**MScBMI 33200 – Machine Learning for Biomedical Informatics**
**Assignment IV**
**<Troy Zhongyi Zhang>**

Directions:
1. Fill out below information (tables and methods)
2. Submit this document along with your code in an HTML/PDF format

## Section 1: EMR Bots 30-day Readmission study

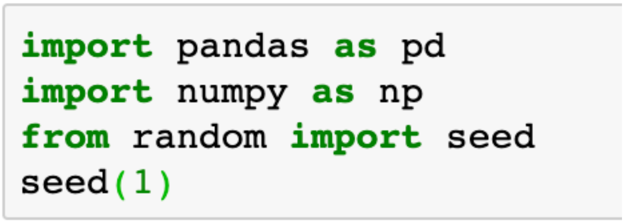
Using the training datasets, create the following models:

1. Naïve model: This model utilizes only patient characteristics (age, gender and race) to predict 30-day readmission in a logistic regression framework
2. GLM model : This model utilizes patient characteristics and most-recent lab recordings to predict 30-day admissions in a logistic regression framework.
3. ANN model: This model utilizes patient characteristics and most-recent lab recordings to predict 30-day admissions using an artificial neural network. Feature engineering steps include balancing classes using SMOTE as well as data normalization/standardization of continuous variables.

Utilize a five-fold cross-validation technique to build your model.
Calculate AUC on the test dataset. Fill out the following Table.

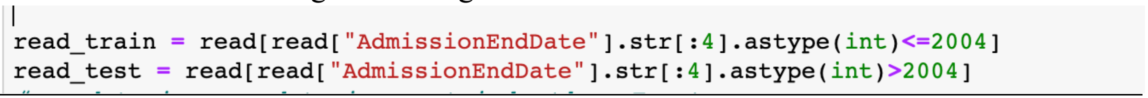
| | AUC (95% CI) |
|---|---|
| Naïve Model | 0.4909 (0.4105, 0.5714) |
| Logistic Regression | 0.4788 (0.3936, 0.5639) |
| ANN | 0.5052 (0.4313, 0.5791) |

Insert details (packages, parameter selection, etc.) on the models that were developed in the space given below.
Methods:



```
import pandas as pd
import numpy as np
from random import seed
seed(1)
```

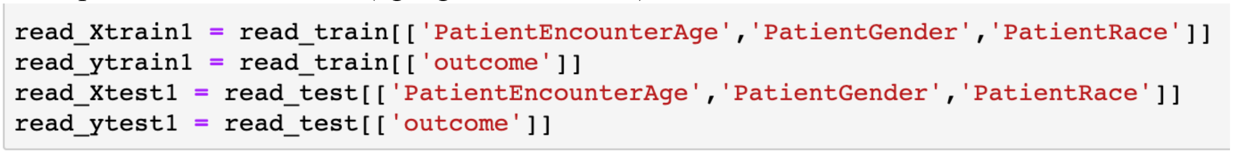
Firstly, import pandas as pd
import numpy as np
I separate the dataset into training and testing sets.



```
read_train = read[read["AdmissionEndDate"].str[:4].astype(int)<=2004]
read_test = read[read["AdmissionEndDate"].str[:4].astype(int)>2004]
```

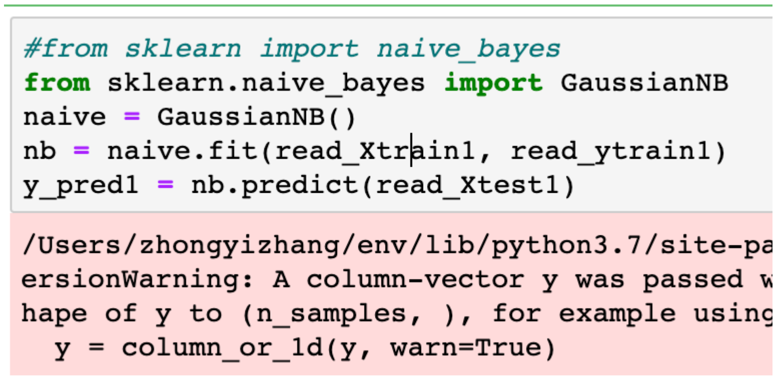
## Naïve Model:

