MScBMI 33200 – Assignment III¶

Savita K Gupta¶

17 May 2023¶

In [1]:

```
#Imports

import numpy as np
import pandas as pd
from scipy.stats import sem
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LogisticRegressionCV
from sklearn.linear_model import Ridge
from sklearn.linear_model import RidgeCV

from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler

from sklearn.metrics import roc_auc_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score

from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier

from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

from sklearn.neural_network import MLPClassifier
```

Section 1: ER Bots 30-Day Readmission Study¶

S1 Question 1: Naive Model¶

In [2]:

```
# Import train and test dataset

rm_train_full = pd.read_csv(r'C:\Users\vitak\Downloads\readmission_train.csv')
rm_test_full = pd.read_csv(r'C:\Users\vitak\Downloads\readmission_test.csv')
```

In [3]:

```
# Setup xTrain and yTrain
```

```
rm_n = rm_train_full.drop(rm_train_full.columns[[2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17]], axis=1)

rm_Xtrain_n = rm_n.drop(["outcome"], axis=1)
rm_Ytrain_n = rm_n['outcome']
```

In [4]:

```
#Setup xTest and yTest

rm_n_test = rm_test_full.drop(rm_train_full.columns[[2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17]],
axis=1)

rm_Xtest_n = rm_n_test.drop(["outcome"], axis=1)
rm_Ytest_n = rm_n_test['outcome']
```

In [5]:

```
# Run logistic regression

naive_model = LogisticRegression()
naive_model.fit(rm_Xtrain_n, rm_Ytrain_n)
```

Out[5]:

```
LogisticRegression()
```

In [6]:

```
#Calculate AUC using predict_proba

naive_model_prediction = naive_model.predict_proba(rm_Xtest_n)
print("Naive Model train set AUC score - with predict proba: %f" % roc_auc_score(rm_Ytest_n,
naive_model_prediction[:,1]))
```

```
Naive Model train set AUC score - with predict proba: 0.499419
```

In [7]:

```
# Calculate Confidence Interval using bootstrap for Naive model

#Y train value for Naive model
y_true_n = np.array(rm_Ytest_n)

#Predict_proba value for Naive model
y_pred_n = np.array(naive_model_prediction[:,1])


n_bootstraps = 100
rng_seed = 42  # control reproducibility
bootstrapped_scores = []


rng = np.random.RandomState(rng_seed)

for i in range(n_bootstraps):
    indices = rng.randint(0, len(y_pred_n), len(y_pred_n))
    if len(np.unique(y_true_n[indices])) < 2:
        continue

    score = roc_auc_score(y_true_n[indices], y_pred_n[indices])
```

```
    bootstrapped_scores.append(score)

sorted_scores = np.array(bootstrapped_scores)
sorted_scores.sort()

con_low_n = sorted_scores[int(0.05 * len(sorted_scores))]
con_up_n = sorted_scores[int(0.95 * len(sorted_scores))]
print("Confidence interval for the score: [{:0.5f} - {:0.5}]".format(con_low_n, con_up_n))
```

```
Confidence interval for the score: [0.41708 - 0.56076]
```

Cross Validation: Naive Model¶

In [8]:

```
#Cross Validation with kfold
k = 5
rm_kf = KFold(n_splits=k, random_state=None)
rm_model = LogisticRegression(solver= 'liblinear')

result = cross_val_score(rm_model , rm_Xtest_n, rm_Ytest_n, cv = rm_kf)
print("Avg accuracy: {}".format(result.mean()))
```

```
Avg accuracy: 0.9965800273597811
```

In [9]:

```
#LogisticRegressionCV (KFold=5)

rm_log_cv = LogisticRegressionCV(cv=5, random_state = 0)
rm_log_cv.fit(rm_Xtrain_n, rm_Ytrain_n)

rm_log_cv_predict = rm_log_cv.predict_proba(rm_Xtest_n)

rm_n_auc = roc_auc_score(rm_Ytest_n, rm_log_cv_predict[:,1])
print("Naive Model training AUC (with Kfold CV): %f" % rm_n_auc)
```

```
Naive Model training AUC (with Kfold CV): 0.526428
```

In [10]:

```
# Calculate Confidence Interval using bootstrap for Naive model (with LogReg Kfold CV)

#Y train value for Naive model
y_true_n = np.array(rm_Ytest_n)

#Predict_proba value for Naive model
y_pred_n = np.array(rm_log_cv_predict[:,1])


n_bootstraps = 100
rng_seed = 42  # control reproducibility
bootstrapped_scores = []


rng = np.random.RandomState(rng_seed)

for i in range(n_bootstraps):
    indices = rng.randint(0, len(y_pred_n), len(y_pred_n))
```

```
    if len(np.unique(y_true_n[indices])) < 2:
        continue

    score = roc_auc_score(y_true_n[indices], y_pred_n[indices])
    bootstrapped_scores.append(score)

sorted_scores = np.array(bootstrapped_scores)
sorted_scores.sort()

cv_con_low_n = sorted_scores[int(0.05 * len(sorted_scores))]
cv_con_up_n = sorted_scores[int(0.95 * len(sorted_scores))]
print("Confidence interval for the score: [{:0.5f} - {:0.5}]".format(cv_con_low_n, cv_con_up_n))
```

```
Confidence interval for the score: [0.46040 - 0.58978]
```

S1 Question 2: GLM Model¶

In [11]:

```
# Setup xTrain and yTrain

rm_lr_train = rm_train_full.drop_duplicates(keep='last')

rm_Xtrain_lr = rm_lr_train.drop(["outcome"], axis=1)
rm_Ytrain_lr = rm_lr_train['outcome']
```

In [12]:

```
#Setup xTest and yTest

rm_lr_test = rm_test_full.drop_duplicates(keep='last')

rm_Xtest_lr = rm_lr_test.drop(["outcome"], axis=1)
rm_Ytest_lr = rm_lr_test['outcome']
```

In [13]:

```
# Run logistic regression

lr_model = LogisticRegression()
lr_model.fit(rm_Xtrain_lr, rm_Ytrain_lr)
```

Out[13]:

```
LogisticRegression()
```

In [14]:

```
#Calculate AUC

lr_model_prediction = lr_model.predict_proba(rm_Xtest_lr)
print("Logistic Regression train set AUC score: %f" % roc_auc_score(rm_Ytest_lr,
lr_model_prediction[:,1]))
```

```
Logistic Regression train set AUC score: 0.512161
```

