Assignments 2

October 21, 2019

1 MScBMI 33200 – Machine Learning for Biomedical Informatics

2 Assignment I

Name: Troy Zhongyi Zhang
Netid: zhongyiz@uchicago.edu

2.0.1 Directions:

1. Fill out below information (tables and methods)
2. Write as much detail as possible for the methods section for each problem. The idea is for you to have a record of all the steps you took for building each model.
3. Submit this document along with your code in an HTML/PDF format

2.0.2 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. Logistic Regression model : This model utilizes patient characteristics and most-recent lab recordings to predict 30-day admissions in a logistic regression framework.

[1]:
```
import pandas as pd
import numpy as np
```

[2]:
```
r_outcome = pd.read_csv("readmission_outcome.csv")
info = pd.read_csv("encounter_info.csv")
#df1 = pd.merge(info, r_outcome, on = "Encounter_ID")
labs = pd.read_csv("encounter_labs.csv")
```

[3]:
```
labs = labs.groupby(['Encounter_ID']).tail(1)
labs = labs.reset_index(drop=True)
wrong_ids = labs.iloc[:7]
wrong_ids['Encounter_ID'] = pd.DataFrame(wrong_ids['Encounter_ID'
                                                   ]).applymap(lambda x: x.
 ↪replace('1e+05', '100000'))
```

```
labs.head()
```

```
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-
packages/ipykernel_launcher.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/indexing.html#indexing-view-versus-copy
  """
```

[3]:

	Encounter_ID	Lab_DTTM	CBC..ABSOLUTE.LYMPHOCYTES	CBC..ABSOLUTE.NEUTROPHILS	CBC..BASOPHILS	CBC..EOSINOPHILS	CBC..HEMATOCRIT	CBC..HEMOGLOBIN	CBC..PLATELET.COUNT	CBC..RED.BLOOD.CELL.COUNT	CBC..WHITE.BLOOD.CELL.COUNT	METABOLIC..ALBUMIN	METABOLIC..BILI.TOTAL	METABOLIC..BUN	METABOLIC..CALCIUM	METABOLIC..CREATININE	METABOLIC..POTASSIUM	METABOLIC..SODIUM
0	100000_1	1981-01-08 23:28:24	32.8	76.4	0.0	0.1	35.9	14.8	386.0	4.8	10.7	5.8	0.8	12.6	11.1			
1	100000_2	1996-02-06 17:07:30	16.1	77.9	0.2	0.4	49.6	16.2	339.2	5.8	6.4	2.7	0.9	15.6	8.1			
2	100000_3	2002-04-11 00:36:27	23.4	66.2	0.1	0.4	40.4	18.7	408.7	3.2	8.5	3.7	0.3	15.1	7.9			
3	100000_4	2006-11-29 12:01:11	15.7	78.3	0.0	0.2	54.4	18.9	128.6	5.9	7.0	4.0	0.9	24.4	7.0			
4	100000_5	2007-04-27 16:49:40	27.4	62.2	0.1	0.4	45.2	16.8	315.4	4.9	11.8	5.9	0.1	21.4	7.5			

0	0.9	4.1	135.5
1	1.2	4.7	140.9
2	0.7	5.3	150.0
3	1.1	4.4	155.0
4	0.8	5.6	137.1

[4]:
```
read1 = pd.merge(info, labs, on = "Encounter_ID")
read = pd.merge(read1,r_outcome, on = "Encounter_ID")
read['PatientGender'] = read['PatientGender'].replace('Female',1)
read['PatientGender'] = read['PatientGender'].replace('Male',0)
read['PatientGender'] = read['PatientGender'].astype('category').cat.codes

read['PatientRace'] = read['PatientRace'].replace('African American',0)
read['PatientRace'] = read['PatientRace'].replace('White',1)
read['PatientRace'] = read['PatientRace'].replace('Asian',2)
read['PatientRace'] = read['PatientRace'].replace('Unknown',3)
read['PatientRace'] = read['PatientRace'].astype('category').cat.codes

read_train = read[read["AdmissionEndDate"].str[:4].astype(int)<=2004]
read_test = read[read["AdmissionEndDate"].str[:4].astype(int)>2004]
# read_train = read_train.reset_index(drop=True)
# read_test = read_test.reset_index(drop=True)

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

3 Naïve model:

3.0.1 readmission - read

[5]:
```
#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-
packages/sklearn/utils/validation.py:724: DataConversionWarning: A column-vector
y was passed when a 1d array was expected. Please change the shape of y to
(n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

[6]:
```
prob1 = nb.predict_proba(read_Xtest1)
prob1
```

[6]:
```
array([[0.99599489, 0.00400511],
       [0.99527026, 0.00472974],
       [0.99604199, 0.00395801],
       …,
       [0.98164369, 0.01835631],
       [0.99535436, 0.00464564],
       [0.99434525, 0.00565475]])
```

[7]:
```
from sklearn import metrics
nb_matrix1 = metrics.confusion_matrix(read_ytest1, y_pred1)
nb_matrix1
```

[7]:
```
array([[14599,     0],
       [   50,     0]])
```

[8]:
```
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


/Users/zhongyizhang/env/lib/python3.7/site-
packages/sklearn/metrics/classification.py:1437: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples.
  'precision', 'predicted', average, warn_for)
```

[9]:
```
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
```

[10]:
```
from sklearn.metrics import accuracy_score
as1 = accuracy_score(read_ytest1, y_pred1)
```