I select patient characteristics (age, gender and race) as the dataset



```
read_Xtrain1 = read_train[['PatientEncounterAge','PatientGender','PatientRace']]
read_ytrain1 = read_train[['outcome']]
read_Xtest1 = read_test[['PatientEncounterAge','PatientGender','PatientRace']]
read_ytest1 = read_test[['outcome']]
```

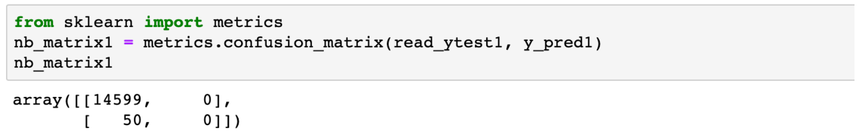
from sklearn.naive_bayes import GaussianNB to fit my model



```
#from sklearn import naive_bayes
from sklearn.naive_bayes import GaussianNB
naive = GaussianNB()
nb = naive.fit(read_Xtrain1, read_ytrain1)
y_pred1 = nb.predict(read_Xtest1)
```

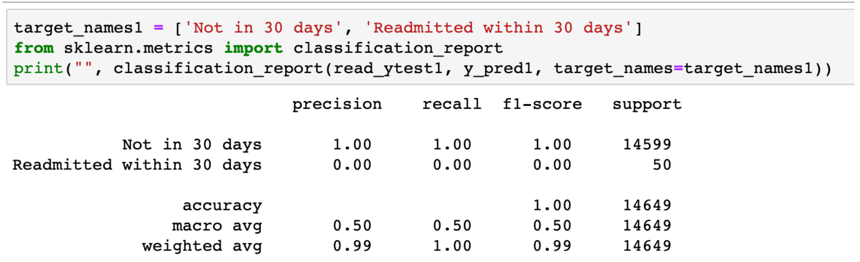
```
/Users/zhongyizhang/env/lib/python3.7/site-pa
ersionWarning: A column-vector y was passed w
hape of y to (n_samples, ), for example using
  y = column_or_1d(y, warn=True)
```

from sklearn import metrics for confusion matrix
from sklearn.metrics import classification_report



```
from sklearn import metrics
nb_matrix1 = metrics.confusion_matrix(read_ytest1, y_pred1)
nb_matrix1
```

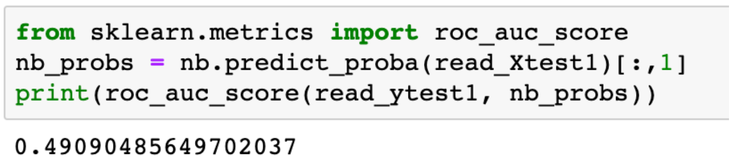
```
array([[14599,     0],
       [   50,     0]])
```



```
target_names1 = ['Not in 30 days', 'Readmitted within 30 days']
from sklearn.metrics import classification_report
print("", classification_report(read_ytest1, y_pred1, target_names=target_names1))
```

```
                           precision    recall  f1-score   support

           Not in 30 days       1.00      1.00      1.00     14599
Readmitted within 30 days       0.00      0.00      0.00        50

                 accuracy                           1.00     14649
                macro avg       0.50      0.50      0.50     14649
             weighted avg       0.99      1.00      0.99     14649
```

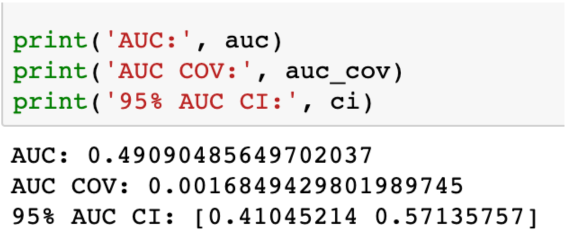
from sklearn.metrics import roc_auc_score to calculate 95% AUC



```
from sklearn.metrics import roc_auc_score
nb_probs = nb.predict_proba(read_Xtest1)[:,1]
print(roc_auc_score(read_ytest1, nb_probs))
```

```
0.49090485649702037
```

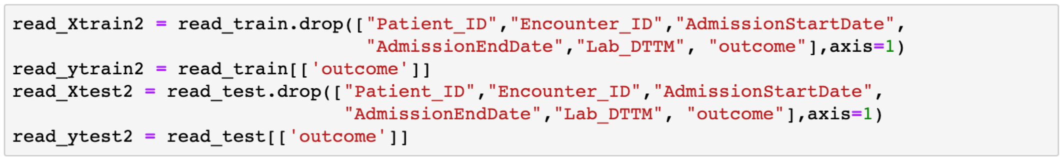
Since Python does not have the pROC package, I transplanted the DeLong function from R to calculate the confidence interval in Python
import numpy as np; import scipy.stats; from scipy import stats



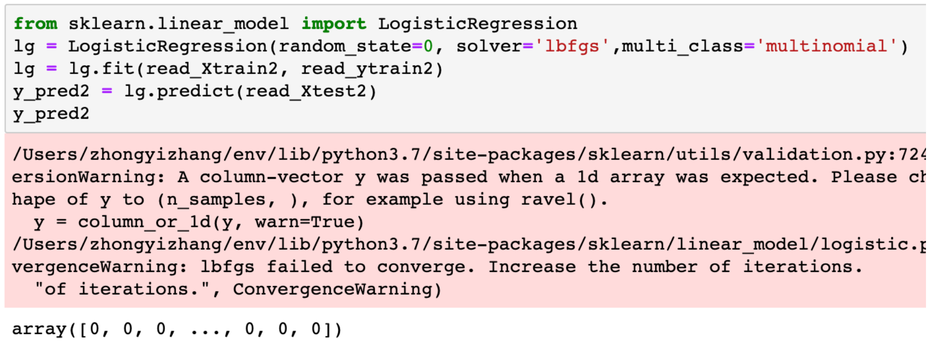
```
print('AUC:', auc)
print('AUC COV:', auc_cov)
print('95% AUC CI:', ci)
```

```
AUC: 0.49090485649702037
AUC COV: 0.0016849429801989745
95% AUC CI: [0.41045214 0.57135757]
```

## Logistic Regression:


```
read_Xtrain2 = read_train.drop(["Patient_ID","Encounter_ID","AdmissionStartDate",
                                 "AdmissionEndDate","Lab_DTTM", "outcome"],axis=1)
read_ytrain2 = read_train[['outcome']]
read_Xtest2 = read_test.drop(["Patient_ID","Encounter_ID","AdmissionStartDate",
                               "AdmissionEndDate","Lab_DTTM", "outcome"],axis=1)
read_ytest2 = read_test[['outcome']]
```

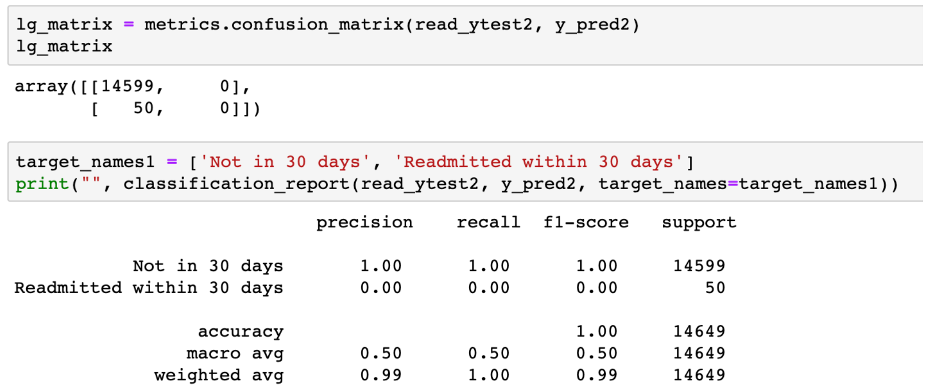
I choose the patient characteristics and latest lab observations as my dataset.
from sklearn.linear_model import LogisticRegression to fit a logistic regression model


```
from sklearn.linear_model import LogisticRegression
lg = LogisticRegression(random_state=0, solver='lbfgs',multi_class='multinomial')
lg = lg.fit(read_Xtrain2, read_ytrain2)
y_pred2 = lg.predict(read_Xtest2)
y_pred2
```

```
/Users/zhongyizhang/env/lib/python3.7/site-packages/sklearn/utils/validation.py:724
ersionWarning: A column-vector y was passed when a 1d array was expected. Please ch
hape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/Users/zhongyizhang/env/lib/python3.7/site-packages/sklearn/linear_model/logistic.p
vergenceWarning: lbfgs failed to converge. Increase the number of iterations.
  "of iterations.", ConvergenceWarning)
```

```
array([0, 0, 0, ..., 0, 0, 0])
```

Use the metrics package to get the confusion matrix

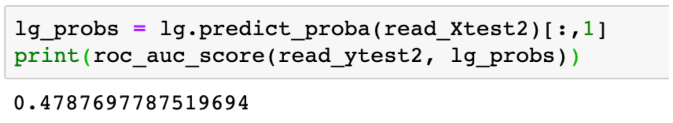

```
lg_matrix = metrics.confusion_matrix(read_ytest2, y_pred2)
lg_matrix
```

```
array([[14599,     0],
       [   50,     0]])
```

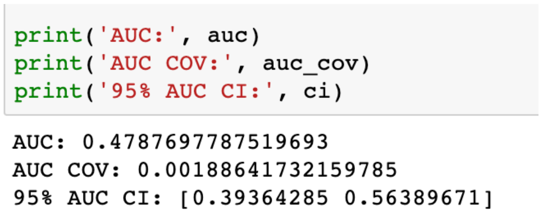
```
target_names1 = ['Not in 30 days', 'Readmitted within 30 days']
print("", classification_report(read_ytest2, y_pred2, target_names=target_names1))
```

```
                            precision    recall  f1-score   support

           Not in 30 days       1.00      1.00      1.00     14599
Readmitted within 30 days       0.00      0.00      0.00        50

                 accuracy                           1.00     14649
                macro avg       0.50      0.50      0.50     14649
             weighted avg       0.99      1.00      0.99     14649
```

Use the same function as above to find the AUC score and confidence interval:


```
lg_probs = lg.predict_proba(read_Xtest2)[:,1]
print(roc_auc_score(read_ytest2, lg_probs))
```

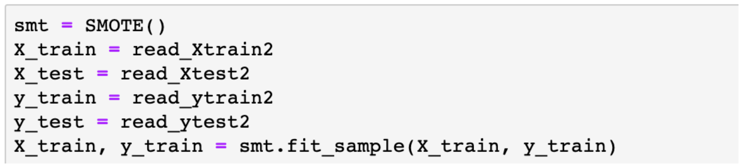
```
0.4787697787519694
```


```
print('AUC:', auc)
print('AUC COV:', auc_cov)
print('95% AUC CI:', ci)
```

```
AUC: 0.4787697787519693
AUC COV: 0.0018864173215978 5
95% AUC CI: [0.39364285 0.56389671]
```

## Artificial Neural Network:

from imblearn.over_sampling import SMOTE for using the SMOTE to balance the dataset.
Then I filled my training sets:


```
smt = SMOTE()
X_train = read_Xtrain2
X_test = read_Xtest2
y_train = read_ytrain2
y_test = read_ytest2
X_train, y_train = smt.fit_sample(X_train, y_train)
```

To test my new dataset after SMOTE is balanced: (Both outcome=1 and outcome=1 has the same number of rows. I upsampled the minority group to equal to the majority group)

```
y_train[y_train['outcome']==0].shape
```

```
(21416, 1)
```

```
y_train[y_train['outcome']==1].shape
```

```
(21416, 1)
```

from sklearn.preprocessing import StandardScaler to scale my data. I fitted my scaler with the training set and transferred the scaling to my testing set to avoid different variances.

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
# Fit only to the training data
scaler = scaler.fit(X_train)
X_trains = scaler.transform(X_train)
X_tests = scaler.transform(X_test)
```

from sklearn.neural_network import MLPClassifier to fit a neural network model.
mlp = MLPClassifier(solver='lbfgs', random_state=1), solver='lbfgs' is for back propagation, and then I set a fixed random state:

```
from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier(solver='lbfgs', random_state=1)
mlp = mlp.fit(X_trains, y_train)
ann_pred1 = mlp.predict(X_tests)
ann_pred1
```

```
/Users/zhongyizhang/env/lib/python3.7/site-packages/s
ron.py:921: DataConversionWarning: A column-vector y
Please change the shape of y to (n_samples, ), for ex
  y = column_or_1d(y, warn=True)
```

```
array([0, 0, 0, ..., 0, 0, 0])
```

I used metrics package to obtain the confusion matrix:

```
ann_matrix = metrics.confusion_matrix(y_test, ann_pred1)
ann_matrix
```

```
array([[14515,    84],
       [   50,     0]])
```

```
target_names1 = ['Not in 30 days', 'Readmitted within 30 days']
print("", classification_report(y_test, ann_pred1,
                                target_names=target_names1))
```

