MScBMI 33200 – Assignment II¶

Savita K Gupta¶

24 April 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.neural_network import MLPClassifier

from sklearn.model_selection import train_test_split
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.model_selection import GridSearchCV

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

Section 1: ER Bots 30-Day Readmission Study¶

Question 1 - The Naive Model¶

In [4]:

```
# Import training dataset

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

In [5]:

```
# 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 [6]:

```
# Import testing dataset

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

In [7]:

```
#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 [48]:

```
# Run logistic regression

#naive_lr = LogisticRegression(penalty='l2', max_iter = 10000)
naive_model = LogisticRegression(penalty='none', max_iter = 10000)
naive_model.fit(rm_Xtrain_n, rm_Ytrain_n)
```

Out[48]:

```
LogisticRegression(max_iter=10000, penalty='none')
```

In [49]:

```
#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: 0.497019
```

In [19]:

```
# 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: [0.41544 - 0.55933]
```

Question 2 - The Logistic Regression Model¶

In [20]:

```
# 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 [21]:

```
#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 [53]:

```
# Run logistic regression

lr_model = LogisticRegressionCV(cv=5,penalty='l2', max_iter = 10000)

lr_model.fit(rm_Xtrain_lr, rm_Ytrain_lr)
```

Out[53]:

```
LogisticRegressionCV(cv=5, max_iter=10000)
```

In [54]:

```
#Calculate AUC using predict_proba

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

```
Logistic Regression train set AUC score (Logistic Regression - with predict proba): 0.543278
```

In [55]:

```
# 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: [0.46654 - 0.60726]
```

Section 2: GUSTO Study¶

Question 1 - Random Forest Model¶

In [35]:

```
# 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 [36]:

```
# Setup xTrain and yTrain

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

In [37]:

```
# Setup xTest and yTest

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

In [69]:

```
# Create random forest classifier model

g_rfClassifier = RandomForestClassifier(random_state=None, n_estimators=1000)

g_rfClassifier.fit(g_Xtrain,g_Ytrain)
```

Out[69]:

```
RandomForestClassifier(n_estimators=1000)
```

In [70]:

```
#Calculate AUC using predict_proba

g_rf_prediction = g_rfClassifier.predict_proba(g_Xtest)
print("Random Forest train set AUC score: %f" % roc_auc_score(g_Ytest, g_rf_prediction[:,1]))
```

```
Random Forest train set AUC score: 0.892280
```

In [71]:

```
# Calculate Confidence Interval using bootstrap for Random Forest Model

#Y train value for Random Forest model
y_true_rf = np.array(g_Ytest)

#Predict_proba value for Rain Forest Model model
y_pred_rf = np.array(g_rf_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_rf), len(y_pred_rf))
    if len(np.unique(y_true_rf[indices])) < 2:
        continue

    score = roc_auc_score(y_true_rf[indices], y_pred_rf[indices])
    bootstrapped_scores.append(score)

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

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

```
Confidence interval for the score: [0.86784 - 0.91657]
```

Question 2: Gradient Boosted Machine Model¶

In [42]:

```
# Create a GBM Model
```

```
g_gbm = GradientBoostingClassifier()

g_gbm.fit(g_Xtrain,g_Ytrain)
```

Out[42]:

```
GradientBoostingClassifier()
```

In [43]:

```
#Calculate AUC using predict_proba

g_gbm_prediction = g_gbm.predict_proba(g_Xtest)
print("Gradient Boosted Machine train set AUC score: %f" % roc_auc_score(g_Ytest,
g_gbm_prediction[:,1]))
```

```
Gradient Boosted Machine train set AUC score: 0.833949
```

In [44]:

```
# Calculate Confidence Interval using bootstrap for GBM Model

#Y train value for GBM model
y_true_gbm = np.array(g_Ytest)

#Predict_proba value for Rain Forest Model model
y_pred_gbm = np.array(g_gbm_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_gbm), len(y_pred_gbm))
    if len(np.unique(y_true_gbm[indices])) < 2:
        continue

    score = roc_auc_score(y_true_gbm[indices], y_pred_gbm[indices])
    bootstrapped_scores.append(score)

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

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

```
Confidence interval for the score: [0.80664 - 0.86494]
```

In [ ]: