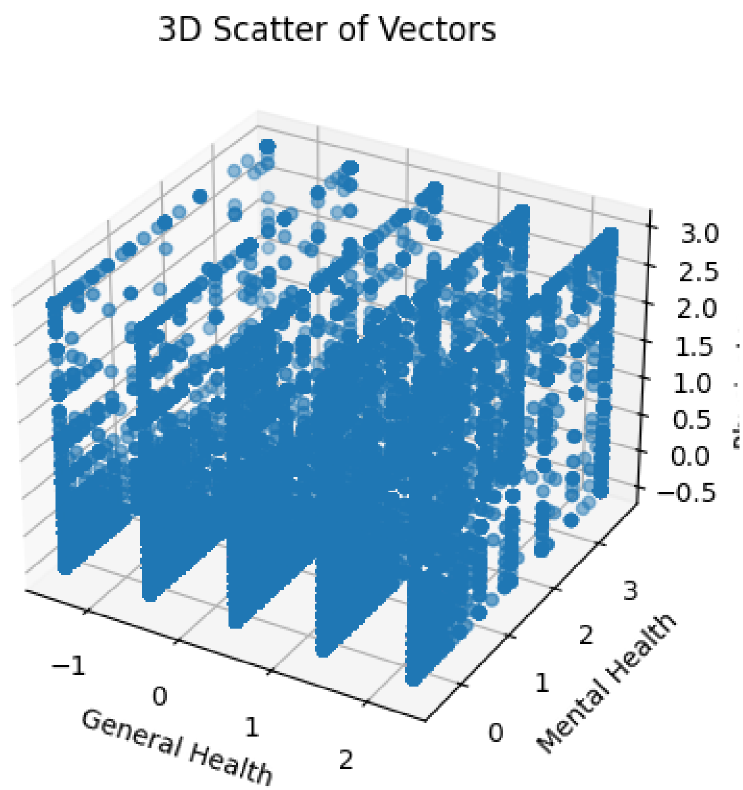
```
253677       0          0        0        0          0          1      1
253678       1          0        0        0          0          0      1
253679       1          1        0        0          1          1      1

        Vegetables  HeavyDrinker  HasHealthcare  NotAbleToAffordDoctor  \
0                1             0              1                      0
1                0             0              0                      1
2                0             0              1                      1
3                1             0              1                      0
4                1             0              1                      0
...            ...           ...            ...                    ...
253675           1             0              1                      0
253676           0             0              1                      0
253677           0             0              1                      0
253678           1             0              1                      0
253679           0             0              1                      0

        GeneralHealth  MentalHealth  PhysicalHealth  HardToClimbStairs  \
0                   5            18              15                  1
1                   3             0               0                  0
2                   5            30              30                  1
3                   2             0               0                  0
4                   2             3               0                  0
...               ...           ...             ...                ...
253675              3             0               5                  0
253676              4             0               0                  1
253677              1             0               0                  0
253678              3             0               0                  0
253679              2             0               0                  0

        BiologicalSex  AgeBracket  EducationBracket  IncomeBracket  Zodiac
0                   1           9                 4              3      10
1                   1           7                 6              1      11
2                   1           9                 4              8       2
3                   1          11                 3              6      11
4                   1          11                 5              4       8
...               ...         ...               ...            ...     ...
253675              2           5                 6              7      11
253676              1          11                 2              4       6
253677              1           2                 5              2       5
253678              2           7                 5              1       6
253679              1           9                 6              2       6

[253680 rows x 20 columns]
y:
0         40
1         25
2         28
```

```
3         27
4         24
          ..
253675    45
253676    18
253677    28
253678    23
253679    25
Name: BMI, Length: 253680, dtype: int64
```

[68]:
```
# let's preprocess some of our data, since we're working with a simpler MLP, we␣
↪don't need to perform PCA
## (working w/ less layers [no convolutional layer])

# get mean_centered data
X_bmi = (X_bmi - X_bmi.mean(axis=0)) / X_bmi.std(axis=0)

# features to scale - based on previous analysis
theta_between_gm = cosine_similarity(X_bmi['GeneralHealth'],␣
↪X_bmi['MentalHealth'])
theta_between_gp = cosine_similarity(X_bmi['GeneralHealth'],␣
↪X_bmi['PhysicalHealth'])
theta_between_mp = cosine_similarity(X_bmi['MentalHealth'],␣
↪X_bmi['PhysicalHealth'])

# output thetas
print(f'angle between general health & mental health {theta_between_gm}')
print(f'angle between general health & physical health {theta_between_gp}')
print(f'angle between mental health & physical health {theta_between_mp}')
```

```
angle between general health & mental health 1.2643479460331948
angle between general health & physical health 1.0188287261739337
angle between mental health & physical health 1.2093592038400125
```

[69]:
```
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

ax.scatter(X_bmi['GeneralHealth'], X_bmi['MentalHealth'],␣
↪X_bmi['PhysicalHealth'])

ax.set_xlabel('General Health')
ax.set_ylabel('Mental Health')
ax.set_zlabel('Physical Health')
ax.set_title("3D Scatter of Vectors")

plt.show()
```



We can see that all of angles are around 1, indicating there is collinearity between these vectors. The 3d projection also shows relative parallelism between the vectors.

We can combine these vectors using PCA or even simpler, maybe just even finding the average between the 3 vectors.

[70]:
```
# reduce to one vector and then add back to dataset
from sklearn.decomposition import PCA

X_collinear = X_bmi[['GeneralHealth', 'MentalHealth', 'PhysicalHealth']]

# perform pca
pca = PCA(n_components=1)
X_pca_collinear = pca.fit_transform(X_collinear)

# add it back
X_bmi['HealthPCA'] = X_pca_collinear

# drop columns since they're now combined with PCA
X_bmi_reduced = X_bmi.drop(columns=['GeneralHealth', 'MentalHealth',␣
↪'PhysicalHealth'])
print(X_bmi_reduced)
```

```
          HighBP  HighChol    Smoker    Stroke  Myocardial  PhysActivity  \
0       1.153686  1.165252  1.120925 -0.205636   -0.322457     -1.762810
1      -0.866784 -0.858180  1.120925 -0.205636   -0.322457      0.567274
2       1.153686  1.165252 -0.892117 -0.205636   -0.322457     -1.762810
3       1.153686 -0.858180 -0.892117 -0.205636   -0.322457      0.567274
4       1.153686  1.165252 -0.892117 -0.205636   -0.322457      0.567274
...          ...       ...       ...       ...         ...           ...
253675  1.153686  1.165252 -0.892117 -0.205636   -0.322457     -1.762810
253676  1.153686  1.165252 -0.892117 -0.205636   -0.322457     -1.762810
253677 -0.866784 -0.858180 -0.892117 -0.205636   -0.322457      0.567274
253678  1.153686 -0.858180 -0.892117 -0.205636   -0.322457     -1.762810
253679  1.153686  1.165252 -0.892117 -0.205636    3.101177      0.567274

           Fruit  Vegetables  HeavyDrinker  HasHealthcare  \
0      -1.316869    0.482086     -0.244014       0.226862
1      -1.316869   -2.074312     -0.244014      -4.407945
2       0.759374   -2.074312     -0.244014       0.226862
3       0.759374    0.482086     -0.244014       0.226862
4       0.759374    0.482086     -0.244014       0.226862
...          ...         ...           ...            ...
253675  0.759374    0.482086     -0.244014       0.226862
253676 -1.316869   -2.074312     -0.244014       0.226862
253677  0.759374   -2.074312     -0.244014       0.226862
253678  0.759374    0.482086     -0.244014       0.226862
253679  0.759374   -2.074312     -0.244014       0.226862

        NotAbleToAffordDoctor  HardToClimbStairs  BiologicalSex  AgeBracket  \
0                   -0.303173           2.223611      -0.887019    0.316899
1                    3.298439          -0.449717      -0.887019   -0.337932
2                    3.298439           2.223611      -0.887019    0.316899
3                   -0.303173          -0.449717      -0.887019    0.971731
4                   -0.303173          -0.449717      -0.887019    0.971731
...                       ...                ...            ...         ...
253675              -0.303173          -0.449717       1.127367   -0.992764
253676              -0.303173           2.223611      -0.887019    0.971731
253677              -0.303173          -0.449717      -0.887019   -1.975011
253678              -0.303173          -0.449717       1.127367   -0.337932
253679              -0.303173          -0.449717      -0.887019    0.316899

        EducationBracket  IncomeBracket    Zodiac  HealthPCA
0              -1.065593      -1.474484  1.012212   3.172271
1               0.963270      -2.440133  1.301778  -0.243819
2              -1.065593       0.939636 -1.304314   5.055967
3              -2.080024      -0.026012  1.301778  -0.806019
4              -0.051161      -0.991660  0.433081  -0.602021
...                  ...            ...       ...        ...
253675          0.963270       0.456812  1.301778   0.112082
253676         -3.094455      -0.991660 -0.146051   0.318380
```

```
253677          -0.051161      -1.957309 -0.435617  -1.368219
253678          -0.051161      -2.440133 -0.146051  -0.243819
253679           0.963270      -1.957309 -0.146051  -0.806019

[253680 rows x 18 columns]
```

[71]:
```
from sklearn.model_selection import train_test_split

# split data
X_bmi_train, X_bmi_test, y_bmi_train, y_bmi_test =␣
↪train_test_split(X_bmi_reduced, y_bmi, test_size = 0.2, random_state =␣
↪14173755)

# turn data into torch tensors for model
X_bmi_train_tensor = torch.as_tensor(X_bmi_train.values, dtype = torch.float32)
y_bmi_train_tensor = torch.as_tensor(y_bmi_train.values, dtype = torch.float32)

X_bmi_test_tensor = torch.as_tensor(X_bmi_test.values, dtype = torch.float32)
y_bmi_test_tensor = torch.as_tensor(y_bmi_test.values, dtype = torch.float32)
```