In [15]:

```
# Calculate Confidence Interval using bootstrap for Logistic Regression model
```

```
#Y train value for Logistic Regression model
y_true_lr = np.array(rm_Ytest_lr)

#Predict_proba value for Logistic Regression model
y_pred_lr = np.array(lr_model_prediction[:,1])

n_bootstraps = 100
rng_seed = 42  # control reproducibility
bootstrapped_scores = []

rng = np.random.RandomState(rng_seed)

for i in range(n_bootstraps):
    indices = rng.randint(0, len(y_pred_lr), len(y_pred_lr))
    if len(np.unique(y_true_lr[indices])) < 2:
        continue

    score = roc_auc_score(y_true_lr[indices], y_pred_lr[indices])
    bootstrapped_scores.append(score)

sorted_scores = np.array(bootstrapped_scores)
sorted_scores.sort()

con_low_lr = sorted_scores[int(0.05 * len(sorted_scores))]
con_up_lr = sorted_scores[int(0.95 * len(sorted_scores))]
print("Confidence interval for the score: [{:0.5f} - {:0.5}]".format(con_low_lr, con_up_lr))
```

```
Confidence interval for the score: [0.43869 - 0.57803]
```

Cross Validation: GLM Model¶

In [16]:

```
#Cross Validation with kfold
k = 5
rm_kf_lr = KFold(n_splits=k, random_state=None)
rm_model_lr = LogisticRegression(solver= 'liblinear')

result_lr = cross_val_score(lr_model , rm_Xtest_lr, rm_Ytest_lr, cv = rm_kf_lr)
print("Avg accuracy: {}".format(result_lr.mean()))
```

```
Avg accuracy: 0.9965800273597811
```

In [17]:

```
#LogisticRegressionCV (KFold=5)

lr_log_cv = LogisticRegressionCV(cv=5, random_state = 0)
lr_log_cv.fit(rm_Xtrain_lr, rm_Ytrain_lr)

lr_log_cv_predict = lr_log_cv.predict_proba(rm_Xtest_lr)

rm_lr_auc = roc_auc_score(rm_Ytest_lr, lr_log_cv_predict[:,1])
print("GLM Model training AUC (with Kfold CV): %f" % rm_lr_auc)
```

```
GLM Model training AUC (with Kfold CV): 0.543278
```

In [18]:

```
# Calculate Confidence Interval using bootstrap for Logistic Regression model (with Kfold CV)

#Y train value for Naive model
y_true_lr = np.array(rm_Ytest_lr)

#Predict_proba value for Naive model
y_pred_lr = np.array(lr_log_cv_predict[:,1])


n_bootstraps = 100
rng_seed = 42  # control reproducibility
bootstrapped_scores = []


rng = np.random.RandomState(rng_seed)

for i in range(n_bootstraps):
    indices = rng.randint(0, len(y_pred_lr), len(y_pred_lr))
    if len(np.unique(y_true_lr[indices])) < 2:
        continue

    score = roc_auc_score(y_true_lr[indices], y_pred_lr[indices])
    bootstrapped_scores.append(score)

sorted_scores = np.array(bootstrapped_scores)
sorted_scores.sort()

cv_con_low_lr = sorted_scores[int(0.05 * len(sorted_scores))]
cv_con_up_lr = sorted_scores[int(0.95 * len(sorted_scores))]
print("Confidence interval for the score: [{:0.5f} - {:0.5}]".format(cv_con_low_lr, cv_con_up_lr))
```

```
Confidence interval for the score: [0.46654 - 0.60726]
```

S1 Question 3: Neural Net Classifier¶

In [19]:

```
mlp = MLPClassifier(
    hidden_layer_sizes=(100,80,10,1),
    activation='relu',
    max_iter=500,
    alpha=0.10,
    solver="sgd",
    verbose=10,
    random_state=1,
    learning_rate='adaptive',
    learning_rate_init=0.001,
)


mlp.fit(rm_Xtrain_lr, rm_Ytrain_lr)

mlp_predtrain = mlp.predict_proba(rm_Xtrain_lr)
print("Train set AUC score: %f" % roc_auc_score(rm_Ytrain_lr, mlp_predtrain[:,1]))


mlp_predtest = mlp.predict_proba(rm_Xtest_lr)
print("Train set AUC score: %f" % roc_auc_score(rm_Ytest_lr, mlp_predtest[:,1]))
```

```
Iteration 1, loss = 1.15732308
Iteration 2, loss = 0.77207644
Iteration 3, loss = 0.54423731
Iteration 4, loss = 0.40992959
Iteration 5, loss = 0.32656229
Iteration 6, loss = 0.27162578
Iteration 7, loss = 0.23344792
Iteration 8, loss = 0.20571243
Iteration 9, loss = 0.18481482
Iteration 10, loss = 0.16859700
Iteration 11, loss = 0.15569664
Iteration 12, loss = 0.14521574
Iteration 13, loss = 0.13655940
Iteration 14, loss = 0.12929547
Iteration 15, loss = 0.12313022
Iteration 16, loss = 0.11783121
Iteration 17, loss = 0.11323331
Iteration 18, loss = 0.10921284
Iteration 19, loss = 0.10566712
Iteration 20, loss = 0.10252006
Iteration 21, loss = 0.09970892
Iteration 22, loss = 0.09718369
Iteration 23, loss = 0.09490381
Iteration 24, loss = 0.09283553
Iteration 25, loss = 0.09095184
Iteration 26, loss = 0.08923024
Iteration 27, loss = 0.08764949
Iteration 28, loss = 0.08619258
Iteration 29, loss = 0.08484827
Iteration 30, loss = 0.08360235
Iteration 31, loss = 0.08244613
Iteration 32, loss = 0.08136827
Iteration 33, loss = 0.08036236
Iteration 34, loss = 0.07942191
Iteration 35, loss = 0.07853851
Iteration 36, loss = 0.07770997
Iteration 37, loss = 0.07692921
Iteration 38, loss = 0.07619374
Iteration 39, loss = 0.07549965
Iteration 40, loss = 0.07484266
Iteration 41, loss = 0.07422010
Iteration 42, loss = 0.07363016
Iteration 43, loss = 0.07306858
Iteration 44, loss = 0.07253567
Iteration 45, loss = 0.07202734
Iteration 46, loss = 0.07154264
Iteration 47, loss = 0.07108051
Iteration 48, loss = 0.07063765
Iteration 49, loss = 0.07021467
Iteration 50, loss = 0.06980999
Iteration 51, loss = 0.06942186
Iteration 52, loss = 0.06904900
Iteration 53, loss = 0.06869067
Iteration 54, loss = 0.06834631
Iteration 55, loss = 0.06801406
Iteration 56, loss = 0.06769524
Iteration 57, loss = 0.06738745
Iteration 58, loss = 0.06709080
Iteration 59, loss = 0.06680453
Iteration 60, loss = 0.06652718
```

```
Iteration 61, loss = 0.06626047
Iteration 62, loss = 0.06600186
Iteration 63, loss = 0.06575170
Iteration 64, loss = 0.06550937
Iteration 65, loss = 0.06527459
Iteration 66, loss = 0.06504677
Iteration 67, loss = 0.06482587
Iteration 68, loss = 0.06461141
Iteration 69, loss = 0.06440268
Iteration 70, loss = 0.06419998
Iteration 71, loss = 0.06400305
Iteration 72, loss = 0.06381120
Iteration 73, loss = 0.06362457
Iteration 74, loss = 0.06344306
Iteration 75, loss = 0.06326656
Iteration 76, loss = 0.06309389
Iteration 77, loss = 0.06292556
Iteration 78, loss = 0.06276140
Iteration 79, loss = 0.06260127
Iteration 80, loss = 0.06244486
Iteration 81, loss = 0.06229189
Iteration 82, loss = 0.06214261
Iteration 83, loss = 0.06199634
Iteration 84, loss = 0.06185374
Iteration 85, loss = 0.06171394
Iteration 86, loss = 0.06157731
Iteration 87, loss = 0.06144356
Iteration 88, loss = 0.06131215
Iteration 89, loss = 0.06118358
Iteration 90, loss = 0.06105744
Iteration 91, loss = 0.06093412
Iteration 92, loss = 0.06081282
Iteration 93, loss = 0.06069396
Iteration 94, loss = 0.06057773
Iteration 95, loss = 0.06046359
Iteration 96, loss = 0.06035158
Iteration 97, loss = 0.06024083
Iteration 98, loss = 0.06013240
Iteration 99, loss = 0.06002559
Iteration 100, loss = 0.05992082
Iteration 101, loss = 0.05981778
Iteration 102, loss = 0.05971623
Iteration 103, loss = 0.05961646
Iteration 104, loss = 0.05951869
Iteration 105, loss = 0.05942213
Iteration 106, loss = 0.05932713
Iteration 107, loss = 0.05923352
Iteration 108, loss = 0.05914107
Iteration 109, loss = 0.05905019
Iteration 110, loss = 0.05896059
Iteration 111, loss = 0.05887263
Iteration 112, loss = 0.05878578
Iteration 113, loss = 0.05869997
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Setting learning
rate to 0.000200
Iteration 114, loss = 0.05864381
Iteration 115, loss = 0.05862647
Iteration 116, loss = 0.05860972
Iteration 117, loss = 0.05859297
Iteration 118, loss = 0.05857629
Iteration 119, loss = 0.05855965
```

```
Iteration 120, loss = 0.05854299
Iteration 121, loss = 0.05852643
Iteration 122, loss = 0.05850986
Iteration 123, loss = 0.05849337
Iteration 124, loss = 0.05847693
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Setting learning
rate to 0.000040
Iteration 125, loss = 0.05846612
Iteration 126, loss = 0.05846273
Iteration 127, loss = 0.05845945
Iteration 128, loss = 0.05845618
Iteration 129, loss = 0.05845291
Iteration 130, loss = 0.05844963
Iteration 131, loss = 0.05844635
Iteration 132, loss = 0.05844309
Iteration 133, loss = 0.05843982
Iteration 134, loss = 0.05843656
Iteration 135, loss = 0.05843330
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Setting learning
rate to 0.000008
Iteration 136, loss = 0.05843114
Iteration 137, loss = 0.05843047
Iteration 138, loss = 0.05842982
Iteration 139, loss = 0.05842917
Iteration 140, loss = 0.05842852
Iteration 141, loss = 0.05842786
Iteration 142, loss = 0.05842721
Iteration 143, loss = 0.05842656
Iteration 144, loss = 0.05842591
Iteration 145, loss = 0.05842526
Iteration 146, loss = 0.05842461
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Setting learning
rate to 0.000002
Iteration 147, loss = 0.05842418
Iteration 148, loss = 0.05842405
Iteration 149, loss = 0.05842392
Iteration 150, loss = 0.05842379
Iteration 151, loss = 0.05842366
Iteration 152, loss = 0.05842353
Iteration 153, loss = 0.05842340
Iteration 154, loss = 0.05842327
Iteration 155, loss = 0.05842314
Iteration 156, loss = 0.05842300
Iteration 157, loss = 0.05842287
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Setting learning
rate to 0.000000
Iteration 158, loss = 0.05842279
Iteration 159, loss = 0.05842276
Iteration 160, loss = 0.05842274
Iteration 161, loss = 0.05842271
Iteration 162, loss = 0.05842268
Iteration 163, loss = 0.05842266
Iteration 164, loss = 0.05842263
Iteration 165, loss = 0.05842261
Iteration 166, loss = 0.05842258
Iteration 167, loss = 0.05842255
Iteration 168, loss = 0.05842253
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Learning rate too
small. Stopping.
Train set AUC score: 0.500000
```

```
Train set AUC score: 0.500000
```

Cross Validation: Neural Net Model¶

#This section must run overnight - mlp GRIDSEARCH can take up to 8 hours. mlp = MLPClassifier(max_iter=200) parameter_space = { 'hidden_layer_sizes': [(50,20,10,1),(50,10),(20,10,5), (100,80,10,1)], 'activation': ['relu'], 'solver': ['sgd','adam'], 'batch_size': [300,500], 'alpha': [0.10,0.40,0.60,0.80,1.0], 'random_state': [1], 'learning_rate': ['constant', 'adaptive'], 'learning_rate_init': [0.1, 0.01, 0.001] } clf = GridSearchCV(mlp, parameter_space, n_jobs=-1, cv=5) clf.fit(rm_Xtrain_lr, rm_Ytrain_lr) print('Best parameters found:\n', clf.best_params_)
In [20]:

```
#Calculate AUC after GridSearch

mlp = MLPClassifier(
    hidden_layer_sizes=(50,20,10,1),
    activation='relu',
    solver="sgd",
    batch_size = 300,
    alpha=0.1,
    random_state=1,
    learning_rate='constant',
    learning_rate_init=0.1,
    max_iter=500,
    verbose=10,
)


mlp.fit(rm_Xtrain_lr, rm_Ytrain_lr)

mlp_predtrain = mlp.predict_proba(rm_Xtrain_lr)
print("Train set AUC score: %f" % roc_auc_score(rm_Ytrain_lr, mlp_predtrain[:,1]))

mlp_predtest = mlp.predict_proba(rm_Xtest_lr)
print("Test set AUC score: %f" % roc_auc_score(rm_Ytest_lr, mlp_predtest[:,1]))
```

```
Iteration 1, loss = 0.17746690
Iteration 2, loss = 0.03565157
Iteration 3, loss = 0.03495925
Iteration 4, loss = 0.03437503
Iteration 5, loss = 0.03385178
Iteration 6, loss = 0.03336765
Iteration 7, loss = 0.03291337
Iteration 8, loss = 0.03248774
Iteration 9, loss = 0.03208162
Iteration 10, loss = 0.03169665
Iteration 11, loss = 0.03133468
Iteration 12, loss = 0.03098504
Iteration 13, loss = 0.03065611
Iteration 14, loss = 0.03034100
Iteration 15, loss = 0.03004367
Iteration 16, loss = 0.02975651
Iteration 17, loss = 0.02948066
Iteration 18, loss = 0.02922234
Iteration 19, loss = 0.02898125
Iteration 20, loss = 0.02873767
Iteration 21, loss = 0.02851857
Iteration 22, loss = 0.02830582
Iteration 23, loss = 0.02810097
```

```
Iteration 24, loss = 0.02790380
Iteration 25, loss = 0.02772029
Iteration 26, loss = 0.02754282
Iteration 27, loss = 0.02737829
Iteration 28, loss = 0.02721541
Iteration 29, loss = 0.02706395
Iteration 30, loss = 0.02691763
Iteration 31, loss = 0.02678367
Iteration 32, loss = 0.02664662
Iteration 33, loss = 0.02652123
Iteration 34, loss = 0.02640304
Iteration 35, loss = 0.02628797
Iteration 36, loss = 0.02617875
Iteration 37, loss = 0.02607648
Iteration 38, loss = 0.02597706
Iteration 39, loss = 0.02588207
Iteration 40, loss = 0.02579810
Iteration 41, loss = 0.02570791
Iteration 42, loss = 0.02562843
Iteration 43, loss = 0.02555246
Iteration 44, loss = 0.02547759
Iteration 45, loss = 0.02540935
Iteration 46, loss = 0.02533895
Iteration 47, loss = 0.02527500
Iteration 48, loss = 0.02521629
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
Train set AUC score: 0.500000
Test set AUC score: 0.500000
```