```
                           precision    recall  f1-score   support

           Not in 30 days       1.00      0.99      1.00     14599
Readmitted within 30 days       0.00      0.00      0.00        50

                 accuracy                           0.99     14649
                macro avg       0.50      0.50      0.50     14649
             weighted avg       0.99      0.99      0.99     14649
```

Then I used the same package to calculate the roc score and confidence interval:

```
ann_probs = mlp.predict_proba(X_tests)[:,1]
print(roc_auc_score(y_test, ann_probs))
```

```
0.505211315843551
```

```
print('AUC:', auc)
print('AUC COV:', auc_cov)
print('95% AUC CI:', ci)
```

```
AUC: 0.505211315843551
AUC COV: 0.0014218917463916226
95% AUC CI: [0.43130503 0.5791176 ]
```

## Section 2: Gusto study

Using the training datasets, create the following models:

1. GLM model : This model utilizes all features to predict 30-day mortality in a logistic regression framework.
2. Ridge Regression model : This model utilizes all features to predict 30-day mortality in a logistic regression framework with regularization.
3. ANN model: This model utilizes all features to predict 30-day mortality using an artificial neural network. Feature engineering steps should use normalization/standardization of continuous variables.

Utilize a five-fold cross-validation technique to build your model.
Calculate AUC on the test dataset. Fill out the following Table.

| | AUC (95% CI) |
|---|---|
| Logistic Regression | 0.8285 (0.7919, 0.8650) |
| Ridge Regression | 0.8280 (0.7912, 0.8647) |
| ANN | 0.7596 (0.7145, 0.8047) |

Insert details on the models that were developed in the space given below.
Methods:
Firstly, same as the above question, import pandas as pd, import numpy as np.

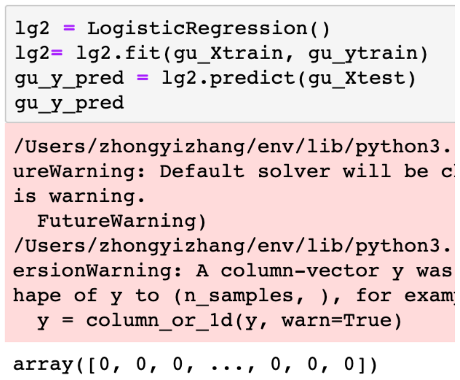
```python
import pandas as pd
import numpy as np
from random import seed
seed(1)
```

Then I separated the GUSTO dataset into training and testing datasets:

```python
gu_train = gusto.loc[(gusto['GROUP'] == 1
                     ) | (gusto['GROUP'] == 2
                         ) | (gusto['GROUP'] == 3)]
gu_test = gusto[gusto['GROUP']==0]

gu_Xtrain = gu_train.drop("DAY30",axis=1)
gu_ytrain = gu_train[['DAY30']]
gu_Xtest = gu_test.drop("DAY30",axis=1)
gu_ytest = gu_test[['DAY30']]
```

## Logistic Regression:

from sklearn.linear_model import LogisticRegression to fit a logistic regression model

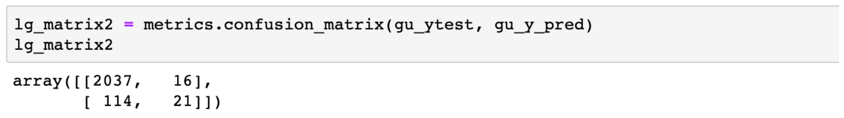


```
lg2 = LogisticRegression()
lg2= lg2.fit(gu_Xtrain, gu_ytrain)
gu_y_pred = lg2.predict(gu_Xtest)
gu_y_pred
```

```
/Users/zhongyizhang/env/lib/python3.
ureWarning: Default solver will be c
is warning.
  FutureWarning)
/Users/zhongyizhang/env/lib/python3.
ersionWarning: A column-vector y was
hape of y to (n_samples, ), for exam
  y = column_or_1d(y, warn=True)
```

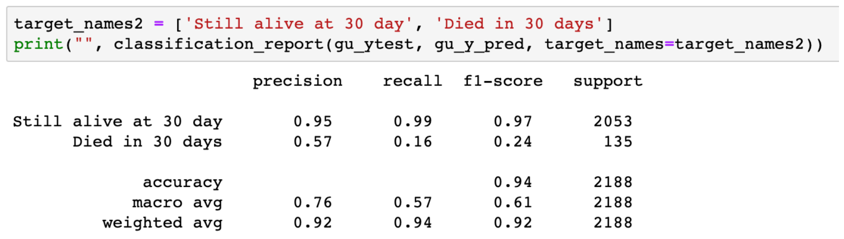
```
array([0, 0, 0, ..., 0, 0, 0])
```

Then I imported the metrics package to obtain the confusion matrix:



```
lg_matrix2 = metrics.confusion_matrix(gu_ytest, gu_y_pred)
lg_matrix2
```

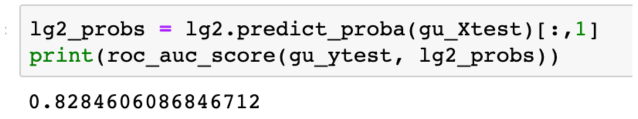
```
array([[2037,   16],
       [ 114,   21]])
```



```
target_names2 = ['Still alive at 30 day', 'Died in 30 days']
print("", classification_report(gu_ytest, gu_y_pred, target_names=target_names2))
```

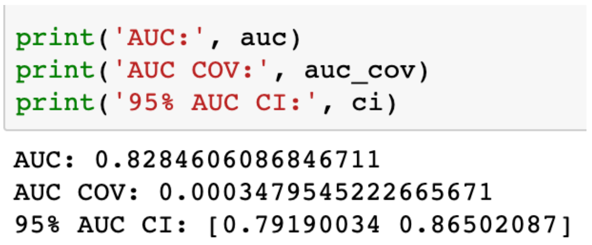
```
                        precision    recall  f1-score   support

Still alive at 30 day       0.95      0.99      0.97      2053
      Died in 30 days       0.57      0.16      0.24       135

             accuracy                           0.94      2188
            macro avg       0.76      0.57      0.61      2188
         weighted avg       0.92      0.94      0.92      2188
```

Use the above roc_auc_score package from Python to calculate the AUC and transplanted DeLong function from the pROC package in R to calculate the confidence interval:



```
lg2_probs = lg2.predict_proba(gu_Xtest)[:,1]
print(roc_auc_score(gu_ytest, lg2_probs))
```

```
0.8284606086846712
```

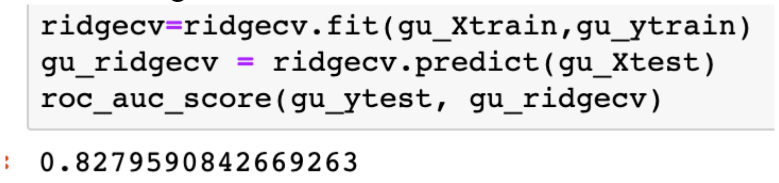


```
print('AUC:', auc)
print('AUC COV:', auc_cov)
print('95% AUC CI:', ci)
```

```
AUC: 0.8284606086846711
AUC COV: 0.0003479545222665671
95% AUC CI: [0.79190034 0.86502087]
```

## Ridge Regression:

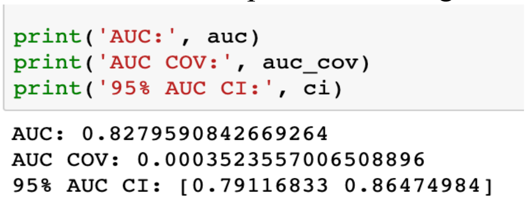
from sklearn.linear_model import RidgeCV to fit a ridge regression model and calculate the AUC score for Ridge with 5 cross validation:



```
ridgecv=ridgecv.fit(gu_Xtrain,gu_ytrain)
gu_ridgecv = ridgecv.predict(gu_Xtest)
roc_auc_score(gu_ytest, gu_ridgecv)
```

```
0.8279590842669263
```

Then I used the transplanted DeLong function to calculate the confidence interval:

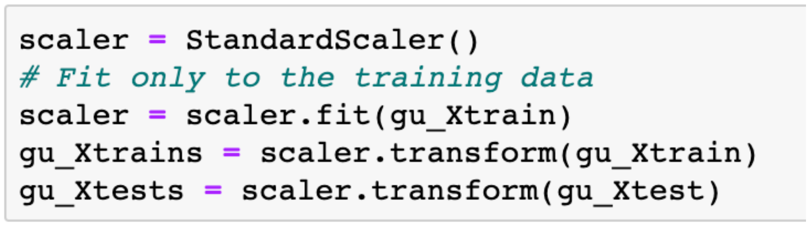


```
print('AUC:', auc)
print('AUC COV:', auc_cov)
print('95% AUC CI:', ci)
```

```
AUC: 0.8279590842669264
AUC COV: 0.0003523557006508896
95% AUC CI: [0.79116833 0.86474984]
```

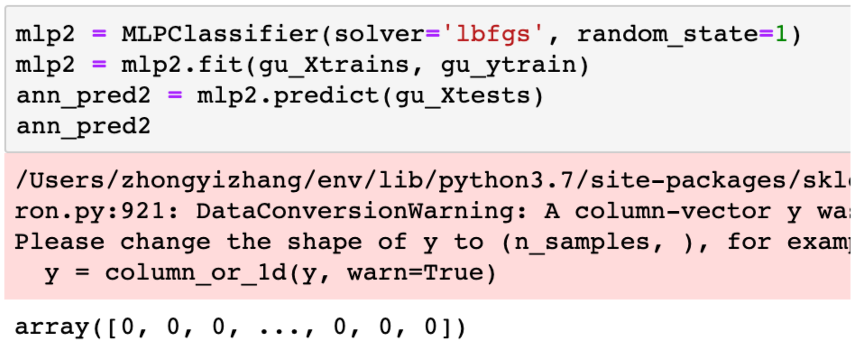
## Artificial Neural Network for GUSTO dataset:

from sklearn.preprocessing import StandardScaler to scale my data. I fitted my scaler with the training set and transferred the scaling to my testing set to avoid different variances.



```
scaler = StandardScaler()
# Fit only to the training data
scaler = scaler.fit(gu_Xtrain)
gu_Xtrains = scaler.transform(gu_Xtrain)
gu_Xtests = scaler.transform(gu_Xtest)
```

from sklearn.neural_network import MLPClassifier to fit a neural network model.
mlp = MLPClassifier(solver='lbfgs', random_state=1), solver='lbfgs' is for back propagation, and then I set a fixed random state:

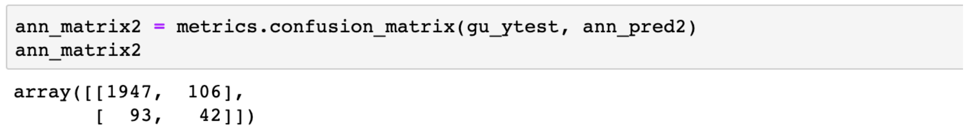


```
mlp2 = MLPClassifier(solver='lbfgs', random_state=1)
mlp2 = mlp2.fit(gu_Xtrains, gu_ytrain)
ann_pred2 = mlp2.predict(gu_Xtests)
ann_pred2
```

```
/Users/zhongyizhang/env/lib/python3.7/site-packages/skl
ron.py:921: DataConversionWarning: A column-vector y wa
Please change the shape of y to (n_samples, ), for exam
  y = column_or_1d(y, warn=True)
```

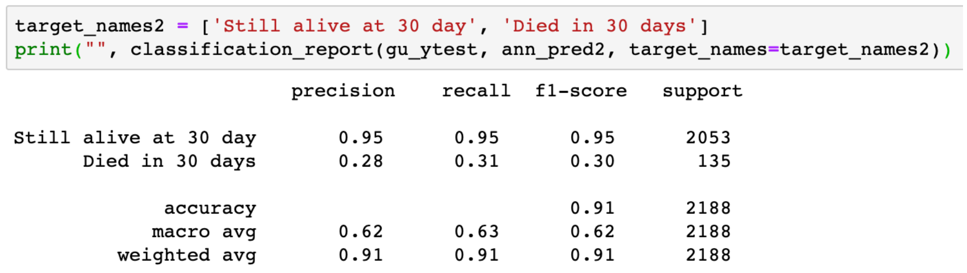
```
array([0, 0, 0, ..., 0, 0, 0])
```

With the metrics package from Python, I calculated the confusion matrix:



```
ann_matrix2 = metrics.confusion_matrix(gu_ytest, ann_pred2)
ann_matrix2
```

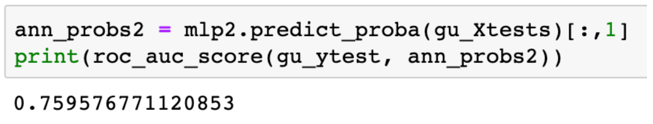
```
array([[1947,  106],
       [  93,   42]])
```



```
target_names2 = ['Still alive at 30 day', 'Died in 30 days']
print("", classification_report(gu_ytest, ann_pred2, target_names=target_names2))
```

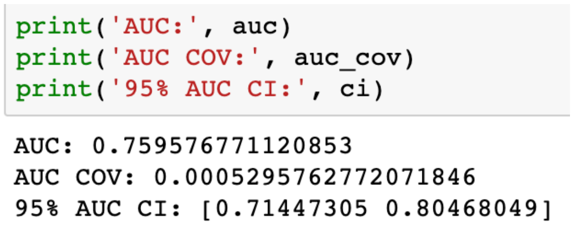
```
                        precision    recall  f1-score   support

Still alive at 30 day       0.95      0.95      0.95      2053
      Died in 30 days       0.28      0.31      0.30       135

             accuracy                           0.91      2188
            macro avg       0.62      0.63      0.62      2188
         weighted avg       0.91      0.91      0.91      2188
```

Using the same roc_auc_score package and DeLong function to calculate the AUC score and confidence interval of the ANN model for GUSTO dataset:



```
ann_probs2 = mlp2.predict_proba(gu_Xtests)[:,1]
print(roc_auc_score(gu_ytest, ann_probs2))
```

```
0.759576771120853
```



```
print('AUC:', auc)
print('AUC COV:', auc_cov)
print('95% AUC CI:', ci)
```

```
AUC: 0.759576771120853
AUC COV: 0.0005295762772071846
95% AUC CI: [0.71447305 0.80468049]
```

## Section 3: Short Answer Questions

Please answer the following questions briefly.

**a. Consider a prediction problem where the outcome is a categorical variable with 7 categories. You are asked to create a neural network. What is the size of the output layer and why?**

**Ans.** There will be 6 neurons in the output layer since there are 7 classification outcomes out. The size of output layer should always have C-1 neurons for classification. In our case, it will be 7-1=6 neurons in the output layer. All 6 neurons will output their probabilities for 6 classification outcomes, and the last classification outcome's probability will be 1 minus the sum of the previous 6 outcomes' probabilities. Therefore, there will be 6 neurons in the output layer.

**b. We created a neural network in class with a w_0 for each neuron. This is called as the bias term. What is the purpose of the bias term? (Feel free to search the net)**

**Ans.** Bias is somehow similar to the constant *b* of a linear function "*y = ax + b*". It allows me to move the line up and down to fit the prediction with the data better. Without *b* the line always goes through the origin (0, 0), and I may get a poorer fit. The bias neuron lies in one layer and is connected to all the neurons in the next layer, but it doesn't connect with the previous layer. It always emits 1. A layer in a neural network without a bias is nothing more than the multiplication of an input vector with a matrix. Using a bias, I am effectively adding another dimension to my input space, which always takes the value one, so I am avoiding an input vector of all zeros.

**c. Which algorithm is more useful when the data is too big to hold in memory? Gradient Descent or Stochastic Descent.**

**Ans.** In the standard gradient descent algorithm, the parameters are updated on the entire data set iteratively. On a large data set, standard gradient descent algorithm is not very efficient to find the global minimum of parameters (weights). Unlike the Gradient Descent algorithm, rather than waiting to sum up all the gradient terms (for all the m training data points) before we can calculate one change in parameters, Stochastic Descent computes the gradient term for just one single data point and starting to make progress to improve the parameters already. In Stochastic Descent, the change in parameters is going to be much faster, but each parameter's change is only trying to ensure that the particular predicted training data point fits better to the actual value.

**d. How is regularization performed in a neural network learning algorithm?**

**Ans.** Regularization techniques work by limiting the capacity of models—such as neural networks, linear regression, or logistic regression—by adding a parameter norm penalty $\Omega(\theta)$ to the objective function. Dropout is the one of the most interesting types of regularization techniques. At every iteration, it randomly selects some nodes and removes them along with all of their incoming and outgoing connections. Therefore, each iteration has a different set of nodes and this results in a different set of outputs. It can also be thought of as an ensemble technique in machine learning. Ensemble models usually perform better than a single model as they capture more randomness. Similarly, dropout also performs better than a normal neural network model.

Extra: Watch this video that illustrates how chain rule makes backpropagation work - https://www.youtube.com/watch?v=GlcnxUlrtek (Math alert!)