[11]:
```
as1
```

[11]:
```
0.9965867977336337
```

[12]:
```
error1 = 1-as1
error1
```

[12]:
```
0.003413202266366322
```

[13]:
```
n1 = len(y_pred1)
```

[14]:
```
import math
error1 + 1.96 * math.sqrt((error1 * (1 - error1)) / n1)
```

[14]:
```
0.004357677685114603
```

[15]:
```
error1 - 1.96 * math.sqrt((error1 * (1 - error1)) / n1)
```

[15]:
```
0.0024687268476180405
```

[16]:
```
import numpy as np, scipy.stats as st
st.t.interval(0.95, len(read_ytest1)-1, loc=np.mean(read_ytest1), scale=st.
 ↪sem(read_ytest1))
```

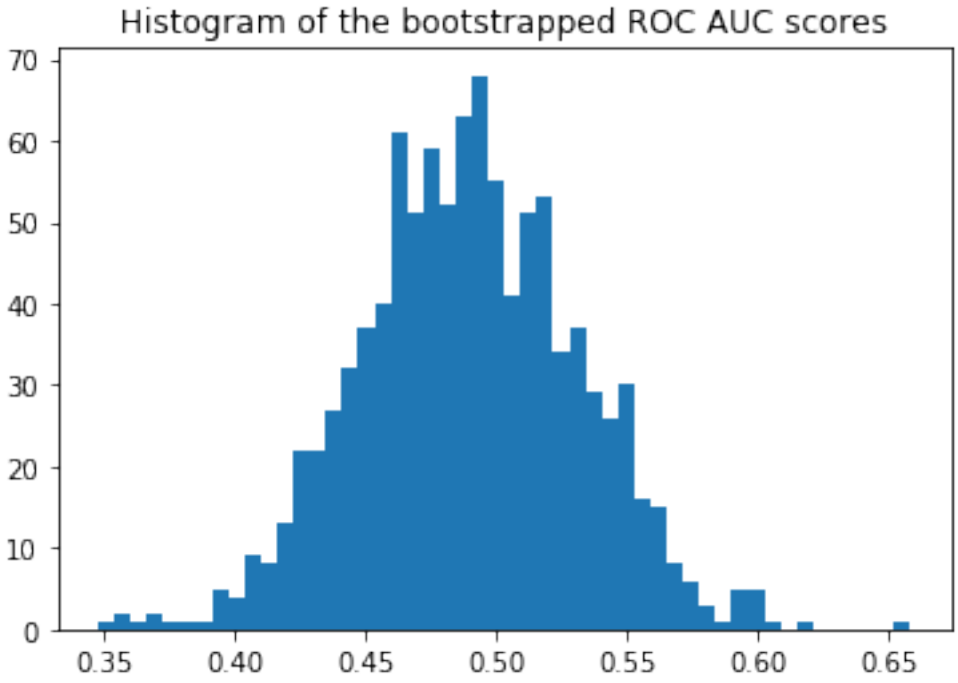
[16]:
```
(array([0.00246863]), array([0.00435777]))
```

[18]:
```
read_ytest1 = read_ytest1.values
```

[20]:
```
#Calculated the Confidence Interval by bootstrapping
import numpy as np
from scipy.stats import sem
from sklearn.metrics import roc_auc_score

y_pred = nb_probs
y_true = read_ytest1

print("Original ROC area: {:0.4f}".format(roc_auc_score(y_true, y_pred)))

n_bootstraps = 1000
rng_seed = 42  # control reproducibility
bootstrapped_scores = []

rng = np.random.RandomState(rng_seed)
for i in range(n_bootstraps):
    # bootstrap by sampling with replacement on the prediction indices
    indices = rng.randint(0, len(y_pred), len(y_pred))
    if len(np.unique(y_true[indices])) < 2:
```

```
        # We need at least one positive and one negative sample for ROC AUC
        # to be defined: reject the sample
        continue

    score = roc_auc_score(y_true[indices], y_pred[indices])
    bootstrapped_scores.append(score)
    #print("Bootstrap #{} ROC area: {:0.3f}".format(i + 1, score))

import matplotlib.pyplot as plt
plt.hist(bootstrapped_scores, bins=50)
plt.title('Histogram of the bootstrapped ROC AUC scores')
plt.show()

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

# Computing the lower and upper bound of the 90% confidence interval
# You can change the bounds percentiles to 0.025 and 0.975 to get
# a 95% confidence interval instead.
confidence_lower = sorted_scores[int(0.05 * len(sorted_scores))]
confidence_upper = sorted_scores[int(0.95 * len(sorted_scores))]
print("Confidence interval for the score: [{:0.4f} - {:0.4}]".format(
    confidence_lower, confidence_upper))
```

```
Original ROC area: 0.4909
```



```
Confidence interval for the score: [0.4232 - 0.5578]
```

[21]:
```python
#Transplanted the pROC package from R into Python for CI computation
import numpy as np
import scipy.stats
from scipy import stats

read_ytest1=read_ytest1.reshape((14649,))
# AUC comparison adapted from
# https://github.com/Netflix/vmaf/
def compute_midrank(x):
    """Computes midranks.
    Args:
       x - a 1D numpy array
    Returns:
       array of midranks
    """
    J = np.argsort(x)
    Z = x[J]
    N = len(x)
    T = np.zeros(N, dtype=np.float)
    i = 0
    while i < N:
        j = i
        while j < N and Z[j] == Z[i]:
            j += 1
        T[i:j] = 0.5*(i + j - 1)
        i = j
    T2 = np.empty(N, dtype=np.float)
    # Note(kazeevn) +1 is due to Python using 0-based indexing
    # instead of 1-based in the AUC formula in the paper
    T2[J] = T + 1
    return T2


def compute_midrank_weight(x, sample_weight):
    """Computes midranks.
    Args:
       x - a 1D numpy array
    Returns:
       array of midranks
    """
    J = np.argsort(x)
    Z = x[J]
    cumulative_weight = np.cumsum(sample_weight[J])
    N = len(x)
```

```
    T = np.zeros(N, dtype=np.float)
    i = 0
    while i < N:
        j = i
        while j < N and Z[j] == Z[i]:
            j += 1
        T[i:j] = cumulative_weight[i:j].mean()
        i = j
    T2 = np.empty(N, dtype=np.float)
    T2[J] = T
    return T2


def fastDeLong(predictions_sorted_transposed, label_1_count, sample_weight):
    if sample_weight is None:
        return fastDeLong_no_weights(predictions_sorted_transposed,␣
 ↪label_1_count)
    else:
        return fastDeLong_weights(predictions_sorted_transposed, label_1_count,␣
 ↪sample_weight)


def fastDeLong_weights(predictions_sorted_transposed, label_1_count,␣
 ↪sample_weight):
    """
    The fast version of DeLong's method for computing the covariance of
    unadjusted AUC.
    Args:
       predictions_sorted_transposed: a 2D numpy.array[n_classifiers,␣
 ↪n_examples]
          sorted such as the examples with label "1" are first
    Returns:
       (AUC value, DeLong covariance)
    Reference:
     @article{sun2014fast,
       title={Fast Implementation of DeLong's Algorithm for
              Comparing the Areas Under Correlated Receiver Oerating␣
 ↪Characteristic Curves},
       author={Xu Sun and Weichao Xu},
       journal={IEEE Signal Processing Letters},
       volume={21},
       number={11},
       pages={1389--1393},
       year={2014},
       publisher={IEEE}
     }
    """
```

```
    # Short variables are named as they are in the paper
    m = label_1_count
    n = predictions_sorted_transposed.shape[1] - m
    positive_examples = predictions_sorted_transposed[:, :m]
    negative_examples = predictions_sorted_transposed[:, m:]
    k = predictions_sorted_transposed.shape[0]

    tx = np.empty([k, m], dtype=np.float)
    ty = np.empty([k, n], dtype=np.float)
    tz = np.empty([k, m + n], dtype=np.float)
    for r in range(k):
        tx[r, :] = compute_midrank_weight(positive_examples[r, :],␣
↪sample_weight[:m])
        ty[r, :] = compute_midrank_weight(negative_examples[r, :],␣
↪sample_weight[m:])
        tz[r, :] = compute_midrank_weight(predictions_sorted_transposed[r, :],␣
↪sample_weight)
    total_positive_weights = sample_weight[:m].sum()
    total_negative_weights = sample_weight[m:].sum()
    pair_weights = np.dot(sample_weight[:m, np.newaxis], sample_weight[np.
↪newaxis, m:])
    total_pair_weights = pair_weights.sum()
    aucs = (sample_weight[:m]*(tz[:, :m] - tx)).sum(axis=1) / total_pair_weights
    v01 = (tz[:, :m] - tx[:, :]) / total_negative_weights
    v10 = 1. - (tz[:, m:] - ty[:, :]) / total_positive_weights
    sx = np.cov(v01)
    sy = np.cov(v10)
    delongcov = sx / m + sy / n
    return aucs, delongcov


def fastDeLong_no_weights(predictions_sorted_transposed, label_1_count):
    """
    The fast version of DeLong's method for computing the covariance of
    unadjusted AUC.
    Args:
       predictions_sorted_transposed: a 2D numpy.array[n_classifiers,␣
↪n_examples]
          sorted such as the examples with label "1" are first
    Returns:
       (AUC value, DeLong covariance)
    Reference:
     @article{sun2014fast,
       title={Fast Implementation of DeLong's Algorithm for
              Comparing the Areas Under Correlated Receiver Oerating
              Characteristic Curves},
       author={Xu Sun and Weichao Xu},
```

```
      journal={IEEE Signal Processing Letters},
      volume={21},
      number={11},
      pages={1389--1393},
      year={2014},
      publisher={IEEE}
    }
    """
    # Short variables are named as they are in the paper
    m = label_1_count
    n = predictions_sorted_transposed.shape[1] - m
    positive_examples = predictions_sorted_transposed[:, :m]
    negative_examples = predictions_sorted_transposed[:, m:]
    k = predictions_sorted_transposed.shape[0]

    tx = np.empty([k, m], dtype=np.float)
    ty = np.empty([k, n], dtype=np.float)
    tz = np.empty([k, m + n], dtype=np.float)
    for r in range(k):
        tx[r, :] = compute_midrank(positive_examples[r, :])
        ty[r, :] = compute_midrank(negative_examples[r, :])
        tz[r, :] = compute_midrank(predictions_sorted_transposed[r, :])
    aucs = tz[:, :m].sum(axis=1) / m / n - float(m + 1.0) / 2.0 / n
    v01 = (tz[:, :m] - tx[:, :]) / n
    v10 = 1.0 - (tz[:, m:] - ty[:, :]) / m
    sx = np.cov(v01)
    sy = np.cov(v10)
    delongcov = sx / m + sy / n
    return aucs, delongcov


def calc_pvalue(aucs, sigma):
    """Computes log(10) of p-values.
    Args:
      aucs: 1D array of AUCs
      sigma: AUC DeLong covariances
    Returns:
      log10(pvalue)
    """
    l = np.array([[1, -1]])
    z = np.abs(np.diff(aucs)) / np.sqrt(np.dot(np.dot(l, sigma), l.T))
    return np.log10(2) + scipy.stats.norm.logsf(z, loc=0, scale=1) / np.log(10)


def compute_ground_truth_statistics(ground_truth, sample_weight):
    assert np.array_equal(np.unique(ground_truth), [0, 1])
    order = (-ground_truth).argsort()
```

```
    label_1_count = int(ground_truth.sum())
    if sample_weight is None:
        ordered_sample_weight = None
    else:
        ordered_sample_weight = sample_weight[order]

    return order, label_1_count, ordered_sample_weight


def delong_roc_variance(ground_truth, predictions, sample_weight=None):
    """
    Computes ROC AUC variance for a single set of predictions
    Args:
       ground_truth: np.array of 0 and 1
       predictions: np.array of floats of the probability of being class 1
    """
    order, label_1_count, ordered_sample_weight =␣
↪compute_ground_truth_statistics(
        ground_truth, sample_weight)
    predictions_sorted_transposed = predictions[np.newaxis, order]
    aucs, delongcov = fastDeLong(predictions_sorted_transposed, label_1_count,␣
↪ordered_sample_weight)
    assert len(aucs) == 1, "There is a bug in the code, please forward this to␣
↪the developers"
    return aucs[0], delongcov


alpha = .95
y_pred = nb_probs
y_true = read_ytest1

auc, auc_cov = delong_roc_variance(
    y_true,
    y_pred)

auc_std = np.sqrt(auc_cov)
lower_upper_q = np.abs(np.array([0, 1]) - (1 - alpha) / 2)

ci = stats.norm.ppf(
    lower_upper_q,
    loc=auc,
    scale=auc_std)

ci[ci > 1] = 1

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]
```

4 Logistic Regression model

[22]:
```
read_Xtrain2 = read_train.
 ↪drop(["Patient_ID","Encounter_ID","AdmissionStartDate","AdmissionEndDate",
                                ␣
 ↪"PatientGender","PatientRace","PatientEncounterAge","Lab_DTTM", "outcome"],
                                axis=1)
read_ytrain2 = read_train[['outcome']]
read_Xtest2 = read_test.
 ↪drop(["Patient_ID","Encounter_ID","AdmissionStartDate","AdmissionEndDate",
                                ␣
 ↪"PatientGender","PatientRace","PatientEncounterAge","Lab_DTTM", "outcome"],
                                axis=1)
read_ytest2 = read_test[['outcome']]
```

[23]:
```
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: DataConversionWarning: A column-vector
y was passed when a 1d array was expected. Please change the shape 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.py:947: ConvergenceWarning: lbfgs failed
to converge. Increase the number of iterations.
  "of iterations.", ConvergenceWarning)
```

[23]:
```
array([0, 0, 0, …, 0, 0, 0])
```

[24]:
```
prob2 = lg.predict_proba(read_Xtest2)
prob2
```

[24]:
```
array([[0.99100683, 0.00899317],
       [0.99361524, 0.00638476],
```

```
       [0.99539944, 0.00460056],
       …,
       [0.99509458, 0.00490542],
       [0.9965151 , 0.0034849 ],
       [0.99569133, 0.00430867]])
```

[25]:
```
lg_matrix2 = metrics.confusion_matrix(read_ytest2, y_pred2)
lg_matrix2
```

[25]:
```
array([[14599,     0],
       [   50,     0]])
```

[26]:
```
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


/Users/zhongyizhang/env/lib/python3.7/site-
packages/sklearn/metrics/classification.py:1437: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples.
  'precision', 'predicted', average, warn_for)
```

[27]:
```
lg_probs = lg.predict_proba(read_Xtest2)[:,1]
print(roc_auc_score(read_ytest2, lg_probs))
```

```
0.4842674155764093
```

[28]:
```
import scipy.stats
def mean_confidence_interval(data, confidence=0.95):
    a = 1.0 * np.array(data)
    n = len(a)
    m, se = np.mean(a), scipy.stats.sem(a)
    h = se * scipy.stats.t.ppf((1 + confidence) / 2., n-1)
    return m, m-h, m+h
mean_confidence_interval(read_ytest2, confidence=0.95)
```

[28]:
```
(0.003413202266366305, array([0.00246863]), array([0.00435777]))
```

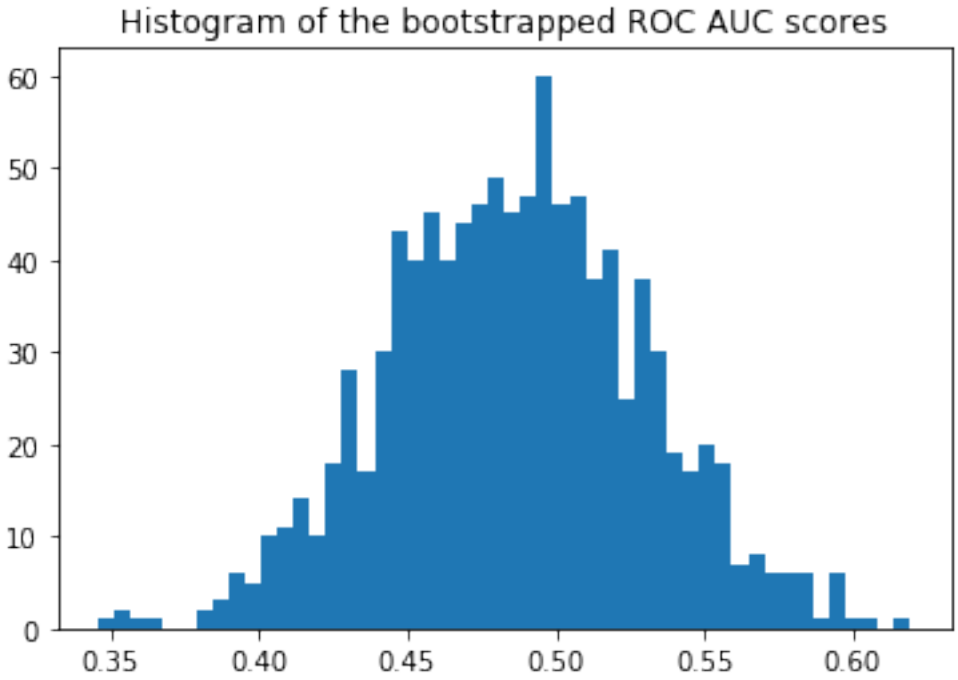
[29]:
```
import numpy as np, scipy.stats as st
st.t.interval(0.95, len(read_ytest2)-1, loc=np.mean(read_ytest2), scale=st.
 ↪sem(read_ytest2))
```

[29]:
```
(array([0.00246863]), array([0.00435777]))
```

[30]:
```
read_ytest2=read_ytest2.values
```

[31]:
```
#Boostrapping calculated 95% CI
y_pred = lg_probs
y_true = read_ytest2

print("Original ROC area: {:0.4f}".format(roc_auc_score(y_true, y_pred)))

n_bootstraps = 1000
rng_seed = 42  # control reproducibility
bootstrapped_scores = []

rng = np.random.RandomState(rng_seed)
for i in range(n_bootstraps):
    # bootstrap by sampling with replacement on the prediction indices
    indices = rng.randint(0, len(y_pred), len(y_pred))
    if len(np.unique(y_true[indices])) < 2:
        # We need at least one positive and one negative sample for ROC AUC
        # to be defined: reject the sample
        continue

    score = roc_auc_score(y_true[indices], y_pred[indices])
    bootstrapped_scores.append(score)
    #print("Bootstrap #{} ROC area: {:0.3f}".format(i + 1, score))

import matplotlib.pyplot as plt
plt.hist(bootstrapped_scores, bins=50)
plt.title('Histogram of the bootstrapped ROC AUC scores')
plt.show()

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

# Computing the lower and upper bound of the 90% confidence interval
# You can change the bounds percentiles to 0.025 and 0.975 to get
# a 95% confidence interval instead.
confidence_lower = sorted_scores[int(0.05 * len(sorted_scores))]
confidence_upper = sorted_scores[int(0.95 * len(sorted_scores))]
print("Confidence interval for the score: [{:0.4f} - {:0.4}]".format(
    confidence_lower, confidence_upper))
```

```
Original ROC area: 0.4843
```




```
Confidence interval for the score: [0.4147 - 0.5574]
```

[32]:
```
#pROC calculated 95% CI without bootstrapping
alpha = .95
read_ytest2=read_ytest2.reshape((14649,))
y_pred = lg_probs
y_true = read_ytest2

auc, auc_cov = delong_roc_variance(
    y_true,
    y_pred)

auc_std = np.sqrt(auc_cov)
lower_upper_q = np.abs(np.array([0, 1]) - (1 - alpha) / 2)

ci = stats.norm.ppf(
    lower_upper_q,
    loc=auc,
    scale=auc_std)

ci[ci > 1] = 1

print('AUC:', auc)
```

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

```
AUC: 0.4842674155764093
AUC COV: 0.0019394122536979157
95% AUC CI: [0.39795303 0.5705818 ]
```

4.0.1 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. Utilize a 5 fold cross validation to build the parameters for your model.

5 Gusto

[33]:
```
gusto = pd.read_csv("gusto_data.csv")
gusto['GROUP'] = gusto['GROUP'].replace('west',0)
gusto['GROUP'] = gusto['GROUP'].replace('sample2',1)
gusto['GROUP'] = gusto['GROUP'].replace('sample4',2)
gusto['GROUP'] = gusto['GROUP'].replace('sample5',3)
gusto['GROUP'] = gusto['GROUP'].astype('category').cat.codes
```

[34]:
```
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']]
```

[35]:
```
lg2 = LogisticRegression()
lg2= lg2.fit(gu_Xtrain, gu_ytrain)
gu_y_pred = lg2.predict(gu_Xtest)
gu_y_pred
```

```
/Users/zhongyizhang/env/lib/python3.7/site-
packages/sklearn/linear_model/logistic.py:432: FutureWarning: Default solver
will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
/Users/zhongyizhang/env/lib/python3.7/site-
packages/sklearn/utils/validation.py:724: DataConversionWarning: A column-vector
y was passed when a 1d array was expected. Please change the shape of y to
```

```
(n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

[35]:
```
array([0, 0, 0, …, 0, 0, 0])
```

[36]:
```
gu_y_pred.shape
```

[36]:
```
(2188,)
```

[37]:
```
prob3 = lg2.predict_proba(gu_Xtest)
prob3
```

[37]:
```
array([[0.90879031, 0.09120969],
       [0.96328343, 0.03671657],
       [0.96116893, 0.03883107],
       …,
       [0.9623824 , 0.0376176 ],
       [0.89408582, 0.10591418],
       [0.97521901, 0.02478099]])
```

[38]:
```
lg_matrix3 = metrics.confusion_matrix(gu_ytest, gu_y_pred)
lg_matrix3
```

[38]:
```
array([[2037,   16],
       [ 114,   21]])
```

[39]:
```
target_names3 = ['Still alive at 30 day', 'Died in 30 days']
print("", classification_report(gu_ytest, gu_y_pred,␣
 ↪target_names=target_names3))
```