In [22]:

```
#Calculate Confidence Interval After Neural Net GridSearch

y_true_nn = np.array(rm_Ytest_lr)
y_pred_nn = np.array(mlp_predtest[:,1])

n_bootstraps = 100
rng_seed = 42
bootstrapped_scores = []

rng = np.random.RandomState(rng_seed)
for i in range(n_bootstraps):
    indices = rng.randint(0, len(y_pred_nn), len(y_pred_nn))
    if len(np.unique(y_true_nn[indices])) < 2:
        continue

    score = roc_auc_score(y_true_nn[indices], y_pred_nn[indices])
    bootstrapped_scores.append(score)

sorted_scores = np.array(bootstrapped_scores)
sorted_scores.sort()

confidence_lower_nn = sorted_scores[int(0.05 * len(sorted_scores))]
confidence_upper_nn = sorted_scores[int(0.95 * len(sorted_scores))]
print("Confidence interval for the score: [{:0.5f} - {:0.5}]".format(confidence_lower_nn,
confidence_upper_nn))
```

```
Confidence interval for the score: [0.50000 - 0.5]
```

Section 2: Gusto Study¶

S2 Question 1: GLM Model¶

In [23]:

```
# Import Data

g_train = pd.read_csv(r'C:\Users\vitak\Downloads\gusto_train.csv')
g_test = pd.read_csv(r'C:\Users\vitak\Downloads\gusto_test.csv')
```

In [24]:

```
# Setup xTrain and yTrain

g_Xtrain = g_train.drop(["DAY30"], axis=1)
g_Ytrain = g_train['DAY30']
```

In [25]:

```
# Setup xTest and yTest

g_Xtest = g_test.drop(["DAY30"], axis=1)
g_Ytest = g_test['DAY30']
```

In [26]:

```
# Run logistic regression

g_lr_model = LogisticRegression()
g_lr_model.fit(g_Xtrain, g_Ytrain)
```

Out[26]:

```
LogisticRegression()
```

In [27]:

```
#Calculate AUC - GUSTO GLM

g_lr_model_prediction = g_lr_model.predict_proba(g_Xtest)
print("GUSTO Logistic Regression AUC score: %f" % roc_auc_score(g_Ytest,
g_lr_model_prediction[:,1]))
```

```
GUSTO Logistic Regression AUC score: 0.826736
```

In [28]:

```
# Calculate Confidence Interval using bootstrap for GUSTO Logistic Regression model

#Y train value for Logistic Regression model
gy_true_lr = np.array(g_Ytest)

#Predict_proba value for Logistic Regression model
gy_pred_lr = np.array(g_lr_model_prediction[:,1])


n_bootstraps = 100
rng_seed = 42  # control reproducibility
```

```
bootstrapped_scores = []


rng = np.random.RandomState(rng_seed)

for i in range(n_bootstraps):
    indices = rng.randint(0, len(gy_pred_lr), len(gy_pred_lr))
    if len(np.unique(gy_true_lr[indices])) < 2:
        continue

    score = roc_auc_score(gy_true_lr[indices], gy_pred_lr[indices])
    bootstrapped_scores.append(score)

sorted_scores = np.array(bootstrapped_scores)
sorted_scores.sort()

gcon_low_lr = sorted_scores[int(0.05 * len(sorted_scores))]
gcon_up_lr = sorted_scores[int(0.95 * len(sorted_scores))]
print("Confidence interval for the score: [{:0.5f} - {:0.5}]".format(gcon_low_lr, gcon_up_lr))
```

```
Confidence interval for the score: [0.79637 - 0.8585]
```

Cross Validation: GUSTO GLM¶

In [29]:

```
#Cross Validation with kfold
k = 5
g_kf = KFold(n_splits=k, random_state=None)
g_model = LogisticRegression(solver= 'liblinear')

gresult_lr = cross_val_score(g_model , g_Xtest, g_Ytest, cv = g_kf)
print("Avg accuracy: {}".format(gresult_lr.mean()))
```

```
Avg accuracy: 0.9405776203462797
```

In [30]:

```
#LogisticRegressionCV (KFold=5)

g_log_cv = LogisticRegressionCV(cv=5, random_state = 0)
g_log_cv.fit(g_Xtrain, g_Ytrain)

g_log_cv_predict = g_log_cv.predict_proba(g_Xtest)

g_lr_auc = roc_auc_score(g_Ytest, g_log_cv_predict[:,1])
print("GUSTO GLM AUC (with Kfold CV): %f" % g_lr_auc)
```

```
GUSTO GLM AUC (with Kfold CV): 0.832318
```

In [31]:

```
# Calculate Confidence Interval using bootstrap for GUSTO Logistic Regression model (with Kfold CV)

#Y train value for Logistic Regression model
gy_true_cv = np.array(g_Ytest)

#Predict_proba value for Logistic Regression model
gy_pred_cv = np.array(g_log_cv_predict[:,1])
```

```
n_bootstraps = 100
rng_seed = 42  # control reproducibility
bootstrapped_scores = []

rng = np.random.RandomState(rng_seed)

for i in range(n_bootstraps):
    indices = rng.randint(0, len(gy_pred_cv), len(gy_pred_cv))
    if len(np.unique(gy_true_cv[indices])) < 2:
        continue

    score = roc_auc_score(gy_true_cv[indices], gy_pred_cv[indices])
    bootstrapped_scores.append(score)

sorted_scores = np.array(bootstrapped_scores)
sorted_scores.sort()

gcon_low_cv = sorted_scores[int(0.05 * len(sorted_scores))]
gcon_up_cv = sorted_scores[int(0.95 * len(sorted_scores))]
print("Confidence interval for the score: [{:0.5f} - {:0.5}]".format(gcon_low_cv, gcon_up_cv))
```

```
Confidence interval for the score: [0.80082 - 0.86391]
```

S2 Question 2: Ridge Regression Model¶

In [32]:

```
#Run Ridge Regression
r_alphas = np.logspace(0,5,100)
ridge = RidgeCV(alphas = r_alphas, scoring = 'r2')
ridge.fit(g_Xtrain, g_Ytrain)
```

Out[32]:

```
RidgeCV(alphas=array([1.00000000e+00, 1.12332403e+00, 1.26185688e+00, 1.41747416e+00,
       1.59228279e+00, 1.78864953e+00, 2.00923300e+00, 2.25701972e+00,
       2.53536449e+00, 2.84803587e+00, 3.19926714e+00, 3.59381366e+00,
       4.03701726e+00, 4.53487851e+00, 5.09413801e+00, 5.72236766e+00,
       6.42807312e+00, 7.22080902e+00, 8.11130831e+00, 9.11162756e+00,
       1.02353102e+01, 1.14975700e+0...
       6.89261210e+03, 7.74263683e+03, 8.69749003e+03, 9.77009957e+03,
       1.09749877e+04, 1.23284674e+04, 1.38488637e+04, 1.55567614e+04,
       1.74752840e+04, 1.96304065e+04, 2.20513074e+04, 2.47707636e+04,
       2.78255940e+04, 3.12571585e+04, 3.51119173e+04, 3.94420606e+04,
       4.43062146e+04, 4.97702356e+04, 5.59081018e+04, 6.28029144e+04,
       7.05480231e+04, 7.92482898e+04, 8.90215085e+04, 1.00000000e+05]),
        scoring='r2')
```

In [33]:

```
#Calculate AUC
grr_prob = ridge.predict(g_Xtest)
grr_auc_roc = roc_auc_score(g_Ytest, grr_prob)
print("Ridge Regression AUC: %f" % grr_auc_roc)
```

```
Ridge Regression AUC: 0.822229
```

In [34]:

```
# Calculate Confidence Interval using bootstrap for GUSTO Ridge Regression model

#Y train value for Logistic Regression model
grr_true = np.array(g_Ytest)

#Predict_proba value for Logistic Regression model
grr_pred = np.array(grr_prob)


n_bootstraps = 100
rng_seed = 42  # control reproducibility
bootstrapped_scores = []

rng = np.random.RandomState(rng_seed)

for i in range(n_bootstraps):
    indices = rng.randint(0, len(grr_pred), len(grr_pred))
    if len(np.unique(grr_true[indices])) < 2:
        continue

    score = roc_auc_score(grr_true[indices], grr_pred[indices])
    bootstrapped_scores.append(score)

sorted_scores = np.array(bootstrapped_scores)
sorted_scores.sort()

grr_low = sorted_scores[int(0.05 * len(sorted_scores))]
grr_up = sorted_scores[int(0.95 * len(sorted_scores))]
print("Confidence interval for the score: [{:0.5f} - {:0.5}]".format(grr_low, grr_up))
```

```
Confidence interval for the score: [0.79003 - 0.856]
```

Cross Validation: GUSTO Ridge Regression¶

In [35]:

```
#Cross Validation with kfold
k = 5
kf = KFold(n_splits=k, random_state=None)
model1 = RidgeCV(alphas= r_alphas, cv = k)

result = cross_val_score(model1 , g_Xtest, g_Ytest, cv = kf)
print("Avg accuracy: {}".format(result.mean()))
```

```
Avg accuracy: 0.10941105301450275
```

In [36]:

```
#Run RidgeCV with KFold=5

grr_cv = RidgeCV(cv=5)
grr_cv.fit(g_Xtrain,g_Ytrain)

grr_pred_cv = grr_cv.predict(g_Xtest)

gcv_auc_roc = roc_auc_score(g_Ytest, grr_pred_cv)
print("Training AUC: %f" % gcv_auc_roc)
```

```
Training AUC: 0.821861
```

In [37]:

```
# Calculate Confidence Interval using bootstrap for GUSTO Ridge Regression model (with Kfold CV)

#Y train value for Logistic Regression model
grr_true_cv = np.array(g_Ytest)

#Predict value for Ridge Regression model
grr_pred_cv1 = np.array(grr_pred_cv)


n_bootstraps = 100
rng_seed = 42  # control reproducibility
bootstrapped_scores = []

rng = np.random.RandomState(rng_seed)

for i in range(n_bootstraps):
    indices = rng.randint(0, len(grr_pred_cv1), len(grr_pred_cv1))
    if len(np.unique(grr_true_cv[indices])) < 2:
        continue

    score = roc_auc_score(grr_true_cv[indices], grr_pred_cv1[indices])
    bootstrapped_scores.append(score)

sorted_scores = np.array(bootstrapped_scores)
sorted_scores.sort()

grr_low_cv = sorted_scores[int(0.05 * len(sorted_scores))]
grr_up_cv = sorted_scores[int(0.95 * len(sorted_scores))]
print("Confidence interval for the score: [{:0.5f} - {:0.5}]".format(grr_low_cv, grr_up_cv))
```

```
Confidence interval for the score: [0.78978 - 0.85573]
```