[72]:
```
# import multiple models
from randomized_nn_model import RandomizedFNN

rfnn = RandomizedFNN(X_bmi_train_tensor, y_bmi_train_tensor)
rfnn_batch_2 = RandomizedFNN(X_bmi_train_tensor, y_bmi_train_tensor)
rfnn_batch_3 = RandomizedFNN(X_bmi_train_tensor, y_bmi_train_tensor)
rfnn_batch_4 = RandomizedFNN(X_bmi_train_tensor, y_bmi_train_tensor)

# first batch
rfnn.X = X_bmi_train_tensor
rfnn.y = y_bmi_train_tensor

# second batch
rfnn_batch_2.X = X_bmi_train_tensor
rfnn_batch_2.y = y_bmi_train_tensor

# last batch
rfnn_batch_3.X = X_bmi_train_tensor
rfnn_batch_3.y = y_bmi_train_tensor

rfnn_batch_4.X = X_bmi_train_tensor
rfnn_batch_4.y = y_bmi_train_tensor
```

[73]:
```
# let's train multiple models with random activations and see how it performs␣
↪(with different num epochs as well)
# batch 1
rfnn._train(num_epochs=200)
```

```python
# batch 2
rfnn_batch_2._train(num_epochs=200)

# batch 3
rfnn_batch_3._train(num_epochs=100)

# batch 3
rfnn_batch_4._train(num_epochs=300)
```

```
Sequential(
  (0): Linear(in_features=18, out_features=36, bias=True)
  (1): ELU(alpha=1.0)
  (2): Linear(in_features=36, out_features=36, bias=True)
  (3): ReLU()
  (4): Linear(in_features=36, out_features=36, bias=True)
  (5): Sigmoid()
  (6): Linear(in_features=36, out_features=1, bias=True)
)
Epoch [1/200], Loss: 856.7877
Epoch [2/200], Loss: 563.5603
Epoch [3/200], Loss: 365.8166
Epoch [4/200], Loss: 228.0593
Epoch [5/200], Loss: 136.7225
Epoch [6/200], Loss: 81.5382
Epoch [7/200], Loss: 53.9516
Epoch [8/200], Loss: 44.1265
Epoch [9/200], Loss: 41.7293
Epoch [10/200], Loss: 41.0999
Epoch [11/200], Loss: 40.7693
Epoch [12/200], Loss: 40.5175
Epoch [13/200], Loss: 40.3115
Epoch [14/200], Loss: 40.1399
Epoch [15/200], Loss: 39.9957
Epoch [16/200], Loss: 39.8736
Epoch [17/200], Loss: 39.7694
Epoch [18/200], Loss: 39.6799
Epoch [19/200], Loss: 39.6027
Epoch [20/200], Loss: 39.5358
Epoch [21/200], Loss: 39.4775
Epoch [22/200], Loss: 39.4265
Epoch [23/200], Loss: 39.3817
Epoch [24/200], Loss: 39.3421
Epoch [25/200], Loss: 39.3071
Epoch [26/200], Loss: 39.2758
Epoch [27/200], Loss: 39.2479
Epoch [28/200], Loss: 39.2229
Epoch [29/200], Loss: 39.2004
```

```
Epoch [30/200], Loss: 39.1800
Epoch [31/200], Loss: 39.1615
Epoch [32/200], Loss: 39.1447
Epoch [33/200], Loss: 39.1294
Epoch [34/200], Loss: 39.1154
Epoch [35/200], Loss: 39.1024
Epoch [36/200], Loss: 39.0904
Epoch [37/200], Loss: 39.0792
Epoch [38/200], Loss: 39.0688
Epoch [39/200], Loss: 39.0591
Epoch [40/200], Loss: 39.0499
Epoch [41/200], Loss: 39.0413
Epoch [42/200], Loss: 39.0331
Epoch [43/200], Loss: 39.0253
Epoch [44/200], Loss: 39.0178
Epoch [45/200], Loss: 39.0107
Epoch [46/200], Loss: 39.0038
Epoch [47/200], Loss: 38.9972
Epoch [48/200], Loss: 38.9908
Epoch [49/200], Loss: 38.9845
Epoch [50/200], Loss: 38.9785
Epoch [51/200], Loss: 38.9726
Epoch [52/200], Loss: 38.9668
Epoch [53/200], Loss: 38.9611
Epoch [54/200], Loss: 38.9556
Epoch [55/200], Loss: 38.9501
Epoch [56/200], Loss: 38.9448
Epoch [57/200], Loss: 38.9395
Epoch [58/200], Loss: 38.9342
Epoch [59/200], Loss: 38.9290
Epoch [60/200], Loss: 38.9239
Epoch [61/200], Loss: 38.9188
Epoch [62/200], Loss: 38.9138
Epoch [63/200], Loss: 38.9087
Epoch [64/200], Loss: 38.9038
Epoch [65/200], Loss: 38.8988
Epoch [66/200], Loss: 38.8939
Epoch [67/200], Loss: 38.8890
Epoch [68/200], Loss: 38.8841
Epoch [69/200], Loss: 38.8792
Epoch [70/200], Loss: 38.8744
Epoch [71/200], Loss: 38.8695
Epoch [72/200], Loss: 38.8646
Epoch [73/200], Loss: 38.8597
Epoch [74/200], Loss: 38.8548
Epoch [75/200], Loss: 38.8500
Epoch [76/200], Loss: 38.8451
Epoch [77/200], Loss: 38.8402
```

```
Epoch [78/200], Loss: 38.8353
Epoch [79/200], Loss: 38.8304
Epoch [80/200], Loss: 38.8254
Epoch [81/200], Loss: 38.8205
Epoch [82/200], Loss: 38.8155
Epoch [83/200], Loss: 38.8105
Epoch [84/200], Loss: 38.8055
Epoch [85/200], Loss: 38.8004
Epoch [86/200], Loss: 38.7954
Epoch [87/200], Loss: 38.7902
Epoch [88/200], Loss: 38.7851
Epoch [89/200], Loss: 38.7800
Epoch [90/200], Loss: 38.7748
Epoch [91/200], Loss: 38.7696
Epoch [92/200], Loss: 38.7644
Epoch [93/200], Loss: 38.7592
Epoch [94/200], Loss: 38.7539
Epoch [95/200], Loss: 38.7486
Epoch [96/200], Loss: 38.7432
Epoch [97/200], Loss: 38.7378
Epoch [98/200], Loss: 38.7324
Epoch [99/200], Loss: 38.7270
Epoch [100/200], Loss: 38.7216
Epoch [101/200], Loss: 38.7161
Epoch [102/200], Loss: 38.7105
Epoch [103/200], Loss: 38.7050
Epoch [104/200], Loss: 38.6994
Epoch [105/200], Loss: 38.6937
Epoch [106/200], Loss: 38.6881
Epoch [107/200], Loss: 38.6824
Epoch [108/200], Loss: 38.6766
Epoch [109/200], Loss: 38.6708
Epoch [110/200], Loss: 38.6650
Epoch [111/200], Loss: 38.6592
Epoch [112/200], Loss: 38.6533
Epoch [113/200], Loss: 38.6474
Epoch [114/200], Loss: 38.6414
Epoch [115/200], Loss: 38.6354
Epoch [116/200], Loss: 38.6293
Epoch [117/200], Loss: 38.6232
Epoch [118/200], Loss: 38.6171
Epoch [119/200], Loss: 38.6109
Epoch [120/200], Loss: 38.6047
Epoch [121/200], Loss: 38.5984
Epoch [122/200], Loss: 38.5921
Epoch [123/200], Loss: 38.5858
Epoch [124/200], Loss: 38.5794
Epoch [125/200], Loss: 38.5730
```

```
Epoch [126/200], Loss: 38.5665
Epoch [127/200], Loss: 38.5600
Epoch [128/200], Loss: 38.5534
Epoch [129/200], Loss: 38.5468
Epoch [130/200], Loss: 38.5401
Epoch [131/200], Loss: 38.5334
Epoch [132/200], Loss: 38.5267
Epoch [133/200], Loss: 38.5199
Epoch [134/200], Loss: 38.5131
Epoch [135/200], Loss: 38.5062
Epoch [136/200], Loss: 38.4993
Epoch [137/200], Loss: 38.4923
Epoch [138/200], Loss: 38.4853
Epoch [139/200], Loss: 38.4783
Epoch [140/200], Loss: 38.4712
Epoch [141/200], Loss: 38.4641
Epoch [142/200], Loss: 38.4570
Epoch [143/200], Loss: 38.4499
Epoch [144/200], Loss: 38.4427
Epoch [145/200], Loss: 38.4355
Epoch [146/200], Loss: 38.4282
Epoch [147/200], Loss: 38.4209
Epoch [148/200], Loss: 38.4135
Epoch [149/200], Loss: 38.4062
Epoch [150/200], Loss: 38.3987
Epoch [151/200], Loss: 38.3913
Epoch [152/200], Loss: 38.3838
Epoch [153/200], Loss: 38.3763
Epoch [154/200], Loss: 38.3688
Epoch [155/200], Loss: 38.3612
Epoch [156/200], Loss: 38.3537
Epoch [157/200], Loss: 38.3461
Epoch [158/200], Loss: 38.3384
Epoch [159/200], Loss: 38.3308
Epoch [160/200], Loss: 38.3232
Epoch [161/200], Loss: 38.3155
Epoch [162/200], Loss: 38.3078
Epoch [163/200], Loss: 38.3001
Epoch [164/200], Loss: 38.2924
Epoch [165/200], Loss: 38.2847
Epoch [166/200], Loss: 38.2770
Epoch [167/200], Loss: 38.2693
Epoch [168/200], Loss: 38.2616
Epoch [169/200], Loss: 38.2539
Epoch [170/200], Loss: 38.2462
Epoch [171/200], Loss: 38.2385
Epoch [172/200], Loss: 38.2308
Epoch [173/200], Loss: 38.2231
```

```
Epoch [174/200], Loss: 38.2154
Epoch [175/200], Loss: 38.2078
Epoch [176/200], Loss: 38.2001