```
                      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
```

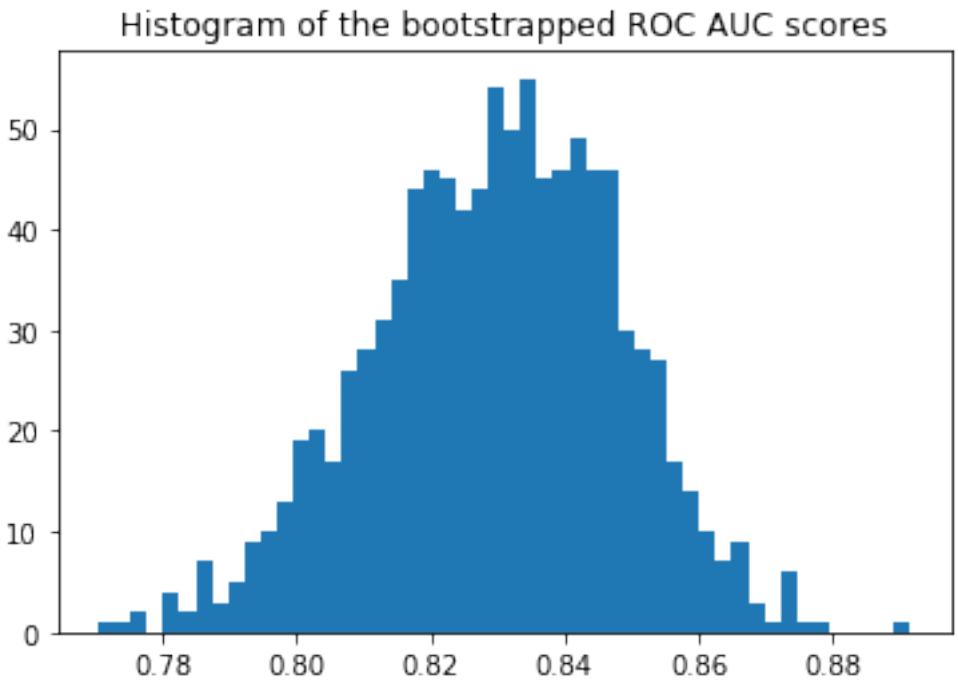
[40]:
```
lg2_probs = lg2.predict_proba(gu_Xtest)[:,1]
print(roc_auc_score(gu_ytest, lg2_probs))
```

```
0.8284606086846712
```

[41]:
```
gu_ytest=gu_ytest.values
```

[42]:
```
#Boostrapping calculated 95% CI
y_pred = lg2_probs
y_true = gu_ytest

print("Original ROC area: {:0.4f}".format(roc_auc_score(y_true, y_pred)))

n_bootstraps = 1000
rng_seed = 42  # control reproducibility
bootstrapped_scores = []

rng = np.random.RandomState(rng_seed)
for i in range(n_bootstraps):
    # bootstrap by sampling with replacement on the prediction indices
    indices = rng.randint(0, len(y_pred), len(y_pred))
    if len(np.unique(y_true[indices])) < 2:
        # We need at least one positive and one negative sample for ROC AUC
        # to be defined: reject the sample
        continue

    score = roc_auc_score(y_true[indices], y_pred[indices])
    bootstrapped_scores.append(score)
    #print("Bootstrap #{} ROC area: {:0.3f}".format(i + 1, score))

import matplotlib.pyplot as plt
plt.hist(bootstrapped_scores, bins=50)
plt.title('Histogram of the bootstrapped ROC AUC scores')
plt.show()

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

# Computing the lower and upper bound of the 90% confidence interval
# You can change the bounds percentiles to 0.025 and 0.975 to get
# a 95% confidence interval instead.
confidence_lower = sorted_scores[int(0.05 * len(sorted_scores))]
confidence_upper = sorted_scores[int(0.95 * len(sorted_scores))]
print("Confidence interval for the score: [{:0.4f} - {:0.4}]".format(
    confidence_lower, confidence_upper))
```

```
Original ROC area: 0.8285
```




```
Confidence interval for the score: [0.7985 - 0.8585]
```

[43]:
```
#pROC calculated 95% CI without bootstrapping
alpha = .95
gu_ytest = gu_ytest.reshape((2188,))
y_pred = lg2_probs
y_true = gu_ytest

auc, auc_cov = delong_roc_variance(
    y_true,
    y_pred)

auc_std = np.sqrt(auc_cov)
lower_upper_q = np.abs(np.array([0, 1]) - (1 - alpha) / 2)

ci = stats.norm.ppf(
    lower_upper_q,
    loc=auc,
    scale=auc_std)

ci[ci > 1] = 1

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]
```

[44]:
```
from sklearn.linear_model import Ridge
ridge_model = Ridge(alpha = 1, solver='cholesky')
# ridge_model = RidgeClassifier(alpha = 1)