S2 Question 3: GUSTO Neural Net Model¶

In [38]:

```
gmlp = MLPClassifier(
    hidden_layer_sizes=(100,80,10,1),
    activation='relu',
    max_iter=500,
    alpha=0.10,
    solver="sgd",
    verbose=10,
    random_state=1,
    learning_rate='adaptive',
    learning_rate_init=0.001,
)


gmlp.fit(g_Xtrain, g_Ytrain)

gmlp_predtrain = gmlp.predict_proba(g_Xtrain)
print("Train set AUC score: %f" % roc_auc_score(g_Ytrain, gmlp_predtrain[:,1]))
```

```
gmlp_predtest = gmlp.predict_proba(g_Xtest)
print("Train set AUC score: %f" % roc_auc_score(g_Ytest, gmlp_predtest[:,1]))
```

```
Iteration 1, loss = 1.31236731
Iteration 2, loss = 1.29154951
Iteration 3, loss = 1.26254330
Iteration 4, loss = 1.23076716
Iteration 5, loss = 1.19815055
Iteration 6, loss = 1.16601123
Iteration 7, loss = 1.13487002
Iteration 8, loss = 1.10436498
Iteration 9, loss = 1.07494032
Iteration 10, loss = 1.04663807
Iteration 11, loss = 1.01935594
Iteration 12, loss = 0.99310518
Iteration 13, loss = 0.96787853
Iteration 14, loss = 0.94346823
Iteration 15, loss = 0.92007136
Iteration 16, loss = 0.89756764
Iteration 17, loss = 0.87567504
Iteration 18, loss = 0.85489052
Iteration 19, loss = 0.83499409
Iteration 20, loss = 0.81582148
Iteration 21, loss = 0.79724668
Iteration 22, loss = 0.77962692
Iteration 23, loss = 0.76259662
Iteration 24, loss = 0.74632744
Iteration 25, loss = 0.73057282
Iteration 26, loss = 0.71559363
Iteration 27, loss = 0.70111971
Iteration 28, loss = 0.68709223
Iteration 29, loss = 0.67381627
Iteration 30, loss = 0.66087546
Iteration 31, loss = 0.64859847
Iteration 32, loss = 0.63675890
Iteration 33, loss = 0.62544051
Iteration 34, loss = 0.61442427
Iteration 35, loss = 0.60398499
Iteration 36, loss = 0.59390704
Iteration 37, loss = 0.58419201
Iteration 38, loss = 0.57498058
Iteration 39, loss = 0.56597638
Iteration 40, loss = 0.55737579
Iteration 41, loss = 0.54915126
Iteration 42, loss = 0.54124735
Iteration 43, loss = 0.53358895
Iteration 44, loss = 0.52617024
Iteration 45, loss = 0.51911539
Iteration 46, loss = 0.51227741
Iteration 47, loss = 0.50568916
Iteration 48, loss = 0.49931539
Iteration 49, loss = 0.49321829
Iteration 50, loss = 0.48725789
Iteration 51, loss = 0.48152504
Iteration 52, loss = 0.47601297
Iteration 53, loss = 0.47069036
Iteration 54, loss = 0.46558894
Iteration 55, loss = 0.46064778
Iteration 56, loss = 0.45590585
Iteration 57, loss = 0.45128652
Iteration 58, loss = 0.44680325
```

```
Iteration 59, loss = 0.44257387
Iteration 60, loss = 0.43836510
Iteration 61, loss = 0.43433457
Iteration 62, loss = 0.43053706
Iteration 63, loss = 0.42671493
Iteration 64, loss = 0.42307913
Iteration 65, loss = 0.41954710
Iteration 66, loss = 0.41612727
Iteration 67, loss = 0.41288221
Iteration 68, loss = 0.40969204
Iteration 69, loss = 0.40658458
Iteration 70, loss = 0.40359761
Iteration 71, loss = 0.40071412
Iteration 72, loss = 0.39791856
Iteration 73, loss = 0.39522108
Iteration 74, loss = 0.39251946
Iteration 75, loss = 0.38997339
Iteration 76, loss = 0.38747614
Iteration 77, loss = 0.38507367
Iteration 78, loss = 0.38270038
Iteration 79, loss = 0.38036397
Iteration 80, loss = 0.37819214
Iteration 81, loss = 0.37605706
Iteration 82, loss = 0.37397577
Iteration 83, loss = 0.37196239
Iteration 84, loss = 0.36996067
Iteration 85, loss = 0.36803394
Iteration 86, loss = 0.36620572
Iteration 87, loss = 0.36434184
Iteration 88, loss = 0.36253859
Iteration 89, loss = 0.36081063
Iteration 90, loss = 0.35911520
Iteration 91, loss = 0.35745326
Iteration 92, loss = 0.35582475
Iteration 93, loss = 0.35429867
Iteration 94, loss = 0.35277043
Iteration 95, loss = 0.35128044
Iteration 96, loss = 0.34987288
Iteration 97, loss = 0.34841858
Iteration 98, loss = 0.34704019
Iteration 99, loss = 0.34567896
Iteration 100, loss = 0.34437833
Iteration 101, loss = 0.34305819
Iteration 102, loss = 0.34185099
Iteration 103, loss = 0.34059931
Iteration 104, loss = 0.33939864
Iteration 105, loss = 0.33825181
Iteration 106, loss = 0.33713885
Iteration 107, loss = 0.33598902
Iteration 108, loss = 0.33494411
Iteration 109, loss = 0.33387372
Iteration 110, loss = 0.33283972
Iteration 111, loss = 0.33184282
Iteration 112, loss = 0.33087627
Iteration 113, loss = 0.32991800
Iteration 114, loss = 0.32894926
Iteration 115, loss = 0.32806938
Iteration 116, loss = 0.32718674
Iteration 117, loss = 0.32631713
Iteration 118, loss = 0.32549772
```

```
Iteration 119, loss = 0.32467060
Iteration 120, loss = 0.32385318
Iteration 121, loss = 0.32306877
Iteration 122, loss = 0.32228462
Iteration 123, loss = 0.32153596
Iteration 124, loss = 0.32080463
Iteration 125, loss = 0.32008347
Iteration 126, loss = 0.31936026
Iteration 127, loss = 0.31866061
Iteration 128, loss = 0.31802177
Iteration 129, loss = 0.31735204
Iteration 130, loss = 0.31669922
Iteration 131, loss = 0.31606825
Iteration 132, loss = 0.31545674
Iteration 133, loss = 0.31485220
Iteration 134, loss = 0.31425611
Iteration 135, loss = 0.31365778
Iteration 136, loss = 0.31308115
Iteration 137, loss = 0.31253490
Iteration 138, loss = 0.31197677
Iteration 139, loss = 0.31143666
Iteration 140, loss = 0.31090954
Iteration 141, loss = 0.31037585
Iteration 142, loss = 0.30986824
Iteration 143, loss = 0.30934827
Iteration 144, loss = 0.30884997
Iteration 145, loss = 0.30834342
Iteration 146, loss = 0.30787879
Iteration 147, loss = 0.30741721
Iteration 148, loss = 0.30695459
Iteration 149, loss = 0.30652143
Iteration 150, loss = 0.30607240
Iteration 151, loss = 0.30566253
Iteration 152, loss = 0.30523092
Iteration 153, loss = 0.30482504
Iteration 154, loss = 0.30442509
Iteration 155, loss = 0.30403203
Iteration 156, loss = 0.30364422
Iteration 157, loss = 0.30327597
Iteration 158, loss = 0.30289292
Iteration 159, loss = 0.30251654
Iteration 160, loss = 0.30215646
Iteration 161, loss = 0.30179302
Iteration 162, loss = 0.30143485
Iteration 163, loss = 0.30109387
Iteration 164, loss = 0.30076780
Iteration 165, loss = 0.30044541
Iteration 166, loss = 0.30012358
Iteration 167, loss = 0.29980146
Iteration 168, loss = 0.29948587
Iteration 169, loss = 0.29917571
Iteration 170, loss = 0.29886565
Iteration 171, loss = 0.29855540
Iteration 172, loss = 0.29827142
Iteration 173, loss = 0.29796720
Iteration 174, loss = 0.29768917
Iteration 175, loss = 0.29741175
Iteration 176, loss = 0.29714771
Iteration 177, loss = 0.29687901
Iteration 178, loss = 0.29662667
Iteration 179, loss = 0.29637409
```

```
Iteration 180, loss = 0.29613096
Iteration 181, loss = 0.29588563
Iteration 182, loss = 0.29565284
Iteration 183, loss = 0.29539494
Iteration 184, loss = 0.29516439
Iteration 185, loss = 0.29493037
Iteration 186, loss = 0.29470100
Iteration 187, loss = 0.29447903
Iteration 188, loss = 0.29425309
Iteration 189, loss = 0.29403270
Iteration 190, loss = 0.29382014
Iteration 191, loss = 0.29361791
Iteration 192, loss = 0.29341651
Iteration 193, loss = 0.29321232
Iteration 194, loss = 0.29300583
Iteration 195, loss = 0.29279636
Iteration 196, loss = 0.29259894
Iteration 197, loss = 0.29239880
Iteration 198, loss = 0.29219334
Iteration 199, loss = 0.29200993
Iteration 200, loss = 0.29180019
Iteration 201, loss = 0.29163172
Iteration 202, loss = 0.29144895
Iteration 203, loss = 0.29126280
Iteration 204, loss = 0.29109418
Iteration 205, loss = 0.29092701
Iteration 206, loss = 0.29073626
Iteration 207, loss = 0.29057256
Iteration 208, loss = 0.29040317
Iteration 209, loss = 0.29024611
Iteration 210, loss = 0.29008714
Iteration 211, loss = 0.28993761
Iteration 212, loss = 0.28977709
Iteration 213, loss = 0.28963712
Iteration 214, loss = 0.28948684
Iteration 215, loss = 0.28934462
Iteration 216, loss = 0.28920377
Iteration 217, loss = 0.28907014
Iteration 218, loss = 0.28892585
Iteration 219, loss = 0.28879242
Iteration 220, loss = 0.28865970
Iteration 221, loss = 0.28852650
Iteration 222, loss = 0.28840494
Iteration 223, loss = 0.28827708
Iteration 224, loss = 0.28815305
Iteration 225, loss = 0.28803946
Iteration 226, loss = 0.28791331
Iteration 227, loss = 0.28778997
Iteration 228, loss = 0.28767109
Iteration 229, loss = 0.28754482
Iteration 230, loss = 0.28743570
Iteration 231, loss = 0.28731147
Iteration 232, loss = 0.28720531
Iteration 233, loss = 0.28709125
Iteration 234, loss = 0.28699718
Iteration 235, loss = 0.28689029
Iteration 236, loss = 0.28677365
Iteration 237, loss = 0.28666891
Iteration 238, loss = 0.28656443
Iteration 239, loss = 0.28645819
```

```
Iteration 240, loss = 0.28635612
Iteration 241, loss = 0.28624162
Iteration 242, loss = 0.28614694
Iteration 243, loss = 0.28604320
Iteration 244, loss = 0.28594768
Iteration 245, loss = 0.28586022
Iteration 246, loss = 0.28577197
Iteration 247, loss = 0.28568948
Iteration 248, loss = 0.28559410
Iteration 249, loss = 0.28550561
Iteration 250, loss = 0.28541338
Iteration 251, loss = 0.28533350
Iteration 252, loss = 0.28524073
Iteration 253, loss = 0.28515500
Iteration 254, loss = 0.28507186
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Setting learning
rate to 0.000200
Iteration 255, loss = 0.28499817
Iteration 256, loss = 0.28495613
Iteration 257, loss = 0.28492560
Iteration 258, loss = 0.28490511
Iteration 259, loss = 0.28488500
Iteration 260, loss = 0.28486790
Iteration 261, loss = 0.28485129
Iteration 262, loss = 0.28483324
Iteration 263, loss = 0.28481692
Iteration 264, loss = 0.28479942
Iteration 265, loss = 0.28478285
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Setting learning
rate to 0.000040
Iteration 266, loss = 0.28476977
Iteration 267, loss = 0.28476048
Iteration 268, loss = 0.28475463
Iteration 269, loss = 0.28475018
Iteration 270, loss = 0.28474674
Iteration 271, loss = 0.28474372
Iteration 272, loss = 0.28474029
Iteration 273, loss = 0.28473698
Iteration 274, loss = 0.28473426
Iteration 275, loss = 0.28473073
Iteration 276, loss = 0.28472784
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Setting learning
rate to 0.000008
Iteration 277, loss = 0.28472506
Iteration 278, loss = 0.28472333
Iteration 279, loss = 0.28472235
Iteration 280, loss = 0.28472146
Iteration 281, loss = 0.28472082
Iteration 282, loss = 0.28472012
Iteration 283, loss = 0.28471952
Iteration 284, loss = 0.28471889
Iteration 285, loss = 0.28471819
Iteration 286, loss = 0.28471765
Iteration 287, loss = 0.28471700
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Setting learning
rate to 0.000002
Iteration 288, loss = 0.28471646
Iteration 289, loss = 0.28471612
Iteration 290, loss = 0.28471591
Iteration 291, loss = 0.28471576
Iteration 292, loss = 0.28471562
```

```
Iteration 293, loss = 0.28471547
Iteration 294, loss = 0.28471535
Iteration 295, loss = 0.28471521
Iteration 296, loss = 0.28471508
Iteration 297, loss = 0.28471497
Iteration 298, loss = 0.28471484
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Setting learning rate to 0.000000
Iteration 299, loss = 0.28471473
Iteration 300, loss = 0.28471468
Iteration 301, loss = 0.28471464
Iteration 302, loss = 0.28471461
Iteration 303, loss = 0.28471458
Iteration 304, loss = 0.28471455
Iteration 305, loss = 0.28471453
Iteration 306, loss = 0.28471450
Iteration 307, loss = 0.28471447
Iteration 308, loss = 0.28471445
Iteration 309, loss = 0.28471442
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Learning rate too small. Stopping.
Train set AUC score: 0.500000
Train set AUC score: 0.500000
```

Cross Validation: GUSTO Neural Net Model¶

#This section must run overnight - mlp GRIDSEARCH can take up to 8 hours. gmlp = MLPClassifier(max_iter=200) parameter_space = { 'hidden_layer_sizes': [(50,20,10,1),(50,10),(20,10,5), (100,80,10,1)], 'activation': ['relu'], 'solver': ['sgd','adam'], 'batch_size': [300,500], 'alpha': [0.10,0.40,0.60,0.80,1.0], 'random_state': [1], 'learning_rate': ['constant', 'adaptive'], 'learning_rate_init': [0.1, 0.01, 0.001] } gclf = GridSearchCV(gmlp, parameter_space, n_jobs=-1, cv=5) gclf.fit(g_Xtrain, g_Ytrain) print('Best parameters found:\n', gclf.best_params_)
In [39]:

```
#Calculate AUC after GridSearch

gmlp2 = MLPClassifier(
    hidden_layer_sizes=(50,10),
    activation='relu',
    max_iter=100,
    alpha=0.60,
    solver="sgd",
    verbose=10,
    random_state=1,
    learning_rate='constant',
    learning_rate_init=0.01,
    batch_size = 500
)


gmlp2.fit(g_Xtrain, g_Ytrain)

gmlp_predtrain2 = gmlp2.predict_proba(g_Xtrain)
print("Train set AUC score: %f" % roc_auc_score(g_Ytrain, gmlp_predtrain2[:,1]))

gmlp_predtest2 = gmlp2.predict_proba(g_Xtest)
print("Test set AUC score: %f" % roc_auc_score(g_Ytest, gmlp_predtest2[:,1]))
```

```
Iteration 1, loss = 0.63577219
```

```
Iteration 2, loss = 0.52894515
Iteration 3, loss = 0.41730600
Iteration 4, loss = 0.34414531
Iteration 5, loss = 0.31150205
Iteration 6, loss = 0.30537341
Iteration 7, loss = 0.30819080
Iteration 8, loss = 0.31104528
Iteration 9, loss = 0.31084716
Iteration 10, loss = 0.30713292
Iteration 11, loss = 0.30156299
Iteration 12, loss = 0.29602701
Iteration 13, loss = 0.29095907
Iteration 14, loss = 0.28674576
Iteration 15, loss = 0.28419370
Iteration 16, loss = 0.28181150
Iteration 17, loss = 0.27975224
Iteration 18, loss = 0.27803536
Iteration 19, loss = 0.27601248
Iteration 20, loss = 0.27407132
Iteration 21, loss = 0.27211341
Iteration 22, loss = 0.27020226
Iteration 23, loss = 0.26839671
Iteration 24, loss = 0.26666484
Iteration 25, loss = 0.26513046
Iteration 26, loss = 0.26352808
Iteration 27, loss = 0.26209841
Iteration 28, loss = 0.26054231
Iteration 29, loss = 0.25923292
Iteration 30, loss = 0.25790896
Iteration 31, loss = 0.25659911
Iteration 32, loss = 0.25547556
Iteration 33, loss = 0.25434827
Iteration 34, loss = 0.25320586
Iteration 35, loss = 0.25215007
Iteration 36, loss = 0.25099438
Iteration 37, loss = 0.25007309
Iteration 38, loss = 0.24911683
Iteration 39, loss = 0.24815018
Iteration 40, loss = 0.24733306
Iteration 41, loss = 0.24639008
Iteration 42, loss = 0.24558733
Iteration 43, loss = 0.24476332
Iteration 44, loss = 0.24407835
Iteration 45, loss = 0.24329060
Iteration 46, loss = 0.24260020
Iteration 47, loss = 0.24196701
Iteration 48, loss = 0.24125485
Iteration 49, loss = 0.24063624
Iteration 50, loss = 0.24001834
Iteration 51, loss = 0.23942460
Iteration 52, loss = 0.23886063
Iteration 53, loss = 0.23828201
Iteration 54, loss = 0.23779688
Iteration 55, loss = 0.23729721
Iteration 56, loss = 0.23680775
Iteration 57, loss = 0.23631233
Iteration 58, loss = 0.23586576
Iteration 59, loss = 0.23540495
Iteration 60, loss = 0.23505014
Iteration 61, loss = 0.23455160
Iteration 62, loss = 0.23427665
```

```
Iteration 63, loss = 0.23374486
Iteration 64, loss = 0.23340118
Iteration 65, loss = 0.23297374
Iteration 66, loss = 0.23275167
Iteration 67, loss = 0.23232059
Iteration 68, loss = 0.23198671
Iteration 69, loss = 0.23163674
Iteration 70, loss = 0.23133209
Iteration 71, loss = 0.23099957
Iteration 72, loss = 0.23070467
Iteration 73, loss = 0.23041528
Iteration 74, loss = 0.23019043
Iteration 75, loss = 0.22986945
Iteration 76, loss = 0.22959736
Iteration 77, loss = 0.22933801
Iteration 78, loss = 0.22906768
Iteration 79, loss = 0.22885613
Iteration 80, loss = 0.22863951
Iteration 81, loss = 0.22840956
Iteration 82, loss = 0.22817774
Iteration 83, loss = 0.22792960
Iteration 84, loss = 0.22770935
Iteration 85, loss = 0.22749430
Iteration 86, loss = 0.22727542
Iteration 87, loss = 0.22708445
Iteration 88, loss = 0.22688979
Iteration 89, loss = 0.22677113
Iteration 90, loss = 0.22650703
Iteration 91, loss = 0.22632788
Iteration 92, loss = 0.22612026
Iteration 93, loss = 0.22598322
Iteration 94, loss = 0.22577882
Iteration 95, loss = 0.22563013
Iteration 96, loss = 0.22544757
Iteration 97, loss = 0.22528276
Iteration 98, loss = 0.22507226
Iteration 99, loss = 0.22487751
Iteration 100, loss = 0.22477138
Train set AUC score: 0.816600
Test set AUC score: 0.805564
```

```
C:\Users\vitak\anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:692:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (100) reached and the optimization
hasn't converged yet.
  warnings.warn(
```

In [41]:

```
#Calculate Confidence Interval After Neural Net GridSearch GUSTO

y_true_g = np.array(g_Ytest)
y_pred_g = np.array(gmlp_predtest2[:,1])

n_bootstraps = 100
rng_seed = 42
bootstrapped_scores = []

rng = np.random.RandomState(rng_seed)
for i in range(n_bootstraps):
    indices = rng.randint(0, len(y_pred_g), len(y_pred_g))
```

```
    if len(np.unique(y_true_g[indices])) < 2:
        continue

    score = roc_auc_score(y_true_g[indices], y_pred_g[indices])
    bootstrapped_scores.append(score)

sorted_scores = np.array(bootstrapped_scores)
sorted_scores.sort()

confidence_lower_nng = sorted_scores[int(0.05 * len(sorted_scores))]
confidence_upper_nng = sorted_scores[int(0.95 * len(sorted_scores))]
print("Confidence interval for the score: [{:0.5f} - {:0.5}]".format(confidence_lower_nng,
confidence_upper_nng))
```

```
Confidence interval for the score: [0.77269 - 0.84376]
```