Epoch [177/200], Loss: 38.1925
Epoch [178/200], Loss: 38.1848
Epoch [179/200], Loss: 38.1772
Epoch [180/200], Loss: 38.1696
Epoch [181/200], Loss: 38.1620
Epoch [182/200], Loss: 38.1544
Epoch [183/200], Loss: 38.1469
Epoch [184/200], Loss: 38.1393
Epoch [185/200], Loss: 38.1318
Epoch [186/200], Loss: 38.1242
Epoch [187/200], Loss: 38.1168
Epoch [188/200], Loss: 38.1093
Epoch [189/200], Loss: 38.1018
Epoch [190/200], Loss: 38.0944
Epoch [191/200], Loss: 38.0871
Epoch [192/200], Loss: 38.0797
Epoch [193/200], Loss: 38.0724
Epoch [194/200], Loss: 38.0652
Epoch [195/200], Loss: 38.0580
Epoch [196/200], Loss: 38.0508
Epoch [197/200], Loss: 38.0436
Epoch [198/200], Loss: 38.0365
Epoch [199/200], Loss: 38.0294
Epoch [200/200], Loss: 38.0223
Sequential(
  (0): Linear(in_features=18, out_features=36, bias=True)
  (1): Tanh()
  (2): Linear(in_features=36, out_features=36, bias=True)
  (3): LeakyReLU(negative_slope=0.01)
  (4): Linear(in_features=36, out_features=36, bias=True)
  (5): ReLU()
  (6): Linear(in_features=36, out_features=1, bias=True)
)
Epoch [1/200], Loss: 849.8004
Epoch [2/200], Loss: 808.3117
Epoch [3/200], Loss: 759.9259
Epoch [4/200], Loss: 672.2692
Epoch [5/200], Loss: 424.0261
Epoch [6/200], Loss: 378.9770
Epoch [7/200], Loss: 733.4362
Epoch [8/200], Loss: 700.2355
Epoch [9/200], Loss: 671.1141
Epoch [10/200], Loss: 632.1178
Epoch [11/200], Loss: 524.9423
Epoch [12/200], Loss: 127.6902
```

```
Epoch [13/200], Loss: 352.9392
Epoch [14/200], Loss: 644.2606
Epoch [15/200], Loss: 589.7434
Epoch [16/200], Loss: 562.7805
Epoch [17/200], Loss: 529.5585
Epoch [18/200], Loss: 475.1107
Epoch [19/200], Loss: 354.3597
Epoch [20/200], Loss: 107.2528
Epoch [21/200], Loss: 156.5554
Epoch [22/200], Loss: 409.9025
Epoch [23/200], Loss: 285.0545
Epoch [24/200], Loss: 102.3724
Epoch [25/200], Loss: 61.5521
Epoch [26/200], Loss: 92.8785
Epoch [27/200], Loss: 58.8495
Epoch [28/200], Loss: 82.7620
Epoch [29/200], Loss: 67.1210
Epoch [30/200], Loss: 107.8868
Epoch [31/200], Loss: 67.6752
Epoch [32/200], Loss: 70.6531
Epoch [33/200], Loss: 51.0526
Epoch [34/200], Loss: 53.4067
Epoch [35/200], Loss: 46.4908
Epoch [36/200], Loss: 48.8668
Epoch [37/200], Loss: 44.5960
Epoch [38/200], Loss: 46.0451
Epoch [39/200], Loss: 43.4241
Epoch [40/200], Loss: 44.3571
Epoch [41/200], Loss: 42.5178
Epoch [42/200], Loss: 43.0915
Epoch [43/200], Loss: 41.8055
Epoch [44/200], Loss: 42.1596
Epoch [45/200], Loss: 41.2437
Epoch [46/200], Loss: 41.4615
Epoch [47/200], Loss: 40.7970
Epoch [48/200], Loss: 40.9294
Epoch [49/200], Loss: 40.4388
Epoch [50/200], Loss: 40.5173
Epoch [51/200], Loss: 40.1488
Epoch [52/200], Loss: 40.1935
Epoch [53/200], Loss: 39.9108
Epoch [54/200], Loss: 39.9343
Epoch [55/200], Loss: 39.7135
Epoch [56/200], Loss: 39.7233
Epoch [57/200], Loss: 39.5474
Epoch [58/200], Loss: 39.5487
Epoch [59/200], Loss: 39.4063
Epoch [60/200], Loss: 39.4025
```

```
Epoch [61/200], Loss: 39.2855
Epoch [62/200], Loss: 39.2789
Epoch [63/200], Loss: 39.1808
Epoch [64/200], Loss: 39.1722
Epoch [65/200], Loss: 39.0890
Epoch [66/200], Loss: 39.0797
Epoch [67/200], Loss: 39.0079
Epoch [68/200], Loss: 38.9983
Epoch [69/200], Loss: 38.9356
Epoch [70/200], Loss: 38.9260
Epoch [71/200], Loss: 38.8706
Epoch [72/200], Loss: 38.8612
Epoch [73/200], Loss: 38.8116
Epoch [74/200], Loss: 38.8029
Epoch [75/200], Loss: 38.7580
Epoch [76/200], Loss: 38.7499
Epoch [77/200], Loss: 38.7088
Epoch [78/200], Loss: 38.7013
Epoch [79/200], Loss: 38.6636
Epoch [80/200], Loss: 38.6571
Epoch [81/200], Loss: 38.6220
Epoch [82/200], Loss: 38.6165
Epoch [83/200], Loss: 38.5837
Epoch [84/200], Loss: 38.5795
Epoch [85/200], Loss: 38.5487
Epoch [86/200], Loss: 38.5458
Epoch [87/200], Loss: 38.5167
Epoch [88/200], Loss: 38.5151
Epoch [89/200], Loss: 38.4875
Epoch [90/200], Loss: 38.4875
Epoch [91/200], Loss: 38.4609
Epoch [92/200], Loss: 38.4625
Epoch [93/200], Loss: 38.4368
Epoch [94/200], Loss: 38.4403
Epoch [95/200], Loss: 38.4155
Epoch [96/200], Loss: 38.4213
Epoch [97/200], Loss: 38.3970
Epoch [98/200], Loss: 38.4052
Epoch [99/200], Loss: 38.3811
Epoch [100/200], Loss: 38.3921
Epoch [101/200], Loss: 38.3680
Epoch [102/200], Loss: 38.3823
Epoch [103/200], Loss: 38.3581
Epoch [104/200], Loss: 38.3761
Epoch [105/200], Loss: 38.3514
Epoch [106/200], Loss: 38.3736
Epoch [107/200], Loss: 38.3480
Epoch [108/200], Loss: 38.3750
```

```
Epoch [109/200], Loss: 38.3481
Epoch [110/200], Loss: 38.3808
Epoch [111/200], Loss: 38.3521
Epoch [112/200], Loss: 38.3914
Epoch [113/200], Loss: 38.3601
Epoch [114/200], Loss: 38.4067
Epoch [115/200], Loss: 38.3723
Epoch [116/200], Loss: 38.4273
Epoch [117/200], Loss: 38.3887
Epoch [118/200], Loss: 38.4534
Epoch [119/200], Loss: 38.4090
Epoch [120/200], Loss: 38.4837
Epoch [121/200], Loss: 38.4327
Epoch [122/200], Loss: 38.5186
Epoch [123/200], Loss: 38.4591
Epoch [124/200], Loss: 38.5565
Epoch [125/200], Loss: 38.4873
Epoch [126/200], Loss: 38.5964
Epoch [127/200], Loss: 38.5158
Epoch [128/200], Loss: 38.6363
Epoch [129/200], Loss: 38.5429
Epoch [130/200], Loss: 38.6738
Epoch [131/200], Loss: 38.5669
Epoch [132/200], Loss: 38.7066
Epoch [133/200], Loss: 38.5861
Epoch [134/200], Loss: 38.7324
Epoch [135/200], Loss: 38.5992
Epoch [136/200], Loss: 38.7497
Epoch [137/200], Loss: 38.6052
Epoch [138/200], Loss: 38.7574
Epoch [139/200], Loss: 38.6032
Epoch [140/200], Loss: 38.7543
Epoch [141/200], Loss: 38.5930
Epoch [142/200], Loss: 38.7403
Epoch [143/200], Loss: 38.5747
Epoch [144/200], Loss: 38.7160
Epoch [145/200], Loss: 38.5491
Epoch [146/200], Loss: 38.6831
Epoch [147/200], Loss: 38.5178
Epoch [148/200], Loss: 38.6436
Epoch [149/200], Loss: 38.4823
Epoch [150/200], Loss: 38.5990
Epoch [151/200], Loss: 38.4434
Epoch [152/200], Loss: 38.5506
Epoch [153/200], Loss: 38.4022
Epoch [154/200], Loss: 38.5002
Epoch [155/200], Loss: 38.3601
Epoch [156/200], Loss: 38.4496
```

```
Epoch [157/200], Loss: 38.3182
Epoch [158/200], Loss: 38.3997
Epoch [159/200], Loss: 38.2772
Epoch [160/200], Loss: 38.3516
Epoch [161/200], Loss: 38.2378
Epoch [162/200], Loss: 38.3059
Epoch [163/200], Loss: 38.2002
Epoch [164/200], Loss: 38.2627
Epoch [165/200], Loss: 38.1648
Epoch [166/200], Loss: 38.2226
Epoch [167/200], Loss: 38.1317
Epoch [168/200], Loss: 38.1854
Epoch [169/200], Loss: 38.1008
Epoch [170/200], Loss: 38.1509
Epoch [171/200], Loss: 38.0723
Epoch [172/200], Loss: 38.1194
Epoch [173/200], Loss: 38.0458
Epoch [174/200], Loss: 38.0907
Epoch [175/200], Loss: 38.0219
Epoch [176/200], Loss: 38.0649
Epoch [177/200], Loss: 38.0001
Epoch [178/200], Loss: 38.0416
Epoch [179/200], Loss: 37.9804
Epoch [180/200], Loss: 38.0210
Epoch [181/200], Loss: 37.9628
Epoch [182/200], Loss: 38.0026
Epoch [183/200], Loss: 37.9469
Epoch [184/200], Loss: 37.9862