ridge_model.fit(gu_Xtrain, gu_ytrain)
gu_regul = ridge_model.predict(gu_Xtest)
auc = roc_auc_score(gu_ytest, gu_regul)
auc
```

[44]:
```
0.8271436560769245
```

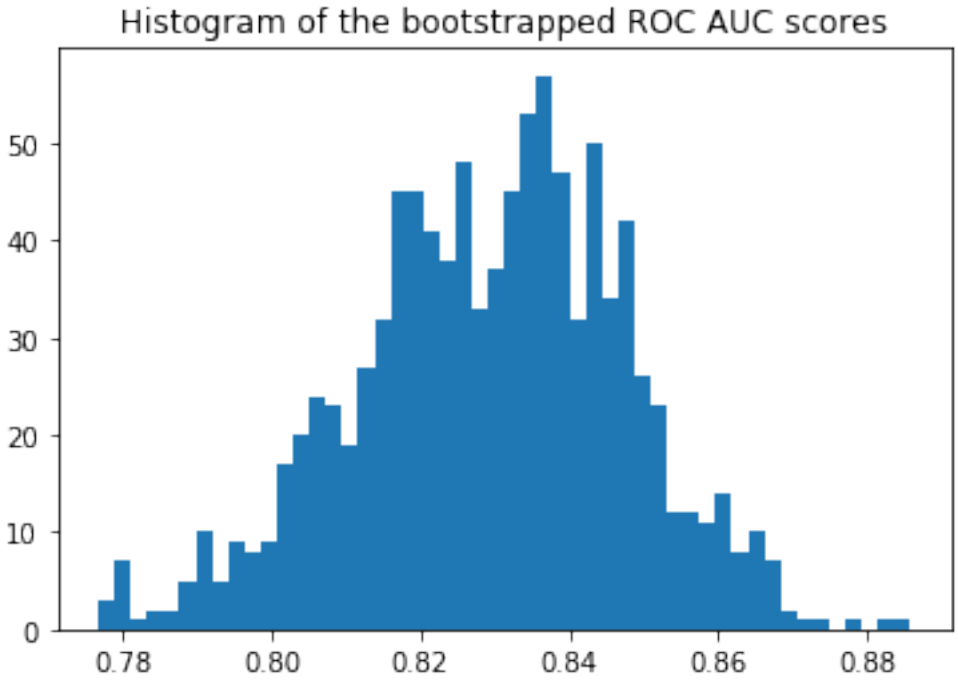
[45]:
```
from sklearn.linear_model import RidgeCV
ridgecv = RidgeCV(alphas=[1e-3, 1e-2, 1e-1, 1, 10], cv=5, fit_intercept=True,␣
↪scoring=None, normalize=False)
ridgecv=ridgecv.fit(gu_Xtrain,gu_ytrain)
gu_ridgecv = ridgecv.predict(gu_Xtest)
roc_auc_score(gu_ytest, gu_ridgecv)
```

[45]:
```
0.8279590842669263
```

[46]:
```
#Boostrapping calculated 95% CI
y_pred = gu_ridgecv
y_true = gu_ytest

print("Original ROC area: {:0.4f}".format(roc_auc_score(y_true, y_pred)))

n_bootstraps = 1000
rng_seed = 42  # control reproducibility
bootstrapped_scores = []

rng = np.random.RandomState(rng_seed)
for i in range(n_bootstraps):
    # bootstrap by sampling with replacement on the prediction indices
    indices = rng.randint(0, len(y_pred), len(y_pred))
    if len(np.unique(y_true[indices])) < 2:
        # We need at least one positive and one negative sample for ROC AUC
        # to be defined: reject the sample
        continue

    score = roc_auc_score(y_true[indices], y_pred[indices])
    bootstrapped_scores.append(score)
    #print("Bootstrap #{} ROC area: {:0.3f}".format(i + 1, score))
```

```
import matplotlib.pyplot as plt
plt.hist(bootstrapped_scores, bins=50)
plt.title('Histogram of the bootstrapped ROC AUC scores')
plt.show()

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

# Computing the lower and upper bound of the 90% confidence interval
# You can change the bounds percentiles to 0.025 and 0.975 to get
# a 95% confidence interval instead.
confidence_lower = sorted_scores[int(0.05 * len(sorted_scores))]
confidence_upper = sorted_scores[int(0.95 * len(sorted_scores))]
print("Confidence interval for the score: [{:0.4f} - {:0.4}]".format(
    confidence_lower, confidence_upper))
```

```
Original ROC area: 0.8280
```




```
Confidence interval for the score: [0.7982 - 0.8586]
```

[47]:
```
#pROC calculated 95% CI without bootstrapping
alpha = .95
gu_ridgecv = gu_ridgecv.reshape((2188,))
y_pred = gu_ridgecv
y_true = gu_ytest
```

```
auc, auc_cov = delong_roc_variance(
    y_true,
    y_pred)

auc_std = np.sqrt(auc_cov)
lower_upper_q = np.abs(np.array([0, 1]) - (1 - alpha) / 2)

ci = stats.norm.ppf(
    lower_upper_q,
    loc=auc,
    scale=auc_std)