Epoch [185/200], Loss: 37.9327
Epoch [186/200], Loss: 37.9719
Epoch [187/200], Loss: 37.9203
Epoch [188/200], Loss: 37.9594
Epoch [189/200], Loss: 37.9094
Epoch [190/200], Loss: 37.9488
Epoch [191/200], Loss: 37.9001
Epoch [192/200], Loss: 37.9400
Epoch [193/200], Loss: 37.8923
Epoch [194/200], Loss: 37.9329
Epoch [195/200], Loss: 37.8858
Epoch [196/200], Loss: 37.9274
Epoch [197/200], Loss: 37.8806
Epoch [198/200], Loss: 37.9232
Epoch [199/200], Loss: 37.8764
Epoch [200/200], Loss: 37.9202
Sequential(
  (0): Linear(in_features=18, out_features=36, bias=True)
  (1): ELU(alpha=1.0)
  (2): Linear(in_features=36, out_features=36, bias=True)
```

```
  (3): Tanh()
  (4): Linear(in_features=36, out_features=36, bias=True)
  (5): ReLU()
  (6): Linear(in_features=36, out_features=1, bias=True)
)
Epoch [1/100], Loss: 847.1845
Epoch [2/100], Loss: 787.1933
Epoch [3/100], Loss: 681.8000
Epoch [4/100], Loss: 369.1623
Epoch [5/100], Loss: 372.8838
Epoch [6/100], Loss: 783.7192
Epoch [7/100], Loss: 727.4870
Epoch [8/100], Loss: 688.1129
Epoch [9/100], Loss: 622.1990
Epoch [10/100], Loss: 462.9252
Epoch [11/100], Loss: 89.5213
Epoch [12/100], Loss: 264.2161
Epoch [13/100], Loss: 635.0772
Epoch [14/100], Loss: 575.4816
Epoch [15/100], Loss: 469.8862
Epoch [16/100], Loss: 244.9409
Epoch [17/100], Loss: 70.9936
Epoch [18/100], Loss: 156.8613
Epoch [19/100], Loss: 99.2395
Epoch [20/100], Loss: 224.7433
Epoch [21/100], Loss: 46.3513
Epoch [22/100], Loss: 52.8439
Epoch [23/100], Loss: 60.6273
Epoch [24/100], Loss: 97.5259
Epoch [25/100], Loss: 75.2612
Epoch [26/100], Loss: 129.8054
Epoch [27/100], Loss: 58.8425
Epoch [28/100], Loss: 81.1535
Epoch [29/100], Loss: 62.2878
Epoch [30/100], Loss: 86.6281
Epoch [31/100], Loss: 59.0610
Epoch [32/100], Loss: 76.8144
Epoch [33/100], Loss: 56.3356
Epoch [34/100], Loss: 69.4289
Epoch [35/100], Loss: 53.8079
Epoch [36/100], Loss: 63.3405
Epoch [37/100], Loss: 51.5288
Epoch [38/100], Loss: 58.3972
Epoch [39/100], Loss: 49.5125
Epoch [40/100], Loss: 54.4209
Epoch [41/100], Loss: 47.7607
Epoch [42/100], Loss: 51.2488
Epoch [43/100], Loss: 46.2616
```

```
Epoch [44/100], Loss: 48.7346
Epoch [45/100], Loss: 44.9966
Epoch [46/100], Loss: 46.7553
Epoch [47/100], Loss: 43.9426
Epoch [48/100], Loss: 45.2020
Epoch [49/100], Loss: 43.0722
Epoch [50/100], Loss: 43.9857
Epoch [51/100], Loss: 42.3598
Epoch [52/100], Loss: 43.0358
Epoch [53/100], Loss: 41.7811
Epoch [54/100], Loss: 42.2953
Epoch [55/100], Loss: 41.3145
Epoch [56/100], Loss: 41.7186
Epoch [57/100], Loss: 40.9406
Epoch [58/100], Loss: 41.2710
Epoch [59/100], Loss: 40.6434
Epoch [60/100], Loss: 40.9249
Epoch [61/100], Loss: 40.4088
Epoch [62/100], Loss: 40.6588
Epoch [63/100], Loss: 40.2251
Epoch [64/100], Loss: 40.4553
Epoch [65/100], Loss: 40.0826
Epoch [66/100], Loss: 40.3010
Epoch [67/100], Loss: 39.9726
Epoch [68/100], Loss: 40.1843
Epoch [69/100], Loss: 39.8879
Epoch [70/100], Loss: 40.0963
Epoch [71/100], Loss: 39.8223
Epoch [72/100], Loss: 40.0295
Epoch [73/100], Loss: 39.7701
Epoch [74/100], Loss: 39.9771
Epoch [75/100], Loss: 39.7267
Epoch [76/100], Loss: 39.9337
Epoch [77/100], Loss: 39.6885
Epoch [78/100], Loss: 39.8953
Epoch [79/100], Loss: 39.6524
Epoch [80/100], Loss: 39.8585
Epoch [81/100], Loss: 39.6164
Epoch [82/100], Loss: 39.8213
Epoch [83/100], Loss: 39.5793
Epoch [84/100], Loss: 39.7823
Epoch [85/100], Loss: 39.5405
Epoch [86/100], Loss: 39.7411
Epoch [87/100], Loss: 39.4996
Epoch [88/100], Loss: 39.6969
Epoch [89/100], Loss: 39.4566
Epoch [90/100], Loss: 39.6503
Epoch [91/100], Loss: 39.4117
```

```
Epoch [92/100], Loss: 39.6014
Epoch [93/100], Loss: 39.3653
Epoch [94/100], Loss: 39.5507
Epoch [95/100], Loss: 39.3178
Epoch [96/100], Loss: 39.4984
Epoch [97/100], Loss: 39.2697
Epoch [98/100], Loss: 39.4454
Epoch [99/100], Loss: 39.2212
Epoch [100/100], Loss: 39.3918
Sequential(
  (0): Linear(in_features=18, out_features=36, bias=True)
  (1): Tanh()
  (2): Linear(in_features=36, out_features=36, bias=True)
  (3): ELU(alpha=1.0)
  (4): Linear(in_features=36, out_features=36, bias=True)
  (5): LeakyReLU(negative_slope=0.01)
  (6): Linear(in_features=36, out_features=1, bias=True)
)
Epoch [1/300], Loss: 850.9338
Epoch [2/300], Loss: 806.1163
Epoch [3/300], Loss: 749.5605
Epoch [4/300], Loss: 644.7297
Epoch [5/300], Loss: 370.4981
Epoch [6/300], Loss: 241.6129
Epoch [7/300], Loss: 687.8868
Epoch [8/300], Loss: 549.5732
Epoch [9/300], Loss: 258.3534
Epoch [10/300], Loss: 81.3282
Epoch [11/300], Loss: 216.5457
Epoch [12/300], Loss: 79.3687
Epoch [13/300], Loss: 138.4482
Epoch [14/300], Loss: 92.5178
Epoch [15/300], Loss: 184.8363
Epoch [16/300], Loss: 53.7691
Epoch [17/300], Loss: 68.9354
Epoch [18/300], Loss: 62.8729
Epoch [19/300], Loss: 87.9748
Epoch [20/300], Loss: 63.5942
Epoch [21/300], Loss: 85.8493
Epoch [22/300], Loss: 59.8743
Epoch [23/300], Loss: 75.3173
Epoch [24/300], Loss: 56.3856
Epoch [25/300], Loss: 66.9483
Epoch [26/300], Loss: 53.3429
Epoch [27/300], Loss: 60.5328
Epoch [28/300], Loss: 50.7518
Epoch [29/300], Loss: 55.6335
Epoch [30/300], Loss: 48.5824
```

```
Epoch [31/300], Loss: 51.8968
Epoch [32/300], Loss: 46.7894
Epoch [33/300], Loss: 49.0456
Epoch [34/300], Loss: 45.3226
Epoch [35/300], Loss: 46.8682
Epoch [36/300], Loss: 44.1325
Epoch [37/300], Loss: 45.2031
Epoch [38/300], Loss: 43.1731
Epoch [39/300], Loss: 43.9282
Epoch [40/300], Loss: 42.4046
Epoch [41/300], Loss: 42.9515
Epoch [42/300], Loss: 41.7930
Epoch [43/300], Loss: 42.2044
Epoch [44/300], Loss: 41.3103
Epoch [45/300], Loss: 41.6357
Epoch [46/300], Loss: 40.9343
Epoch [47/300], Loss: 41.2076
Epoch [48/300], Loss: 40.6478
Epoch [49/300], Loss: 40.8926
Epoch [50/300], Loss: 40.4373
Epoch [51/300], Loss: 40.6711
Epoch [52/300], Loss: 40.2931
Epoch [53/300], Loss: 40.5287
Epoch [54/300], Loss: 40.2070
Epoch [55/300], Loss: 40.4544
Epoch [56/300], Loss: 40.1725
Epoch [57/300], Loss: 40.4392
Epoch [58/300], Loss: 40.1827
Epoch [59/300], Loss: 40.4739
Epoch [60/300], Loss: 40.2294
Epoch [61/300], Loss: 40.5479
Epoch [62/300], Loss: 40.3021
Epoch [63/300], Loss: 40.6478
Epoch [64/300], Loss: 40.3881
Epoch [65/300], Loss: 40.7580
Epoch [66/300], Loss: 40.4736
Epoch [67/300], Loss: 40.8617
Epoch [68/300], Loss: 40.5454
Epoch [69/300], Loss: 40.9440
Epoch [70/300], Loss: 40.5935
Epoch [71/300], Loss: 40.9939
Epoch [72/300], Loss: 40.6124
Epoch [73/300], Loss: 41.0067
Epoch [74/300], Loss: 40.6012
Epoch [75/300], Loss: 40.9827
Epoch [76/300], Loss: 40.5627
Epoch [77/300], Loss: 40.9264
Epoch [78/300], Loss: 40.5018
```

```
Epoch [79/300], Loss: 40.8448
Epoch [80/300], Loss: 40.4244
Epoch [81/300], Loss: 40.7456
Epoch [82/300], Loss: 40.3362
Epoch [83/300], Loss: 40.6355
Epoch [84/300], Loss: 40.2420
Epoch [85/300], Loss: 40.5203
Epoch [86/300], Loss: 40.1458
Epoch [87/300], Loss: 40.4045
Epoch [88/300], Loss: 40.0503
Epoch [89/300], Loss: 40.2910
Epoch [90/300], Loss: 39.9575
Epoch [91/300], Loss: 40.1818
Epoch [92/300], Loss: 39.8682
Epoch [93/300], Loss: 40.0776
Epoch [94/300], Loss: 39.7832
Epoch [95/300], Loss: 39.9791
Epoch [96/300], Loss: 39.7025
Epoch [97/300], Loss: 39.8863
Epoch [98/300], Loss: 39.6261
Epoch [99/300], Loss: 39.7987
Epoch [100/300], Loss: 39.5540
Epoch [101/300], Loss: 39.7163
Epoch [102/300], Loss: 39.4855
Epoch [103/300], Loss: 39.6384
Epoch [104/300], Loss: 39.4204
Epoch [105/300], Loss: 39.5647
Epoch [106/300], Loss: 39.3584
Epoch [107/300], Loss: 39.4946
Epoch [108/300], Loss: 39.2993
Epoch [109/300], Loss: 39.4279
Epoch [110/300], Loss: 39.2427
Epoch [111/300], Loss: 39.3642
Epoch [112/300], Loss: 39.1884
Epoch [113/300], Loss: 39.3033
Epoch [114/300], Loss: 39.1362
Epoch [115/300], Loss: 39.2450
Epoch [116/300], Loss: 39.0861
Epoch [117/300], Loss: 39.1890
Epoch [118/300], Loss: 39.0378
Epoch [119/300], Loss: 39.1353
Epoch [120/300], Loss: 38.9914
Epoch [121/300], Loss: 39.0839
Epoch [122/300], Loss: 38.9467
Epoch [123/300], Loss: 39.0346
Epoch [124/300], Loss: 38.9038
Epoch [125/300], Loss: 38.9872
Epoch [126/300], Loss: 38.8625
```

```
Epoch [127/300], Loss: 38.9419
Epoch [128/300], Loss: 38.8228
Epoch [129/300], Loss: 38.8984
Epoch [130/300], Loss: 38.7847
Epoch [131/300], Loss: 38.8567
Epoch [132/300], Loss: 38.7481
Epoch [133/300], Loss: 38.8168
Epoch [134/300], Loss: 38.7129
Epoch [135/300], Loss: 38.7785
Epoch [136/300], Loss: 38.6790
Epoch [137/300], Loss: 38.7417
Epoch [138/300], Loss: 38.6465
Epoch [139/300], Loss: 38.7065
Epoch [140/300], Loss: 38.6152
Epoch [141/300], Loss: 38.6727
Epoch [142/300], Loss: 38.5851
Epoch [143/300], Loss: 38.6403
Epoch [144/300], Loss: 38.5563
Epoch [145/300], Loss: 38.6092
Epoch [146/300], Loss: 38.5285
Epoch [147/300], Loss: 38.5795
Epoch [148/300], Loss: 38.5019
Epoch [149/300], Loss: 38.5510
Epoch [150/300], Loss: 38.4764
Epoch [151/300], Loss: 38.5238
Epoch [152/300], Loss: 38.4519
Epoch [153/300], Loss: 38.4977
Epoch [154/300], Loss: 38.4284
Epoch [155/300], Loss: 38.4726
Epoch [156/300], Loss: 38.4058
Epoch [157/300], Loss: 38.4486
Epoch [158/300], Loss: 38.3840
Epoch [159/300], Loss: 38.4255
Epoch [160/300], Loss: 38.3631
Epoch [161/300], Loss: 38.4033
Epoch [162/300], Loss: 38.3429
Epoch [163/300], Loss: 38.3820
Epoch [164/300], Loss: 38.3236
Epoch [165/300], Loss: 38.3617
Epoch [166/300], Loss: 38.3051
Epoch [167/300], Loss: 38.3422
Epoch [168/300], Loss: 38.2873
Epoch [169/300], Loss: 38.3235
Epoch [170/300], Loss: 38.2702
Epoch [171/300], Loss: 38.3056
Epoch [172/300], Loss: 38.2538
Epoch [173/300], Loss: 38.2884
Epoch [174/300], Loss: 38.2380
```

```
Epoch [175/300], Loss: 38.2720
Epoch [176/300], Loss: 38.2228
Epoch [177/300], Loss: 38.2561
Epoch [178/300], Loss: 38.2082
Epoch [179/300], Loss: 38.2409
Epoch [180/300], Loss: 38.1941
Epoch [181/300], Loss: 38.2262
Epoch [182/300], Loss: 38.1806
Epoch [183/300], Loss: 38.2123
Epoch [184/300], Loss: 38.1677
Epoch [185/300], Loss: 38.1990
Epoch [186/300], Loss: 38.1553
Epoch [187/300], Loss: 38.1862
Epoch [188/300], Loss: 38.1433
Epoch [189/300], Loss: 38.1738
Epoch [190/300], Loss: 38.1318
Epoch [191/300], Loss: 38.1620
Epoch [192/300], Loss: 38.1208
Epoch [193/300], Loss: 38.1507
Epoch [194/300], Loss: 38.1102
Epoch [195/300], Loss: 38.1399
Epoch [196/300], Loss: 38.1000
Epoch [197/300], Loss: 38.1295
Epoch [198/300], Loss: 38.0903
Epoch [199/300], Loss: 38.1196
Epoch [200/300], Loss: 38.0808
Epoch [201/300], Loss: 38.1100
Epoch [202/300], Loss: 38.0718
Epoch [203/300], Loss: 38.1009
Epoch [204/300], Loss: 38.0631
Epoch [205/300], Loss: 38.0921
Epoch [206/300], Loss: 38.0547
Epoch [207/300], Loss: 38.0836
Epoch [208/300], Loss: 38.0466
Epoch [209/300], Loss: 38.0755
Epoch [210/300], Loss: 38.0388
Epoch [211/300], Loss: 38.0677
Epoch [212/300], Loss: 38.0314
Epoch [213/300], Loss: 38.0602
Epoch [214/300], Loss: 38.0242
Epoch [215/300], Loss: 38.0529
Epoch [216/300], Loss: 38.0171
Epoch [217/300], Loss: 38.0458
Epoch [218/300], Loss: 38.0103
Epoch [219/300], Loss: 38.0390
Epoch [220/300], Loss: 38.0037
Epoch [221/300], Loss: 38.0324
Epoch [222/300], Loss: 37.9972
```

```
Epoch [223/300], Loss: 38.0260
Epoch [224/300], Loss: 37.9910
Epoch [225/300], Loss: 38.0198
Epoch [226/300], Loss: 37.9850
Epoch [227/300], Loss: 38.0138
Epoch [228/300], Loss: 37.9791
Epoch [229/300], Loss: 38.0080
Epoch [230/300], Loss: 37.9734
Epoch [231/300], Loss: 38.0024
Epoch [232/300], Loss: 37.9679
Epoch [233/300], Loss: 37.9969
Epoch [234/300], Loss: 37.9623
Epoch [235/300], Loss: 37.9914
Epoch [236/300], Loss: 37.9570
Epoch [237/300], Loss: 37.9861
Epoch [238/300], Loss: 37.9517
Epoch [239/300], Loss: 37.9810
Epoch [240/300], Loss: 37.9466
Epoch [241/300], Loss: 37.9760
Epoch [242/300], Loss: 37.9416
Epoch [243/300], Loss: 37.9710
Epoch [244/300], Loss: 37.9367
Epoch [245/300], Loss: 37.9661
Epoch [246/300], Loss: 37.9318
Epoch [247/300], Loss: 37.9614
Epoch [248/300], Loss: 37.9271
Epoch [249/300], Loss: 37.9567
Epoch [250/300], Loss: 37.9224
Epoch [251/300], Loss: 37.9522
Epoch [252/300], Loss: 37.9178
Epoch [253/300], Loss: 37.9477
Epoch [254/300], Loss: 37.9134
Epoch [255/300], Loss: 37.9433
Epoch [256/300], Loss: 37.9089
Epoch [257/300], Loss: 37.9390
Epoch [258/300], Loss: 37.9045
Epoch [259/300], Loss: 37.9346
Epoch [260/300], Loss: 37.9002
Epoch [261/300], Loss: 37.9304
Epoch [262/300], Loss: 37.8959
Epoch [263/300], Loss: 37.9261
Epoch [264/300], Loss: 37.8916
Epoch [265/300], Loss: 37.9218
Epoch [266/300], Loss: 37.8873
Epoch [267/300], Loss: 37.9176
Epoch [268/300], Loss: 37.8830
Epoch [269/300], Loss: 37.9134
Epoch [270/300], Loss: 37.8787
```

```
Epoch [271/300], Loss: 37.9092
Epoch [272/300], Loss: 37.8745
Epoch [273/300], Loss: 37.9050
Epoch [274/300], Loss: 37.8703
Epoch [275/300], Loss: 37.9009
Epoch [276/300], Loss: 37.8661
Epoch [277/300], Loss: 37.8967
Epoch [278/300], Loss: 37.8619
Epoch [279/300], Loss: 37.8925
Epoch [280/300], Loss: 37.8577
Epoch [281/300], Loss: 37.8884
Epoch [282/300], Loss: 37.8536
Epoch [283/300], Loss: 37.8843
Epoch [284/300], Loss: 37.8495
Epoch [285/300], Loss: 37.8803
Epoch [286/300], Loss: 37.8455
Epoch [287/300], Loss: 37.8762
Epoch [288/300], Loss: 37.8414
Epoch [289/300], Loss: 37.8722
Epoch [290/300], Loss: 37.8374
Epoch [291/300], Loss: 37.8682
Epoch [292/300], Loss: 37.8334
Epoch [293/300], Loss: 37.8643
Epoch [294/300], Loss: 37.8294
Epoch [295/300], Loss: 37.8603
Epoch [296/300], Loss: 37.8255
Epoch [297/300], Loss: 37.8565
Epoch [298/300], Loss: 37.8217
Epoch [299/300], Loss: 37.8526
Epoch [300/300], Loss: 37.8178
```

[74]:
```
rfnn.test(X_bmi_test_tensor, y_bmi_test_tensor)
rfnn_batch_2.test(X_bmi_test_tensor, y_bmi_test_tensor)
rfnn_batch_3.test(X_bmi_test_tensor, y_bmi_test_tensor)
rfnn_batch_4.test(X_bmi_test_tensor, y_bmi_test_tensor)
```

```
Test Loss: 37.525242
Test Loss: 37.555630
Test Loss: 38.769650
Test Loss: 37.311306
```

1.6 Question #5

Build and train a neural network of your choice to predict BMI from the rest of your dataset. How low can you get RMSE and what design choices does RMSE seem to depend on?

[75]:
```

```

```python
# we'll implement a deeper FNN to train on BMI, but also one with standard relu,␣
↪and one with random activations.
# we'll also play around with the training epochs and see how our RMSE differs␣
↪based on # epochs
from randomized_nn_model import RandomizedFNN
from nn_model import FNN

# 3 hidden layers
fnn_clf_3 = FNN(X_bmi_train_tensor, y_bmi_train_tensor)
rfnn_clf_3 = RandomizedFNN(X_bmi_train_tensor, y_bmi_train_tensor)

fnn_clf_3.X = X_bmi_train_tensor
fnn_clf_3.y = y_bmi_train_tensor

rfnn_clf_3.X = X_bmi_train_tensor
rfnn_clf_3.y = y_bmi_train_tensor

# 4 hidden layers
fnn_clf_4 = FNN(X_bmi_train_tensor, y_bmi_train_tensor, numHiddenLayers = 4)
rfnn_clf_4 = RandomizedFNN(X_bmi_train_tensor, y_bmi_train_tensor,␣
↪numHiddenLayers = 4)

fnn_clf_4.X = X_bmi_train_tensor
fnn_clf_4.y = y_bmi_train_tensor

rfnn_clf_4.X = X_bmi_train_tensor
rfnn_clf_4.y = y_bmi_train_tensor
```

[76]:

```python
# 200 epochs for "standard" network
fnn_clf_3._train()
rfnn_clf_3._train()

# we'll reduce the size of the epochs on the network with more layers
fnn_clf_4._train(num_epochs = 100)
rfnn_clf_4._train(num_epochs = 100)
```

```
Epoch [1/200], Loss: 856.1902
Epoch [2/200], Loss: 787.5620
Epoch [3/200], Loss: 672.0135
Epoch [4/200], Loss: 309.3618
Epoch [5/200], Loss: 2254.8882
Epoch [6/200], Loss: 1836.9645
Epoch [7/200], Loss: 751.0082
Epoch [8/200], Loss: 722.3359
Epoch [9/200], Loss: 265.0567
Epoch [10/200], Loss: 689.3422
Epoch [11/200], Loss: 663.6973
```

```
Epoch [12/200], Loss: 639.0469
Epoch [13/200], Loss: 615.3286
Epoch [14/200], Loss: 592.4561
Epoch [15/200], Loss: 570.2018
Epoch [16/200], Loss: 536.1680
Epoch [17/200], Loss: 10918587.0000
Epoch [18/200], Loss: 5636.9185
Epoch [19/200], Loss: 5414.8672
Epoch [20/200], Loss: 5201.9541
Epoch [21/200], Loss: 4997.3369
Epoch [22/200], Loss: 4800.3618
Epoch [23/200], Loss: 4610.1055
Epoch [24/200], Loss: 4424.9199
Epoch [25/200], Loss: 4241.5786
Epoch [26/200], Loss: 4053.5449
Epoch [27/200], Loss: 3847.8240
Epoch [28/200], Loss: 3600.5686
Epoch [29/200], Loss: 3276.4890
Epoch [30/200], Loss: 2852.4885
Epoch [31/200], Loss: 2394.1035
Epoch [32/200], Loss: 2083.7786
Epoch [33/200], Loss: 1973.1885
Epoch [34/200], Loss: 1924.5416
Epoch [35/200], Loss: 1880.7390
Epoch [36/200], Loss: 1838.0129
Epoch [37/200], Loss: 1796.2849
Epoch [38/200], Loss: 1755.5310
Epoch [39/200], Loss: 1715.7288
Epoch [40/200], Loss: 1676.8558
Epoch [41/200], Loss: 1638.8899
Epoch [42/200], Loss: 1601.8108
Epoch [43/200], Loss: 1565.5968
Epoch [44/200], Loss: 1530.2280
Epoch [45/200], Loss: 1495.6849
Epoch [46/200], Loss: 1461.9480
Epoch [47/200], Loss: 1428.9979
Epoch [48/200], Loss: 1396.8171
Epoch [49/200], Loss: 1365.3871
Epoch [50/200], Loss: 1334.6904
Epoch [51/200], Loss: 1304.7098
Epoch [52/200], Loss: 1275.4287
Epoch [53/200], Loss: 1246.8307
Epoch [54/200], Loss: 1218.8997
Epoch [55/200], Loss: 1191.6201
Epoch [56/200], Loss: 1164.9768
Epoch [57/200], Loss: 1138.9550
Epoch [58/200], Loss: 1113.5398
Epoch [59/200], Loss: 1088.7173
```

```
Epoch [60/200], Loss: 1064.4736
Epoch [61/200], Loss: 1040.7953
Epoch [62/200], Loss: 1017.6687
Epoch [63/200], Loss: 995.0814
Epoch [64/200], Loss: 973.0208
Epoch [65/200], Loss: 951.4741
Epoch [66/200], Loss: 930.4298
Epoch [67/200], Loss: 909.8759
Epoch [68/200], Loss: 889.8008
Epoch [69/200], Loss: 870.1935
Epoch [70/200], Loss: 851.0432
Epoch [71/200], Loss: 832.3390
Epoch [72/200], Loss: 814.0705
Epoch [73/200], Loss: 796.2277
Epoch [74/200], Loss: 778.8003
Epoch [75/200], Loss: 761.7789
Epoch [76/200], Loss: 745.1537
Epoch [77/200], Loss: 728.9159
Epoch [78/200], Loss: 713.0560
Epoch [79/200], Loss: 697.5655
Epoch [80/200], Loss: 682.4356
Epoch [81/200], Loss: 667.6579
Epoch [82/200], Loss: 653.2241
Epoch [83/200], Loss: 639.1263
Epoch [84/200], Loss: 625.3566
Epoch [85/200], Loss: 611.9073
Epoch [86/200], Loss: 598.7709
Epoch [87/200], Loss: 585.9402
Epoch [88/200], Loss: 573.4079
Epoch [89/200], Loss: 561.1672
Epoch [90/200], Loss: 549.2112
Epoch [91/200], Loss: 537.5332
Epoch [92/200], Loss: 526.1268
Epoch [93/200], Loss: 514.9857
Epoch [94/200], Loss: 504.1036
Epoch [95/200], Loss: 493.4744
Epoch [96/200], Loss: 483.0923
Epoch [97/200], Loss: 472.9515
Epoch [98/200], Loss: 463.0464
Epoch [99/200], Loss: 453.3715
Epoch [100/200], Loss: 443.9214
Epoch [101/200], Loss: 434.6909
Epoch [102/200], Loss: 425.6747
Epoch [103/200], Loss: 416.8680
Epoch [104/200], Loss: 408.2658
Epoch [105/200], Loss: 399.8634
Epoch [106/200], Loss: 391.6561
Epoch [107/200], Loss: 383.6393
```

```
Epoch [108/200], Loss: 375.8086
Epoch [109/200], Loss: 368.1597
Epoch [110/200], Loss: 360.6884
Epoch [111/200], Loss: 353.3903
Epoch [112/200], Loss: 346.2617
Epoch [113/200], Loss: 339.2984
Epoch [114/200], Loss: 332.4967
Epoch [115/200], Loss: 325.8527
Epoch [116/200], Loss: 319.3627
Epoch [117/200], Loss: 313.0234
Epoch [118/200], Loss: 306.8309
Epoch [119/200], Loss: 300.7820
Epoch [120/200], Loss: 294.8734
Epoch [121/200], Loss: 289.1016
Epoch [122/200], Loss: 283.4636
Epoch [123/200], Loss: 277.9562
Epoch [124/200], Loss: 272.5764
Epoch [125/200], Loss: 267.3213
Epoch [126/200], Loss: 262.1878
Epoch [127/200], Loss: 257.1732
Epoch [128/200], Loss: 252.2747
Epoch [129/200], Loss: 247.4897
Epoch [130/200], Loss: 242.8154
Epoch [131/200], Loss: 238.2492
Epoch [132/200], Loss: 233.7887
Epoch [133/200], Loss: 229.4314
Epoch [134/200], Loss: 225.1749
Epoch [135/200], Loss: 221.0168
Epoch [136/200], Loss: 216.9549
Epoch [137/200], Loss: 212.9869
Epoch [138/200], Loss: 209.1106
Epoch [139/200], Loss: 205.3239
Epoch [140/200], Loss: 201.6248
Epoch [141/200], Loss: 198.0111
Epoch [142/200], Loss: 194.4808
Epoch [143/200], Loss: 191.0322
Epoch [144/200], Loss: 187.6631
Epoch [145/200], Loss: 184.3719
Epoch [146/200], Loss: 181.1566
Epoch [147/200], Loss: 178.0155
Epoch [148/200], Loss: 174.9470
Epoch [149/200], Loss: 171.9492
Epoch [150/200], Loss: 169.0206
Epoch [151/200], Loss: 166.1596
Epoch [152/200], Loss: 163.3645
Epoch [153/200], Loss: 160.6338
Epoch [154/200], Loss: 157.9661
Epoch [155/200], Loss: 155.3599
```

```
Epoch [156/200], Loss: 152.8138
Epoch [157/200], Loss: 150.3263
Epoch [158/200], Loss: 147.8960
Epoch [159/200], Loss: 145.5218
Epoch [160/200], Loss: 143.2022
Epoch [161/200], Loss: 140.9360
Epoch [162/200], Loss: 138.7220
Epoch [163/200], Loss: 136.5589
Epoch [164/200], Loss: 134.4456
Epoch [165/200], Loss: 132.3809
Epoch [166/200], Loss: 130.3636
Epoch [167/200], Loss: 128.3928
Epoch [168/200], Loss: 126.4672
Epoch [169/200], Loss: 124.5859
Epoch [170/200], Loss: 122.7478
Epoch [171/200], Loss: 120.9520
Epoch [172/200], Loss: 119.1974
Epoch [173/200], Loss: 117.4830
Epoch [174/200], Loss: 115.8081
Epoch [175/200], Loss: 114.1715
Epoch [176/200], Loss: 112.5725
Epoch [177/200], Loss: 111.0102
Epoch [178/200], Loss: 109.4837
Epoch [179/200], Loss: 107.9922
Epoch [180/200], Loss: 106.5349
Epoch [181/200], Loss: 105.1110
Epoch [182/200], Loss: 103.7197
Epoch [183/200], Loss: 102.3603
Epoch [184/200], Loss: 101.0320
Epoch [185/200], Loss: 99.7341
Epoch [186/200], Loss: 98.4660
Epoch [187/200], Loss: 97.2268
Epoch [188/200], Loss: 96.0160
Epoch [189/200], Loss: 94.8329
Epoch [190/200], Loss: 93.6768
Epoch [191/200], Loss: 92.5472
Epoch [192/200], Loss: 91.4433
Epoch [193/200], Loss: 90.3647
Epoch [194/200], Loss: 89.3108
Epoch [195/200], Loss: 88.2809
Epoch [196/200], Loss: 87.2745
Epoch [197/200], Loss: 86.2910
Epoch [198/200], Loss: 85.3300
Epoch [199/200], Loss: 84.3910
Epoch [200/200], Loss: 83.4733
Sequential(
  (0): Linear(in_features=18, out_features=36, bias=True)
  (1): ReLU()
```

```
  (2): Linear(in_features=36, out_features=36, bias=True)
  (3): Tanh()
  (4): Linear(in_features=36, out_features=36, bias=True)
  (5): LeakyReLU(negative_slope=0.01)
  (6): Linear(in_features=36, out_features=1, bias=True)
)
Epoch [1/200], Loss: 858.6627
Epoch [2/200], Loss: 807.2255
Epoch [3/200], Loss: 734.9708
Epoch [4/200], Loss: 525.8394
Epoch [5/200], Loss: 53.1754
Epoch [6/200], Loss: 128.2678
Epoch [7/200], Loss: 830.7034
Epoch [8/200], Loss: 3588.0461
Epoch [9/200], Loss: 1321.1201
Epoch [10/200], Loss: 692.9349
Epoch [11/200], Loss: 483.2754
Epoch [12/200], Loss: 320.4944
Epoch [13/200], Loss: 201.5908
Epoch [14/200], Loss: 450.1253
Epoch [15/200], Loss: 415.9409
Epoch [16/200], Loss: 383.8792
Epoch [17/200], Loss: 353.6895
Epoch [18/200], Loss: 326.5491
Epoch [19/200], Loss: 301.5635
Epoch [20/200], Loss: 278.4698
Epoch [21/200], Loss: 257.1361
Epoch [22/200], Loss: 237.4541
Epoch [23/200], Loss: 219.3240
Epoch [24/200], Loss: 202.6521
Epoch [25/200], Loss: 187.3490
Epoch [26/200], Loss: 173.3280
Epoch [27/200], Loss: 160.4885
Epoch [28/200], Loss: 146.3965
Epoch [29/200], Loss: 97.7333
Epoch [30/200], Loss: 46.0681
Epoch [31/200], Loss: 46.0912
Epoch [32/200], Loss: 46.9628
Epoch [33/200], Loss: 46.8328
Epoch [34/200], Loss: 48.5276
Epoch [35/200], Loss: 47.7764
Epoch [36/200], Loss: 50.5004
Epoch [37/200], Loss: 48.5035
Epoch [38/200], Loss: 52.0524
Epoch [39/200], Loss: 48.7183
Epoch [40/200], Loss: 52.5790
Epoch [41/200], Loss: 48.5589
Epoch [42/200], Loss: 52.3585
```

```
Epoch [43/200], Loss: 48.2659
Epoch [44/200], Loss: 51.8748
Epoch [45/200], Loss: 47.9418
Epoch [46/200], Loss: 51.3330
Epoch [47/200], Loss: 47.6254
Epoch [48/200], Loss: 50.8072
Epoch [49/200], Loss: 47.3256
Epoch [50/200], Loss: 50.3139
Epoch [51/200], Loss: 47.0431
Epoch [52/200], Loss: 49.8531
Epoch [53/200], Loss: 46.7781
Epoch [54/200], Loss: 49.4246
Epoch [55/200], Loss: 46.5302
Epoch [56/200], Loss: 49.0265
Epoch [57/200], Loss: 46.2974
Epoch [58/200], Loss: 48.6539
Epoch [59/200], Loss: 46.0785
Epoch [60/200], Loss: 48.3066
Epoch [61/200], Loss: 45.8727
Epoch [62/200], Loss: 47.9783
Epoch [63/200], Loss: 45.6791
Epoch [64/200], Loss: 47.6713
Epoch [65/200], Loss: 45.4965
Epoch [66/200], Loss: 47.3827
Epoch [67/200], Loss: 45.3232
Epoch [68/200], Loss: 47.1095
Epoch [69/200], Loss: 45.1583
Epoch [70/200], Loss: 46.8516
Epoch [71/200], Loss: 45.0011
Epoch [72/200], Loss: 46.6095
Epoch [73/200], Loss: 44.8513
Epoch [74/200], Loss: 46.3886
Epoch [75/200], Loss: 44.7109
Epoch [76/200], Loss: 46.2034
Epoch [77/200], Loss: 44.5869
Epoch [78/200], Loss: 46.0819
Epoch [79/200], Loss: 44.4720
Epoch [80/200], Loss: 45.9728
Epoch [81/200], Loss: 44.3273
Epoch [82/200], Loss: 45.7368
Epoch [83/200], Loss: 44.1577
Epoch [84/200], Loss: 45.4332
Epoch [85/200], Loss: 43.9820
Epoch [86/200], Loss: 45.1281
Epoch [87/200], Loss: 43.8137
Epoch [88/200], Loss: 44.8440
Epoch [89/200], Loss: 43.6565
Epoch [90/200], Loss: 44.5851
```

```
Epoch [91/200], Loss: 43.5115
Epoch [92/200], Loss: 44.3511
Epoch [93/200], Loss: 43.3779
Epoch [94/200], Loss: 44.1397
Epoch [95/200], Loss: 43.2551
Epoch [96/200], Loss: 43.9488
Epoch [97/200], Loss: 43.1416
Epoch [98/200], Loss: 43.7752
Epoch [99/200], Loss: 43.0370
Epoch [100/200], Loss: 43.6179
Epoch [101/200], Loss: 42.9401
Epoch [102/200], Loss: 43.4745
Epoch [103/200], Loss: 42.8503
Epoch [104/200], Loss: 43.3434
Epoch [105/200], Loss: 42.7671
Epoch [106/200], Loss: 43.2237
Epoch [107/200], Loss: 42.6899
Epoch [108/200], Loss: 43.1146
Epoch [109/200], Loss: 42.6179
Epoch [110/200], Loss: 43.0140
Epoch [111/200], Loss: 42.5511
Epoch [112/200], Loss: 42.9218
Epoch [113/200], Loss: 42.4888
Epoch [114/200], Loss: 42.8370
Epoch [115/200], Loss: 42.4307
Epoch [116/200], Loss: 42.7590
Epoch [117/200], Loss: 42.3765
Epoch [118/200], Loss: 42.6871
Epoch [119/200], Loss: 42.3259
Epoch [120/200], Loss: 42.6208
Epoch [121/200], Loss: 42.2788
Epoch [122/200], Loss: 42.5598
Epoch [123/200], Loss: 42.2348
Epoch [124/200], Loss: 42.5036
Epoch [125/200], Loss: 42.1938
Epoch [126/200], Loss: 42.4519
Epoch [127/200], Loss: 42.1555
Epoch [128/200], Loss: 42.4041
Epoch [129/200], Loss: 42.1196
Epoch [130/200], Loss: 42.3596
Epoch [131/200], Loss: 42.0858
Epoch [132/200], Loss: 42.3188
Epoch [133/200], Loss: 42.0544
Epoch [134/200], Loss: 42.2808
Epoch [135/200], Loss: 42.0247
Epoch [136/200], Loss: 42.2458
Epoch [137/200], Loss: 41.9969
Epoch [138/200], Loss: 42.2134
```

```
Epoch [139/200], Loss: 41.9709
Epoch [140/200], Loss: 42.1835
Epoch [141/200], Loss: 41.9464
Epoch [142/200], Loss: 42.1556
Epoch [143/200], Loss: 41.9233
Epoch [144/200], Loss: 42.1298
Epoch [145/200], Loss: 41.9013
Epoch [146/200], Loss: 42.1055
Epoch [147/200], Loss: 41.8803
Epoch [148/200], Loss: 42.0828
Epoch [149/200], Loss: 41.8605
Epoch [150/200], Loss: 42.0617
Epoch [151/200], Loss: 41.8416
Epoch [152/200], Loss: 42.0418
Epoch [153/200], Loss: 41.8235
Epoch [154/200], Loss: 42.0232
Epoch [155/200], Loss: 41.8063
Epoch [156/200], Loss: 42.0057
Epoch [157/200], Loss: 41.7896
Epoch [158/200], Loss: 41.9891
Epoch [159/200], Loss: 41.7735
Epoch [160/200], Loss: 41.9734
Epoch [161/200], Loss: 41.7580
Epoch [162/200], Loss: 41.9586
Epoch [163/200], Loss: 41.7431
Epoch [164/200], Loss: 41.9445
Epoch [165/200], Loss: 41.7286
Epoch [166/200], Loss: 41.9310
Epoch [167/200], Loss: 41.7145
Epoch [168/200], Loss: 41.9181
Epoch [169/200], Loss: 41.7004
Epoch [170/200], Loss: 41.9052
Epoch [171/200], Loss: 41.6865
Epoch [172/200], Loss: 41.8927
Epoch [173/200], Loss: 41.6727
Epoch [174/200], Loss: 41.8805
Epoch [175/200], Loss: 41.6590
Epoch [176/200], Loss: 41.8684
Epoch [177/200], Loss: 41.6453
Epoch [178/200], Loss: 41.8562
Epoch [179/200], Loss: 41.6315
Epoch [180/200], Loss: 41.8441
Epoch [181/200], Loss: 41.6177
Epoch [182/200], Loss: 41.8319
Epoch [183/200], Loss: 41.6036
Epoch [184/200], Loss: 41.8194
Epoch [185/200], Loss: 41.5893
Epoch [186/200], Loss: 41.8068
```

```
Epoch [187/200], Loss: 41.5749
Epoch [188/200], Loss: 41.7939
Epoch [189/200], Loss: 41.5602
Epoch [190/200], Loss: 41.7808
Epoch [191/200], Loss: 41.5452
Epoch [192/200], Loss: 41.7674
Epoch [193/200], Loss: 41.5301
Epoch [194/200], Loss: 41.7537
Epoch [195/200], Loss: 41.5145
Epoch [196/200], Loss: 41.7395
Epoch [197/200], Loss: 41.4987
Epoch [198/200], Loss: 41.7250
Epoch [199/200], Loss: 41.4825
Epoch [200/200], Loss: 41.7103
Epoch [1/100], Loss: 852.9808
Epoch [2/100], Loss: 809.9378
Epoch [3/100], Loss: 755.5535
Epoch [4/100], Loss: 643.6285
Epoch [5/100], Loss: 235.8228
Epoch [6/100], Loss: 12315.1768
Epoch [7/100], Loss: 37178832.0000
Epoch [8/100], Loss: inf
Epoch [9/100], Loss: nan
Epoch [10/100], Loss: nan
Epoch [11/100], Loss: nan
Epoch [12/100], Loss: nan
Epoch [13/100], Loss: nan
Epoch [14/100], Loss: nan
Epoch [15/100], Loss: nan
Epoch [16/100], Loss: nan
Epoch [17/100], Loss: nan
Epoch [18/100], Loss: nan
Epoch [19/100], Loss: nan
Epoch [20/100], Loss: nan
Epoch [21/100], Loss: nan
Epoch [22/100], Loss: nan
Epoch [23/100], Loss: nan
Epoch [24/100], Loss: nan
Epoch [25/100], Loss: nan
Epoch [26/100], Loss: nan
Epoch [27/100], Loss: nan
Epoch [28/100], Loss: nan
Epoch [29/100], Loss: nan
Epoch [30/100], Loss: nan
Epoch [31/100], Loss: nan
Epoch [32/100], Loss: nan
Epoch [33/100], Loss: nan
Epoch [34/100], Loss: nan
```

```
Epoch [35/100], Loss: nan
Epoch [36/100], Loss: nan
Epoch [37/100], Loss: nan
Epoch [38/100], Loss: nan
Epoch [39/100], Loss: nan
Epoch [40/100], Loss: nan
Epoch [41/100], Loss: nan
Epoch [42/100], Loss: nan
Epoch [43/100], Loss: nan
Epoch [44/100], Loss: nan
Epoch [45/100], Loss: nan
Epoch [46/100], Loss: nan
Epoch [47/100], Loss: nan
Epoch [48/100], Loss: nan
Epoch [49/100], Loss: nan
Epoch [50/100], Loss: nan
Epoch [51/100], Loss: nan
Epoch [52/100], Loss: nan
Epoch [53/100], Loss: nan
Epoch [54/100], Loss: nan
Epoch [55/100], Loss: nan
Epoch [56/100], Loss: nan
Epoch [57/100], Loss: nan
Epoch [58/100], Loss: nan
Epoch [59/100], Loss: nan
Epoch [60/100], Loss: nan
Epoch [61/100], Loss: nan
Epoch [62/100], Loss: nan
Epoch [63/100], Loss: nan
Epoch [64/100], Loss: nan
Epoch [65/100], Loss: nan
Epoch [66/100], Loss: nan
Epoch [67/100], Loss: nan
Epoch [68/100], Loss: nan
Epoch [69/100], Loss: nan
Epoch [70/100], Loss: nan
Epoch [71/100], Loss: nan
Epoch [72/100], Loss: nan
Epoch [73/100], Loss: nan
Epoch [74/100], Loss: nan
Epoch [75/100], Loss: nan
Epoch [76/100], Loss: nan
Epoch [77/100], Loss: nan
Epoch [78/100], Loss: nan
Epoch [79/100], Loss: nan
Epoch [80/100], Loss: nan
Epoch [81/100], Loss: nan
Epoch [82/100], Loss: nan
```

```
Epoch [83/100], Loss: nan
Epoch [84/100], Loss: nan
Epoch [85/100], Loss: nan
Epoch [86/100], Loss: nan
Epoch [87/100], Loss: nan
Epoch [88/100], Loss: nan
Epoch [89/100], Loss: nan
Epoch [90/100], Loss: nan
Epoch [91/100], Loss: nan
Epoch [92/100], Loss: nan
Epoch [93/100], Loss: nan
Epoch [94/100], Loss: nan
Epoch [95/100], Loss: nan
Epoch [96/100], Loss: nan
Epoch [97/100], Loss: nan
Epoch [98/100], Loss: nan
Epoch [99/100], Loss: nan
Epoch [100/100], Loss: nan
Sequential(
  (0): Linear(in_features=18, out_features=36, bias=True)
  (1): Tanh()
  (2): Linear(in_features=36, out_features=36, bias=True)
  (3): LeakyReLU(negative_slope=0.01)
  (4): Linear(in_features=36, out_features=36, bias=True)
  (5): ELU(alpha=1.0)
  (6): Linear(in_features=36, out_features=36, bias=True)
  (7): Sigmoid()
  (8): Linear(in_features=36, out_features=1, bias=True)
)
Epoch [1/100], Loss: 847.5454
Epoch [2/100], Loss: 552.4092
Epoch [3/100], Loss: 354.4751
Epoch [4/100], Loss: 216.2875
Epoch [5/100], Loss: 123.6450
Epoch [6/100], Loss: 69.9073
Epoch [7/100], Loss: 48.3959
Epoch [8/100], Loss: 43.7257
Epoch [9/100], Loss: 42.7794
Epoch [10/100], Loss: 42.2472
Epoch [11/100], Loss: 41.8337
Epoch [12/100], Loss: 41.5012
Epoch [13/100], Loss: 41.2309
Epoch [14/100], Loss: 41.0077
Epoch [15/100], Loss: 40.8177
Epoch [16/100], Loss: 40.6517
Epoch [17/100], Loss: 40.5040
Epoch [18/100], Loss: 40.3706
Epoch [19/100], Loss: 40.2491
```

```
Epoch [20/100], Loss: 40.1376
Epoch [21/100], Loss: 40.0348
Epoch [22/100], Loss: 39.9398
Epoch [23/100], Loss: 39.8519
Epoch [24/100], Loss: 39.7704
Epoch [25/100], Loss: 39.6947
Epoch [26/100], Loss: 39.6245
Epoch [27/100], Loss: 39.5595
Epoch [28/100], Loss: 39.4993
Epoch [29/100], Loss: 39.4435
Epoch [30/100], Loss: 39.3919
Epoch [31/100], Loss: 39.3443
Epoch [32/100], Loss: 39.3003
Epoch [33/100], Loss: 39.2595
Epoch [34/100], Loss: 39.2220
Epoch [35/100], Loss: 39.1875
Epoch [36/100], Loss: 39.1558
Epoch [37/100], Loss: 39.1266
Epoch [38/100], Loss: 39.0998
Epoch [39/100], Loss: 39.0752
Epoch [40/100], Loss: 39.0525
Epoch [41/100], Loss: 39.0315
Epoch [42/100], Loss: 39.0121
Epoch [43/100], Loss: 38.9943
Epoch [44/100], Loss: 38.9778
Epoch [45/100], Loss: 38.9625
Epoch [46/100], Loss: 38.9484
Epoch [47/100], Loss: 38.9352
Epoch [48/100], Loss: 38.9229
Epoch [49/100], Loss: 38.9115
Epoch [50/100], Loss: 38.9008
Epoch [51/100], Loss: 38.8908
Epoch [52/100], Loss: 38.8814
Epoch [53/100], Loss: 38.8726
Epoch [54/100], Loss: 38.8642
Epoch [55/100], Loss: 38.8563
Epoch [56/100], Loss: 38.8488
Epoch [57/100], Loss: 38.8416
Epoch [58/100], Loss: 38.8347
Epoch [59/100], Loss: 38.8281
Epoch [60/100], Loss: 38.8218
Epoch [61/100], Loss: 38.8157
Epoch [62/100], Loss: 38.8098
Epoch [63/100], Loss: 38.8041
Epoch [64/100], Loss: 38.7985
Epoch [65/100], Loss: 38.7931
Epoch [66/100], Loss: 38.7879
Epoch [67/100], Loss: 38.7827
```

```
Epoch [68/100], Loss: 38.7777
Epoch [69/100], Loss: 38.7727
Epoch [70/100], Loss: 38.7678
Epoch [71/100], Loss: 38.7630
Epoch [72/100], Loss: 38.7582
Epoch [73/100], Loss: 38.7535
Epoch [74/100], Loss: 38.7489
Epoch [75/100], Loss: 38.7443
Epoch [76/100], Loss: 38.7398
Epoch [77/100], Loss: 38.7353
Epoch [78/100], Loss: 38.7308
Epoch [79/100], Loss: 38.7264
Epoch [80/100], Loss: 38.7220
Epoch [81/100], Loss: 38.7176
Epoch [82/100], Loss: 38.7132
Epoch [83/100], Loss: 38.7088
Epoch [84/100], Loss: 38.7044
Epoch [85/100], Loss: 38.7001
Epoch [86/100], Loss: 38.6957
Epoch [87/100], Loss: 38.6914
Epoch [88/100], Loss: 38.6870
Epoch [89/100], Loss: 38.6826
Epoch [90/100], Loss: 38.6783
Epoch [91/100], Loss: 38.6739
Epoch [92/100], Loss: 38.6695
Epoch [93/100], Loss: 38.6652
Epoch [94/100], Loss: 38.6608
Epoch [95/100], Loss: 38.6564
Epoch [96/100], Loss: 38.6520
Epoch [97/100], Loss: 38.6475
Epoch [98/100], Loss: 38.6431
Epoch [99/100], Loss: 38.6386
Epoch [100/100], Loss: 38.6342
```

[77]:
```
# test both networks and output the rmse values
fnn_clf_3.test(X_bmi_test_tensor, y_bmi_test_tensor)
rfnn_clf_3.test(X_bmi_test_tensor, y_bmi_test_tensor)

# 4 layer testing
fnn_clf_4.test(X_bmi_test_tensor, y_bmi_test_tensor)
rfnn_clf_4.test(X_bmi_test_tensor, y_bmi_test_tensor)
```

```
Test Loss: 81.537071
Test Loss: 40.916374
Test Loss: nan
Test Loss: 38.109943
```