ci[ci > 1] = 1

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]
```

[ ]:

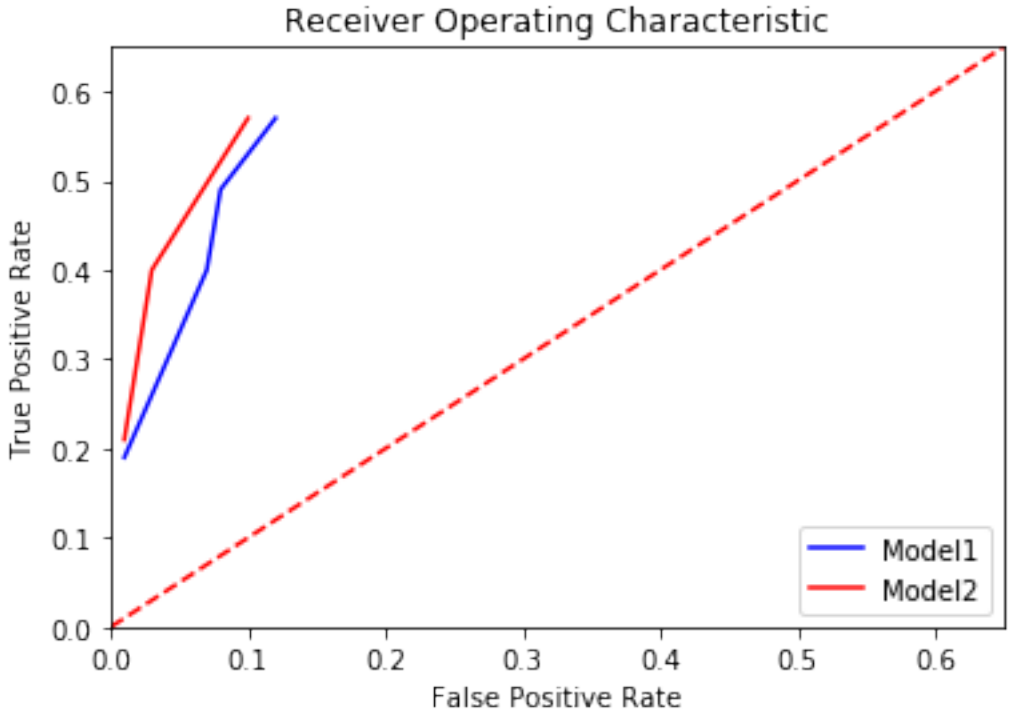
---

6 Section 3: Short Answer Questions

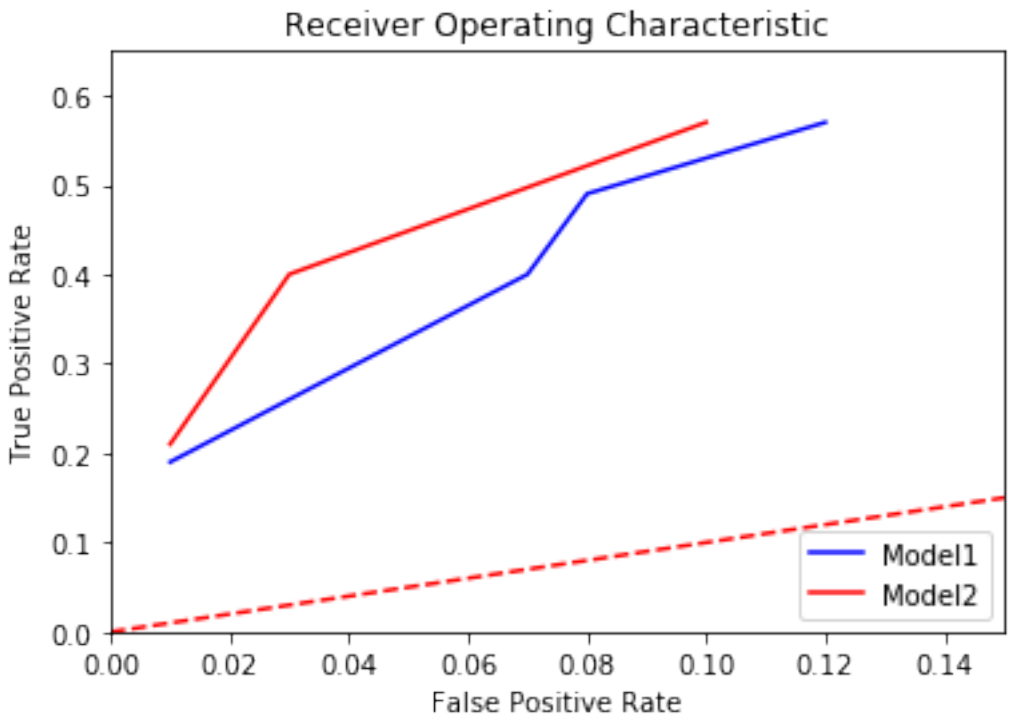
6.0.1 (d)

[59]:
```
import sklearn.metrics as metrics
roc_auc = metrics.auc(fpr, tpr)
tpr = [0.57, 0.53, 0.49, 0.4, 0.19]
fpr = [0.12, 0.1, 0.08, 0.07, 0.01]
tpr2 = [0.57, 0.4, 0.21]
fpr2 = [0.1,0.03, 0.01]

import matplotlib.pyplot as plt
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'Model1' % roc_auc)
plt.plot(fpr2, tpr2, 'r', label = 'Model2' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 0.65])
plt.ylim([0, 0.65])
```

```
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



[60]:
```
import sklearn.metrics as metrics
roc_auc = metrics.auc(fpr, tpr)
tpr = [0.57, 0.53, 0.49, 0.4, 0.19]
fpr = [0.12, 0.1, 0.08, 0.07, 0.01]
tpr2 = [0.57, 0.4, 0.21]
fpr2 = [0.1,0.03, 0.01]

import matplotlib.pyplot as plt
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'Model1' % roc_auc)
plt.plot(fpr2, tpr2, 'r', label = 'Model2' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 0.15])
plt.ylim([0, 0.65])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```


Receiver Operating Characteristic
True Positive Rate
False Positive Rate
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
Model1
Model2

[ ]:

[ ]:

[ ]: