Assignments 5

December 1, 2019

1 MScBMI 33200 – Machine Learning for Biomedical Informatics

2 Assignment V

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

2.0.1 Directions:

1. Fill out below information (tables and methods)

2. Submit this document along with your code in an HTML/PDF format

2.0.2 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.
4. Random Forest model: This model utilizes all features to predict 30-day mortality using a random forest.
5. GBM model: This model utilizes all features to predict 30-day mortality using a gradient boosted machine.
6. SVM model: This model utilizes all features to predict 30-day mortality using a support vector machine.

3 Gusto

[1]:
```
import pandas as pd
import numpy as np
from random import seed
from sklearn.model_selection import GridSearchCV, StratifiedKFold,␣
↪train_test_split
```

[2]:
```
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
```

[3]:
```
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']]
```

4 GLM Model

[4]:
```
from sklearn.linear_model import LogisticRegression
seed(0)

params6 = {'tol' : [1e-6,1e-5,1e-4,1e-3,1e-2],
           'C': [0.5,1.0,1.5,2.0,2.5]}
lg2 = LogisticRegression(random_state=0, solver='warn',multi_class='warn')
lg2 = GridSearchCV(lg2, cv=5, param_grid=params6, scoring = 'roc_auc',refit =␣
↪True,
                   n_jobs=-1, verbose = 5, return_train_score=True)

lg2.fit(gu_Xtrain, gu_ytrain)
lg2.cv_results_
```

```
Fitting 5 folds for each of 25 candidates, totalling 125 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  48 tasks      | elapsed:    2.3s
[Parallel(n_jobs=-1)]: Done 102 out of 125 | elapsed:    2.4s remaining:    0.5s
[Parallel(n_jobs=-1)]: Done 125 out of 125 | elapsed:    2.4s finished
```

```
/usr/local/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)
/usr/local/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)
```

[4]:

```
{'mean_fit_time': array([0.01549258, 0.01680298, 0.0150424 , 0.01326494,
0.01069741,
        0.01661963, 0.01575599, 0.01225686, 0.01076236, 0.00720906,
        0.01093097, 0.00964594, 0.00972013, 0.00796862, 0.00657158,
        0.01530442, 0.01240044, 0.01265583, 0.01129794, 0.00876784,
        0.01455798, 0.01288638, 0.01217575, 0.01026893, 0.00797677]),
 'std_fit_time': array([0.00115231, 0.00119659, 0.00132425, 0.00050567,
0.00161596,
        0.00121617, 0.00055153, 0.00153288, 0.00135912, 0.00121973,
        0.0009199 , 0.0006528 , 0.00173847, 0.00090499, 0.00108703,
        0.00330497, 0.0005184 , 0.00250276, 0.00089983, 0.00115839,
        0.00041526, 0.00091637, 0.00040188, 0.00088579, 0.00106507]),
 'mean_score_time': array([0.00408983, 0.00572977, 0.00534897, 0.00464754,
0.00414557,
        0.00443459, 0.00421901, 0.00333691, 0.00309634, 0.00283136,
        0.0026135 , 0.00253634, 0.00232868, 0.0024735 , 0.00302091,
        0.00366516, 0.00370235, 0.00417194, 0.00349503, 0.00354609,
        0.00336518, 0.00322495, 0.00360041, 0.00309124, 0.00269351]),
 'std_score_time': array([2.71781452e-04, 1.62394445e-03, 9.39529853e-04,
5.04044155e-04,
        3.57610429e-04, 2.62963134e-04, 4.91218779e-04, 6.44983890e-04,
        5.69342688e-04, 2.35531522e-04, 3.67651233e-04, 5.24449201e-04,
        6.43647860e-05, 2.39373897e-04, 6.55439440e-04, 1.60827313e-04,
        9.63255997e-04, 1.02442156e-03, 1.60160537e-04, 1.50101915e-04,
        3.84713998e-04, 4.38880428e-04, 1.79079774e-04, 6.23171376e-04,
        6.67047475e-04]),
 'param_C': masked_array(data=[0.5, 0.5, 0.5, 0.5, 0.5, 1.0, 1.0, 1.0, 1.0, 1.0,
1.5,
                   1.5, 1.5, 1.5, 1.5, 2.0, 2.0, 2.0, 2.0, 2.0, 2.5, 2.5,
                   2.5, 2.5, 2.5],
             mask=[False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False],
        fill_value='?',
             dtype=object),
 'param_tol': masked_array(data=[1e-06, 1e-05, 0.0001, 0.001, 0.01, 1e-06,
```

```
1e-05,
                   0.0001, 0.001, 0.01, 1e-06, 1e-05, 0.0001, 0.001, 0.01,
                   1e-06, 1e-05, 0.0001, 0.001, 0.01, 1e-06, 1e-05,
                   0.0001, 0.001, 0.01],
             mask=[False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False],
       fill_value='?',
            dtype=object),
 'params': [{'C': 0.5, 'tol': 1e-06},
  {'C': 0.5, 'tol': 1e-05},
  {'C': 0.5, 'tol': 0.0001},
  {'C': 0.5, 'tol': 0.001},
  {'C': 0.5, 'tol': 0.01},
  {'C': 1.0, 'tol': 1e-06},
  {'C': 1.0, 'tol': 1e-05},
  {'C': 1.0, 'tol': 0.0001},
  {'C': 1.0, 'tol': 0.001},
  {'C': 1.0, 'tol': 0.01},
  {'C': 1.5, 'tol': 1e-06},
  {'C': 1.5, 'tol': 1e-05},
  {'C': 1.5, 'tol': 0.0001},
  {'C': 1.5, 'tol': 0.001},
  {'C': 1.5, 'tol': 0.01},
  {'C': 2.0, 'tol': 1e-06},
  {'C': 2.0, 'tol': 1e-05},
  {'C': 2.0, 'tol': 0.0001},
  {'C': 2.0, 'tol': 0.001},
  {'C': 2.0, 'tol': 0.01},
  {'C': 2.5, 'tol': 1e-06},
  {'C': 2.5, 'tol': 1e-05},
  {'C': 2.5, 'tol': 0.0001},
  {'C': 2.5, 'tol': 0.001},
  {'C': 2.5, 'tol': 0.01}],
 'split0_test_score': array([0.77952899, 0.77952899, 0.77971014, 0.77663043,
0.775     ,
       0.77608696, 0.77608696, 0.77644928, 0.77300725, 0.77536232,
       0.77572464, 0.77572464, 0.77572464, 0.77192029, 0.77518116,
       0.77463768, 0.77463768, 0.77518116, 0.77264493, 0.77518116,
       0.77445652, 0.77445652, 0.77427536, 0.77463768, 0.77518116]),
 'split1_test_score': array([0.78737605, 0.78756674, 0.78737605, 0.78718535,
0.77517162,
       0.78623188, 0.78623188, 0.78527841, 0.78699466, 0.77459954,
       0.7858505 , 0.7858505 , 0.78546911, 0.78470633, 0.77345538,
       0.78546911, 0.78508772, 0.78546911, 0.78318078, 0.77402746,
       0.78508772, 0.78508772, 0.78451564, 0.78413425, 0.77364607]),
```

```
 'split2_test_score': array([0.80076555, 0.80038278, 0.80114833, 0.79425837,
0.80344498,
        0.79961722, 0.79961722, 0.79923445, 0.79923445, 0.80842105,
        0.79885167, 0.79885167, 0.7984689 , 0.79980861, 0.80937799,
        0.79827751, 0.79866029, 0.79866029, 0.79885167, 0.80861244,
        0.79732057, 0.79732057, 0.79751196, 0.78985646, 0.80842105]),
 'split3_test_score': array([0.77665072, 0.77684211, 0.77722488, 0.77626794,
0.76956938,
        0.7737799 , 0.7737799 , 0.7737799 , 0.77492823, 0.784689  ,
        0.77301435, 0.77301435, 0.77320574, 0.77358852, 0.78602871,
        0.77263158, 0.77263158, 0.77358852, 0.77186603, 0.78717703,
        0.7722488 , 0.77244019, 0.77358852, 0.77110048, 0.78717703]),
 'split4_test_score': array([0.83253589, 0.8323445 , 0.8323445 , 0.82794258,
0.68382775,
        0.8323445 , 0.83215311, 0.83291866, 0.82698565, 0.68382775,
        0.83291866, 0.83291866, 0.83291866, 0.83406699, 0.68382775,
        0.8323445 , 0.8323445 , 0.83215311, 0.83425837, 0.68382775,
        0.83311005, 0.83311005, 0.83291866, 0.83157895, 0.68382775]),
 'mean_test_score': array([0.7953445 , 0.79530629, 0.7955337 , 0.79243187,
0.76143056,
        0.79358329, 0.79354509, 0.79350334, 0.79220039, 0.76539974,
        0.7932431 , 0.7932431 , 0.79312852, 0.79278427, 0.76559259,
        0.7926427 , 0.79264272, 0.79298111, 0.79212776, 0.76578356,
        0.79241531, 0.79245351, 0.79253174, 0.79023619, 0.76566898]),
 'std_test_score': array([0.02036348, 0.02022411, 0.02018156, 0.01897866,
0.04052587,
        0.0213952 , 0.02132602, 0.02159337, 0.01975809, 0.04252833,
        0.02178892, 0.02178892, 0.02176089, 0.02289312, 0.04279299,
        0.02182471, 0.02187083, 0.02151123, 0.02318506, 0.04277193,
        0.02218278, 0.02214816, 0.02195392, 0.02169364, 0.0427192 ]),
 'rank_test_score': array([ 2,  3,  1, 16, 25,  4,  5,  6, 18, 24,  7,  7,  9,
11, 23, 13, 12,
        10, 19, 21, 17, 15, 14, 20, 22], dtype=int32),
 'split0_train_score': array([0.84454324, 0.84453129, 0.84451934, 0.84407715,
0.82021129,
        0.8454037 , 0.84534395, 0.84517663, 0.84405325, 0.82016349,
        0.84521249, 0.84527224, 0.84520054, 0.84371863, 0.82013959,
        0.8454037 , 0.8454037 , 0.84516468, 0.84345571, 0.82015154,
        0.84579808, 0.84578613, 0.84563077, 0.84551126, 0.82016349]),
 'split1_train_score': array([0.83661842, 0.83660663, 0.83657124, 0.83612301,
0.82932871,
        0.83752669, 0.83755028, 0.83734975, 0.8361466 , 0.82947026,
        0.83824622, 0.83816365, 0.83814006, 0.83814006, 0.82945846,
        0.83844675, 0.83844675, 0.83855291, 0.83825802, 0.82939948,
        0.83877703, 0.83877703, 0.8388478 , 0.83798672, 0.82944667]),
 'split2_train_score': array([0.83319584, 0.8331487 , 0.83320763, 0.83279515,
0.82468711,
```

```
        0.83461004, 0.83461004, 0.83451576, 0.83458647, 0.82507601,
        0.83515214, 0.83515214, 0.83506965, 0.83457468, 0.82480496,
        0.83567068, 0.83569425, 0.83575318, 0.83485752, 0.82484031,
        0.83601244, 0.83603601, 0.83576496, 0.83286586, 0.82480496]),
 'split3_train_score': array([0.83880548, 0.83879369, 0.83897047, 0.83795696,
0.81188866,
        0.84006647, 0.84011361, 0.84010182, 0.83987791, 0.82909468,
        0.84050251, 0.84049073, 0.8405143 , 0.84056143, 0.82949537,
        0.84069107, 0.84070285, 0.84089141, 0.84017253, 0.82950715,
        0.84096212, 0.84095034, 0.84110354, 0.84083249, 0.82957786]),
 'split4_train_score': array([0.8224244 , 0.82241261, 0.8224244 , 0.82102199,
0.7459283 ,
        0.8228958 , 0.82288401, 0.82288401, 0.82143446, 0.7459283 ,
        0.82295472, 0.82295472, 0.82282509, 0.82250689, 0.7459283 ,
        0.82337898, 0.82337898, 0.82339077, 0.82233012, 0.7459283 ,
        0.82341434, 0.82343791, 0.82357932, 0.82323756, 0.7459283 ]),
 'mean_train_score': array([0.83511748, 0.83509858, 0.83513861, 0.83439485,
0.80640881,
        0.83610054, 0.83610038, 0.8360056 , 0.83521974, 0.80994655,
        0.83641362, 0.8364067 , 0.83634993, 0.83590034, 0.80996534,
        0.83671824, 0.83672531, 0.83675059, 0.83581478, 0.80996536,
        0.8369928 , 0.83699748, 0.83698528, 0.83608678, 0.80998426]),
 'std_train_score': array([0.00734174, 0.00734357, 0.00735001, 0.00762646,
0.03078262,
        0.00749531, 0.0074906 , 0.00744432, 0.0076292 , 0.03218524,
        0.0074864 , 0.00749525, 0.00752915, 0.00733498, 0.0322052 ,
        0.00739245, 0.00739305, 0.0073576 , 0.00729514, 0.03220352,
        0.00750743, 0.00749425, 0.00743481, 0.0076214 , 0.03221531])}
```

[5]:
```
lg2.best_estimator_
```

[5]:
```
LogisticRegression(C=0.5, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=0, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
```

[6]:
```
lg_pred2 = lg2.best_estimator_.predict(gu_Xtest)
lg_pred2
```

[6]:
```
array([0, 0, 0, …, 0, 0, 0])
```

[7]:
```
prob6 = lg2.best_estimator_.predict_proba(gu_Xtest)
prob6
```

[7]:
```
array([[0.91372545, 0.08627455],
       [0.96230571, 0.03769429],
```

```
        [0.96092094, 0.03907906],
        …,
        [0.96151863, 0.03848137],
        [0.89257942, 0.10742058],
        [0.97582683, 0.02417317]])
```

[8]:
```python
from sklearn import metrics
lg_matrix2 = metrics.confusion_matrix(gu_ytest, lg_pred2)
lg_matrix2
```

[8]:
```
array([[2038,   15],
       [ 116,   19]])
```

[9]:
```python
from sklearn.metrics import classification_report
target_names2 = ['Still alive at 30 day', 'Died in 30 days']
print("", classification_report(gu_ytest, lg_pred2, 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        0.56      0.14      0.22       135

             accuracy                            0.94      2188
            macro avg        0.75      0.57      0.60      2188
         weighted avg        0.92      0.94      0.92      2188
```

[10]:
```python
from sklearn.metrics import roc_auc_score
lg_probs2 = lg2.best_estimator_.predict_proba(gu_Xtest)[:,1]
print(roc_auc_score(gu_ytest, lg_probs2))
```

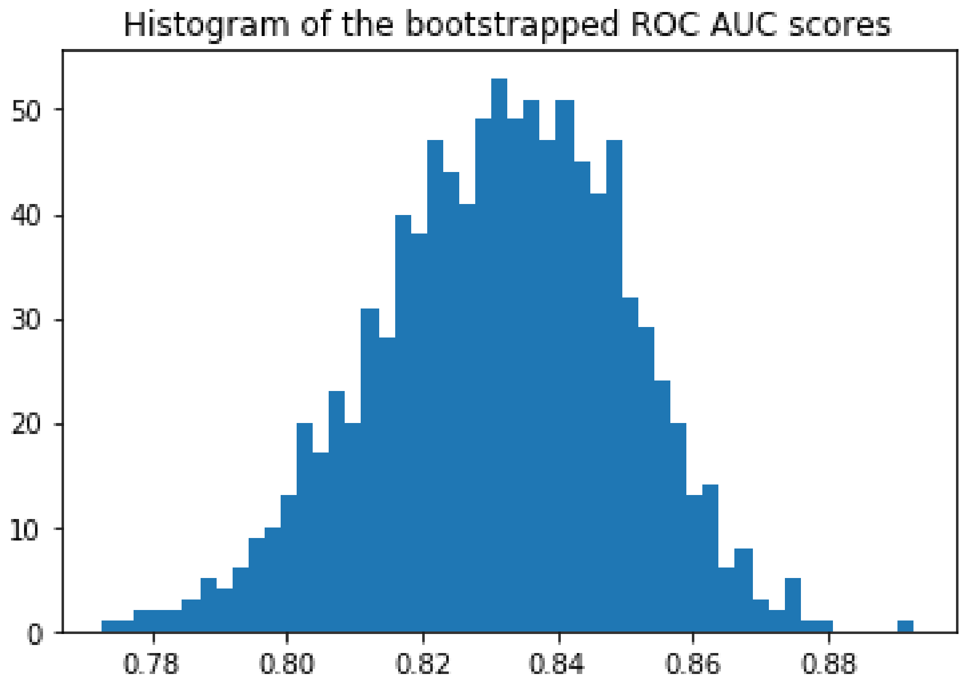
```
0.8302033158341001
```

[11]:
```python
gu_ytest = gu_ytest.values
```

[13]:
```python
#Boostrapping calculated 95% CI
y_pred = lg_probs2
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: 0.8302
```




```
Confidence interval for the score: [0.8000 - 0.8594]
```

[14]:
```
gu_ytest = gu_ytest.reshape((2188,))
```

[15]:
```
#pROC calculated 95% CI without bootstrapping
import numpy as np
import scipy.stats
from scipy import stats

# 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
```

```python
    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 = lg_probs2
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: 0.8302033158341
AUC COV: 0.00034444118566511161
95% AUC CI: [0.7938281  0.86657854]
```

[ ]:

```
```

5 Ridge Regression Model

[16]:

```
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)
```

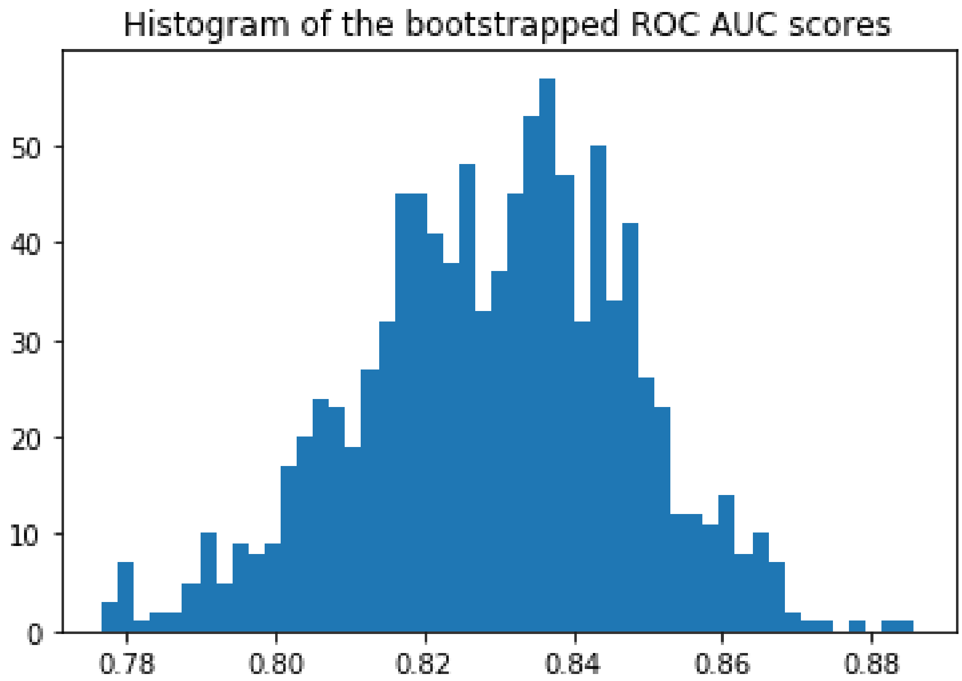
[16]: 0.8279590842669263

[17]:

```
#Boostrapping calculated 95% CI
y_pred = gu_ridgecv
y_true = gu_ytest
```

```python
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]
```

[18]:
```
#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 Artificial Neural Network

[19]:
```python
from sklearn.preprocessing import StandardScaler
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)
```

[20]:
```python
from sklearn.neural_network import MLPClassifier
seed(1)
# skf = StratifiedKFold(n_splits=5)
params7 = {'alpha' : [0.0001,0.01],
           'power_t': [0.5,0.75],
           'max_iter': [200,250]}
mlp2 = MLPClassifier(solver='lbfgs', random_state=1)
mlp2 = GridSearchCV(mlp2, cv=5, param_grid=params7, scoring = 'roc_auc',refit =␣
↪True,
                    n_jobs=-1, verbose = 5, return_train_score=True)

mlp2.fit(gu_Xtrains, gu_ytrain)
mlp2.cv_results_
```

```
Fitting 5 folds for each of 8 candidates, totalling 40 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  26 out of  40 | elapsed:    3.0s remaining:    1.6s
[Parallel(n_jobs=-1)]: Done  35 out of  40 | elapsed:    3.2s remaining:    0.5s
[Parallel(n_jobs=-1)]: Done  40 out of  40 | elapsed:    3.3s finished
/usr/local/lib/python3.7/site-
packages/sklearn/neural_network/multilayer_perceptron.py:921:
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)
```

[20]:
```
{'mean_fit_time': array([0.27808437, 0.30123062, 0.30036077, 0.32006493,
0.43673463,
        0.40928383, 0.35798874, 0.26584067]),
 'std_fit_time': array([0.03082211, 0.02415256, 0.02532153, 0.02310814,
```

```
0.04237356,
        0.02055945, 0.01295359, 0.02486431]),
 'mean_score_time': array([0.00549297, 0.00573235, 0.00630546, 0.00546994,
0.00648966,
        0.00515413, 0.00361214, 0.00244021]),
 'std_score_time': array([0.00012062, 0.00074194, 0.0011323 , 0.00083989,
0.00120899,
        0.00105471, 0.00112323, 0.0004813 ]),
 'param_alpha': masked_array(data=[0.0001, 0.0001, 0.0001, 0.0001, 0.01, 0.01,
0.01, 0.01],
              mask=[False, False, False, False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'param_max_iter': masked_array(data=[200, 200, 250, 250, 200, 200, 250, 250],
              mask=[False, False, False, False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'param_power_t': masked_array(data=[0.5, 0.75, 0.5, 0.75, 0.5, 0.75, 0.5,
0.75],
              mask=[False, False, False, False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'params': [{'alpha': 0.0001, 'max_iter': 200, 'power_t': 0.5},
  {'alpha': 0.0001, 'max_iter': 200, 'power_t': 0.75},
  {'alpha': 0.0001, 'max_iter': 250, 'power_t': 0.5},
  {'alpha': 0.0001, 'max_iter': 250, 'power_t': 0.75},
  {'alpha': 0.01, 'max_iter': 200, 'power_t': 0.5},
  {'alpha': 0.01, 'max_iter': 200, 'power_t': 0.75},
  {'alpha': 0.01, 'max_iter': 250, 'power_t': 0.5},
  {'alpha': 0.01, 'max_iter': 250, 'power_t': 0.75}],
 'split0_test_score': array([0.75896739, 0.75896739, 0.75896739, 0.75896739,
0.75126812,
        0.75126812, 0.75126812, 0.75126812]),
 'split1_test_score': array([0.5891495 , 0.5891495 , 0.5891495 , 0.5891495 ,
0.57360793,
        0.57360793, 0.57360793, 0.57360793]),
 'split2_test_score': array([0.74717703, 0.74717703, 0.74717703, 0.74717703,
0.73741627,
        0.73741627, 0.73741627, 0.73741627]),
 'split3_test_score': array([0.6415311 , 0.6415311 , 0.6415311 , 0.6415311 ,
0.66660287,
        0.66660287, 0.66660287, 0.66660287]),
 'split4_test_score': array([0.74220096, 0.74220096, 0.74220096, 0.74220096,
0.74411483,
        0.74411483, 0.74411483, 0.74411483]),
 'mean_test_score': array([0.69581855, 0.69581855, 0.69581855, 0.69581855,
0.6945968 ,
```

```
        0.6945968 , 0.6945968 , 0.6945968 ]),
 'std_test_score': array([0.06800126, 0.06800126, 0.06800126, 0.06800126,
0.06774062,
        0.06774062, 0.06774062, 0.06774062]),
 'rank_test_score': array([1, 1, 1, 1, 5, 5, 5, 5], dtype=int32),
 'split0_train_score': array([1., 1., 1., 1., 1., 1., 1., 1.]),
 'split1_train_score': array([1., 1., 1., 1., 1., 1., 1., 1.]),
 'split2_train_score': array([1., 1., 1., 1., 1., 1., 1., 1.]),
 'split3_train_score': array([1., 1., 1., 1., 1., 1., 1., 1.]),
 'split4_train_score': array([1., 1., 1., 1., 1., 1., 1., 1.]),
 'mean_train_score': array([1., 1., 1., 1., 1., 1., 1., 1.]),
 'std_train_score': array([0., 0., 0., 0., 0., 0., 0., 0.])}
```

[21]:
```
mlp2.best_estimator_
```

[21]:
```
MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=(100,), learning_rate='constant',
              learning_rate_init=0.001, max_iter=200, momentum=0.9,
              n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
              random_state=1, shuffle=True, solver='lbfgs', tol=0.0001,
              validation_fraction=0.1, verbose=False, warm_start=False)
```

[22]:
```
ann_pred2 = mlp2.best_estimator_.predict(gu_Xtests)
ann_pred2
```

[22]:
```
array([0, 0, 0, …, 0, 0, 0])
```

[23]:
```
prob7 = mlp2.best_estimator_.predict_proba(gu_Xtests)
prob7
```

[23]:
```
array([[1.00000000e+00, 4.92987998e-92],
       [1.00000000e+00, 3.30660487e-78],
       [1.00000000e+00, 5.34809233e-83],
       …,
       [1.00000000e+00, 1.21594736e-30],
       [1.00000000e+00, 8.23828743e-48],
       [1.00000000e+00, 8.12154336e-39]])
```

[24]:
```
ann_matrix2 = metrics.confusion_matrix(gu_ytest, ann_pred2)
ann_matrix2
```

[24]:
```
array([[1947,  106],
       [  93,   42]])
```

[25]:
```
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
```

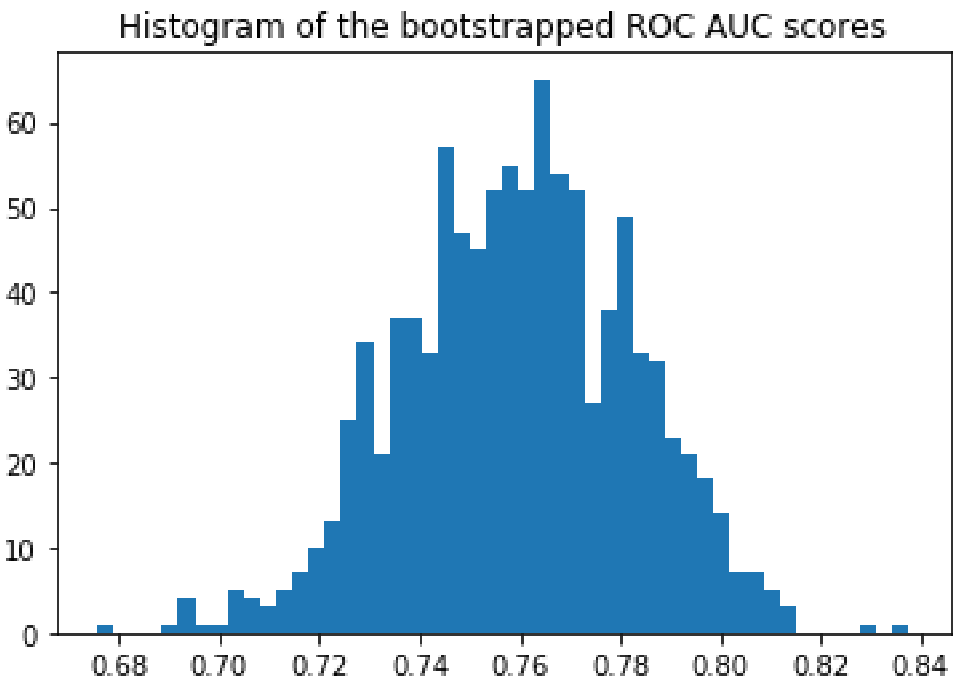
[26]:
```
ann_probs2 = mlp2.best_estimator_.predict_proba(gu_Xtests)[:,1]
print(roc_auc_score(gu_ytest, ann_probs2))
```

```
0.759576771120853
```

[27]:
```
#Boostrapping calculated 95% CI
y_pred = ann_probs2
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: 0.7596
```




```
Confidence interval for the score: [0.7232 - 0.7963]
```

[28]:
```
#pROC calculated 95% CI without bootstrapping
alpha = .95
y_pred = ann_probs2
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: 0.759576771120853
AUC COV: 0.0005295762772071846
95% AUC CI: [0.71447305 0.80468049]
```

[ ]:

7 Random Forest

[29]:

```
from sklearn.ensemble import RandomForestClassifier
seed(42)

params8 = {'n_estimators' : [10,100,150],
           'min_samples_leaf': [1,2,3]}
rf2 = RandomForestClassifier(random_state=42)
rf2 = GridSearchCV(rf2, cv=5, param_grid=params8, scoring = 'roc_auc',refit =␣
 ↪True,
                   n_jobs=-1, verbose = 5, return_train_score=True)

rf2.fit(gu_Xtrain,gu_ytrain)
rf2.cv_results_
```

```
Fitting 5 folds for each of 9 candidates, totalling 45 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  32 out of  45 | elapsed:    2.9s remaining:    1.2s
[Parallel(n_jobs=-1)]: Done  42 out of  45 | elapsed:    3.1s remaining:    0.2s
[Parallel(n_jobs=-1)]: Done  45 out of  45 | elapsed:    3.2s finished
/usr/local/lib/python3.7/site-packages/sklearn/model_selection/_search.py:715:
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().
  self.best_estimator_.fit(X, y, **fit_params)
```

[29]:

```
{'mean_fit_time': array([0.03426881, 0.27883363, 0.41557159, 0.03126526,
0.26815095,
```

```
       0.39476094, 0.03139949, 0.25758681, 0.27651353]),
 'std_fit_time': array([0.00141601, 0.00441368, 0.00469748, 0.00190913,
0.00351207,
       0.00498689, 0.0020941 , 0.00279042, 0.03816554]),
 'mean_score_time': array([0.00626278, 0.01958594, 0.02709827, 0.00515699,
0.01952624,
       0.02721171, 0.00534773, 0.01840653, 0.01495228]),
 'std_score_time': array([9.31513526e-04, 3.06682837e-04, 1.34821106e-03,
8.77498277e-05,
       6.76603885e-04, 5.90346595e-04, 1.60732816e-04, 1.65642035e-03,
       9.19618958e-04]),
 'param_min_samples_leaf': masked_array(data=[1, 1, 1, 2, 2, 2, 3, 3, 3],
             mask=[False, False, False, False, False, False, False, False,
                   False],
       fill_value='?',
            dtype=object),
 'param_n_estimators': masked_array(data=[10, 100, 150, 10, 100, 150, 10, 100,
150],
             mask=[False, False, False, False, False, False, False, False,
                   False],
       fill_value='?',
            dtype=object),
 'params': [{'min_samples_leaf': 1, 'n_estimators': 10},
  {'min_samples_leaf': 1, 'n_estimators': 100},
  {'min_samples_leaf': 1, 'n_estimators': 150},
  {'min_samples_leaf': 2, 'n_estimators': 10},
  {'min_samples_leaf': 2, 'n_estimators': 100},
  {'min_samples_leaf': 2, 'n_estimators': 150},
  {'min_samples_leaf': 3, 'n_estimators': 10},
  {'min_samples_leaf': 3, 'n_estimators': 100},
  {'min_samples_leaf': 3, 'n_estimators': 150}],
 'split0_test_score': array([0.73632246, 0.81757246, 0.81666667, 0.69248188,
0.79981884,
       0.81322464, 0.77753623, 0.83115942, 0.83822464]),
 'split1_test_score': array([0.72053776, 0.73951182, 0.74094203, 0.63567887,
0.75629291,
       0.74980931, 0.70061022, 0.76735317, 0.76372998]),
 'split2_test_score': array([0.71406699, 0.79282297, 0.79110048, 0.78593301,
0.77358852,
       0.77339713, 0.73645933, 0.78832536, 0.79406699]),
 'split3_test_score': array([0.69253589, 0.76870813, 0.7569378 , 0.7508134 ,
0.77521531,
       0.76669856, 0.69416268, 0.76669856, 0.75923445]),
 'split4_test_score': array([0.85339713, 0.8476555 , 0.85090909, 0.74277512,
0.82593301,
       0.8430622 , 0.83732057, 0.84976077, 0.85550239]),
 'mean_test_score': array([0.74334697, 0.79325071, 0.79131144, 0.72143872,
```

```
0.78616797,
        0.78924417, 0.74922326, 0.80067826, 0.80217458]),
 'std_test_score': array([0.05672565, 0.03752678, 0.03977511, 0.05229473,
0.02423484,
        0.03400995, 0.05311361, 0.0339244 , 0.03880603]),
 'rank_test_score': array([8, 3, 4, 9, 6, 5, 7, 2, 1], dtype=int32),
 'split0_train_score': array([0.99986257, 1.        , 1.        , 0.99442492,
0.99960562,
        0.99955782, 0.98568287, 0.99600841, 0.99709594]),
 'split1_train_score': array([0.99840169, 1.        , 1.        , 0.9910471 ,
0.99961074,
        0.9997169 , 0.9869658 , 0.99639053, 0.99689774]),
 'split2_train_score': array([0.99987037, 1.        , 1.        , 0.9958517 ,
0.99892757,
        0.99923398, 0.98869823, 0.99629953, 0.99648809]),
 'split3_train_score': array([0.99989983, 1.        , 1.        , 0.99582224,
0.99926933,
        0.9992929 , 0.98688925, 0.99560421, 0.99604026]),
 'split4_train_score': array([0.99975252, 1.        , 1.        , 0.99544512,
0.99951682,
        0.99963467, 0.98635303, 0.99675914, 0.99698305]),
 'mean_train_score': array([0.99955739, 1.        , 1.        , 0.99451822,
0.99938602,
        0.99948725, 0.98691784, 0.99621236, 0.99670102]),
 'std_train_score': array([5.80011721e-04, 4.96506831e-17, 0.00000000e+00,
1.81080856e-03,
        2.60624598e-04, 1.90457287e-04, 1.00170650e-03, 3.87232642e-04,
        3.88884105e-04])}
```

[30]:
```
rf2.best_estimator_
```

[30]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=None, max_features='auto', max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=3, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=150,
                       n_jobs=None, oob_score=False, random_state=42, verbose=0,
                       warm_start=False)
```

[31]:
```
rf_pred2 = rf2.best_estimator_.predict(gu_Xtest)
rf_pred2
```

[31]:
```
array([0, 0, 0, …, 0, 0, 0])
```

[32]:
```
prob8 = rf2.best_estimator_.predict_proba(gu_Xtest)
prob8
```

[32]:
```
array([[0.96881419, 0.03118581],
       [0.94229728, 0.05770272],
       [0.99634343, 0.00365657],
       …,
       [0.97513973, 0.02486027],
       [0.83568952, 0.16431048],
       [0.97687497, 0.02312503]])
```

[33]:
```
rf_matrix2 = metrics.confusion_matrix(gu_ytest, rf_pred2)
rf_matrix2
```

[33]:
```
array([[2051,    2],
       [ 128,    7]])
```

[34]:
```
print("", classification_report(gu_ytest, rf_pred2,
                                target_names=target_names2))
```

```
                        precision    recall  f1-score   support

Still alive at 30 day       0.94      1.00      0.97      2053
      Died in 30 days       0.78      0.05      0.10       135

             accuracy                           0.94      2188
            macro avg       0.86      0.53      0.53      2188
         weighted avg       0.93      0.94      0.92      2188
```

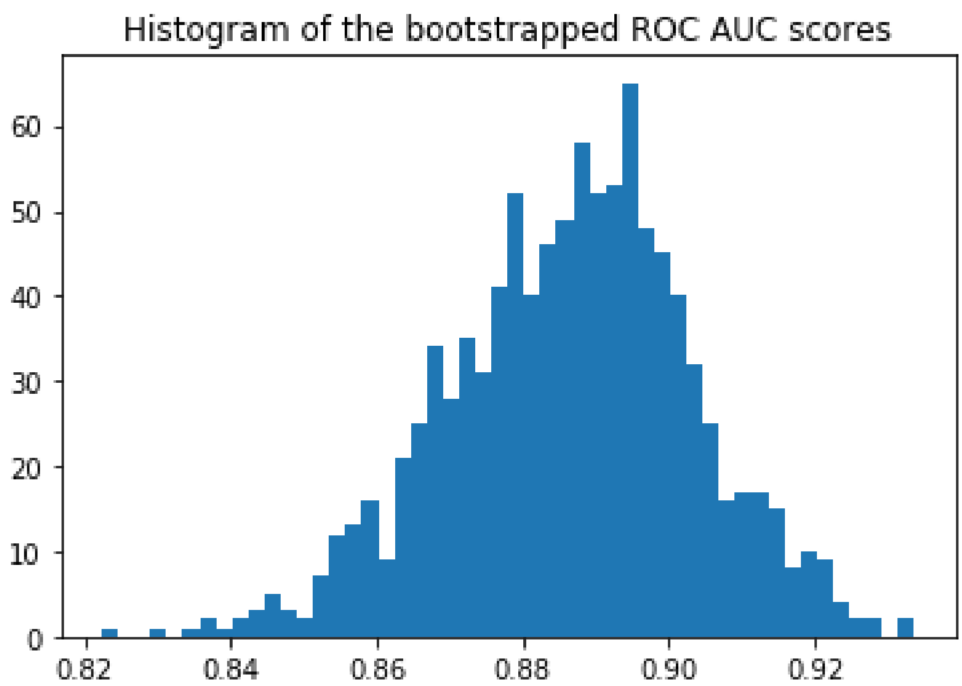
[35]:
```
rf_probs2 = rf2.best_estimator_.predict_proba(gu_Xtest)[:,1]
print(roc_auc_score(gu_ytest, rf_probs2))
```

```
0.885331312803305
```

[36]:
```
#Boostrapping calculated 95% CI
y_pred = rf_probs2
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: 0.8853
```



```
Confidence interval for the score: [0.8575 - 0.9138]
```

[37]:
```
#pROC calculated 95% CI without bootstrapping
alpha = .95
y_pred = rf_probs2
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: 0.885331312803305
AUC COV: 0.0002835084573262921
95% AUC CI: [0.85233001 0.91833262]
```

[ ]:

8 Gradient Boosting Machines

[38]:
```
from sklearn.ensemble import GradientBoostingClassifier
seed(10)

params9 = {'learning_rate' : [0.05,0.1,0.2],
           'n_estimators': [100,150],
           'min_samples_split': [2,3,4]}
gbm2 = GradientBoostingClassifier(random_state=10)
gbm2 = GridSearchCV(gbm2, cv=5, param_grid=params9, scoring = 'roc_auc',refit =␣
 ↪True,
                    n_jobs=-1, verbose = 5, return_train_score=True)
```

```
gbm2.fit(gu_Xtrain,gu_ytrain)
gbm2.cv_results_
```

```
Fitting 5 folds for each of 18 candidates, totalling 90 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  48 tasks      | elapsed:    3.3s
[Parallel(n_jobs=-1)]: Done  86 out of  90 | elapsed:    4.1s remaining:    0.2s
[Parallel(n_jobs=-1)]: Done  90 out of  90 | elapsed:    4.2s finished
/usr/local/lib/python3.7/site-
packages/sklearn/ensemble/gradient_boosting.py:1450: 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)
```

[38]:
```
{'mean_fit_time': array([0.21591759, 0.31171274, 0.21119657, 0.30165682,
0.20505085,
        0.31778083, 0.21604004, 0.31748967, 0.21064258, 0.30856447,
        0.19796476, 0.29425311, 0.20156455, 0.31398468, 0.2083787 ,
        0.30039825, 0.19774466, 0.20412064]),
 'std_fit_time': array([0.0056044 , 0.00702359, 0.01232283, 0.00609257,
0.00772605,
        0.01428885, 0.01009541, 0.00883545, 0.00734894, 0.0096207 ,
        0.00407808, 0.00844164, 0.00828902, 0.00990215, 0.00551054,
        0.01645469, 0.00495295, 0.02872154]),
 'mean_score_time': array([0.00563526, 0.00516491, 0.00454321, 0.00547996,
0.00660977,
        0.00500064, 0.00556917, 0.0047678 , 0.00544276, 0.00471015,
        0.00444007, 0.0050961 , 0.00572243, 0.00481868, 0.00482841,
        0.00471101, 0.00447583, 0.00257258]),
 'std_score_time': array([8.69524816e-04, 2.34852026e-04, 3.57920292e-04,
8.49180947e-04,
        1.95817910e-03, 4.04191528e-04, 1.16676813e-03, 2.35012840e-04,
        1.81804900e-03, 1.43596595e-04, 1.15136279e-04, 6.36400680e-04,
        1.36987998e-03, 5.35703957e-04, 5.79642088e-04, 1.33745581e-03,
        4.81773102e-05, 1.36307002e-04]),
 'param_learning_rate': masked_array(data=[0.05, 0.05, 0.05, 0.05, 0.05, 0.05,
0.1, 0.1, 0.1, 0.1,
                    0.1, 0.1, 0.2, 0.2, 0.2, 0.2, 0.2, 0.2],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False],
        fill_value='?',
             dtype=object),
 'param_min_samples_split': masked_array(data=[2, 2, 3, 3, 4, 4, 2, 2, 3, 3, 4,
4, 2, 2, 3, 3, 4, 4],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
```

```
                    False, False],
        fill_value='?',
             dtype=object),
 'param_n_estimators': masked_array(data=[100, 150, 100, 150, 100, 150, 100,
150, 100, 150, 100,
                    150, 100, 150, 100, 150, 100, 150],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False],
        fill_value='?',
             dtype=object),
 'params': [{'learning_rate': 0.05,
   'min_samples_split': 2,
   'n_estimators': 100},
  {'learning_rate': 0.05, 'min_samples_split': 2, 'n_estimators': 150},
  {'learning_rate': 0.05, 'min_samples_split': 3, 'n_estimators': 100},
  {'learning_rate': 0.05, 'min_samples_split': 3, 'n_estimators': 150},
  {'learning_rate': 0.05, 'min_samples_split': 4, 'n_estimators': 100},
  {'learning_rate': 0.05, 'min_samples_split': 4, 'n_estimators': 150},
  {'learning_rate': 0.1, 'min_samples_split': 2, 'n_estimators': 100},
  {'learning_rate': 0.1, 'min_samples_split': 2, 'n_estimators': 150},
  {'learning_rate': 0.1, 'min_samples_split': 3, 'n_estimators': 100},
  {'learning_rate': 0.1, 'min_samples_split': 3, 'n_estimators': 150},
  {'learning_rate': 0.1, 'min_samples_split': 4, 'n_estimators': 100},
  {'learning_rate': 0.1, 'min_samples_split': 4, 'n_estimators': 150},
  {'learning_rate': 0.2, 'min_samples_split': 2, 'n_estimators': 100},
  {'learning_rate': 0.2, 'min_samples_split': 2, 'n_estimators': 150},
  {'learning_rate': 0.2, 'min_samples_split': 3, 'n_estimators': 100},
  {'learning_rate': 0.2, 'min_samples_split': 3, 'n_estimators': 150},
  {'learning_rate': 0.2, 'min_samples_split': 4, 'n_estimators': 100},
  {'learning_rate': 0.2, 'min_samples_split': 4, 'n_estimators': 150}],
 'split0_test_score': array([0.77382246, 0.74565217, 0.78623188, 0.77028986,
0.76449275,
        0.75199275, 0.76539855, 0.75742754, 0.76105072, 0.7509058 ,
        0.74873188, 0.74764493, 0.75797101, 0.75181159, 0.77952899,
        0.77663043, 0.7365942 , 0.71050725]),
 'split1_test_score': array([0.73455378, 0.74523265, 0.74780702, 0.74599542,
0.74885584,
        0.75076278, 0.74084668, 0.73455378, 0.73741419, 0.70747521,
        0.73779558, 0.7305492 , 0.73512586, 0.71720061, 0.70175439,
        0.67982456, 0.71929825, 0.69012204]),
 'split2_test_score': array([0.77205742, 0.77301435, 0.77148325, 0.77799043,
0.77186603,
        0.77875598, 0.78373206, 0.78411483, 0.77952153, 0.78009569,
        0.78296651, 0.79043062, 0.78047847, 0.78028708, 0.78392344,
        0.78354067, 0.76574163, 0.74392344]),
 'split3_test_score': array([0.71100478, 0.68956938, 0.70258373, 0.6815311 ,
```

```
0.72114833,
       0.68660287, 0.64210526, 0.65129187, 0.63751196, 0.6845933 ,
       0.62411483, 0.63559809, 0.6076555 , 0.5691866 , 0.58660287,
       0.5984689 , 0.62602871, 0.58947368]),
 'split4_test_score': array([0.82755981, 0.81818182, 0.82526316, 0.82315789,
0.82870813,
       0.81952153, 0.81913876, 0.8061244 , 0.82411483, 0.83062201,
       0.81741627, 0.81033493, 0.82277512, 0.79177033, 0.82507177,
       0.8292823 , 0.8262201 , 0.81722488]),
 'mean_test_score': array([0.7637934 , 0.75431212, 0.76668756, 0.75979783,
0.76699846,
       0.75751508, 0.75025846, 0.7467088 , 0.74793334, 0.75070926,
       0.74221088, 0.74290959, 0.74082065, 0.72208836, 0.73541342,
       0.73357139, 0.73476854, 0.71023694]),
 'std_test_score': array([0.03965896, 0.04185947, 0.04071548, 0.04637067,
0.0353775 ,
       0.04331702, 0.05974167, 0.05342025, 0.06203115, 0.05190187,
       0.06524184, 0.06075918, 0.07252206, 0.08058388, 0.08433274,
       0.08319232, 0.06534065, 0.07418203]),
 'rank_test_score': array([ 3,  6,  2,  4,  1,  5,  8, 10,  9,  7, 12, 11, 13,
17, 14, 16, 15,
       18], dtype=int32),
 'split0_train_score': array([0.96441632, 0.97920551, 0.96136288, 0.98126105,
0.96078923,
       0.98096228, 0.98704527, 0.99378555, 0.98962666, 0.99580525,
       0.98771452, 0.99539892, 0.99916344, 1.        , 0.99998805,
       1.        , 0.99939051, 1.        ]),
 'split1_train_score': array([0.96179978, 0.9758425 , 0.9625606 , 0.9752999 ,
0.95852059,
       0.97386083, 0.99284004, 0.99933944, 0.98685964, 0.99902096,
       0.99203793, 0.99542329, 0.99911533, 1.        , 1.        ,
       1.        , 1.        , 1.        ]),
 'split2_train_score': array([0.94661418, 0.97316567, 0.94852335, 0.9745563 ,
0.94762769,
       0.97388455, 0.98368963, 0.99614632, 0.98559879, 0.99629953,
       0.98736064, 0.99589884, 0.99985858, 1.        , 0.99981144,
       1.        , 0.99941075, 1.        ]),
 'split3_train_score': array([0.95035001, 0.97647135, 0.94496429, 0.97045513,
0.95118085,
       0.97113866, 0.9809732 , 0.99402503, 0.98148585, 0.99641738,
       0.98154477, 0.99252834, 1.        , 1.        , 0.99868009,
       1.        , 0.9985151 , 1.        ]),
 'split4_train_score': array([0.94395668, 0.96426804, 0.94526481, 0.96371414,
0.94537087,
       0.96552903, 0.97766752, 0.99895114, 0.98188653, 0.99645273,
       0.98304146, 0.99435501, 0.99545101, 1.        , 0.99949325,
       1.        , 0.99996465, 1.        ]),
```

```
 'mean_train_score': array([0.9534274 , 0.97379061, 0.95253518, 0.97305731,
0.95269785,
        0.97307507, 0.98444313, 0.9964495 , 0.98509149, 0.99679917,
        0.98633987, 0.99472088, 0.99871767, 1.        , 0.99959456,
        1.        , 0.9994562 , 1.        ]),
 'std_train_score': array([0.0082029 , 0.00513396, 0.00780652, 0.00580764,
0.00601776,
        0.00498344, 0.00521145, 0.00235272, 0.00307323, 0.00113487,
        0.00372215, 0.0012068 , 0.00167176, 0.        , 0.0004925 ,
        0.        , 0.00053783, 0.        ])}
```

[39]:
```
gbm2.best_estimator_
```

[39]:
```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
                           learning_rate=0.05, loss='deviance', max_depth=3,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=4,
                           min_weight_fraction_leaf=0.0, n_estimators=100,
                           n_iter_no_change=None, presort='auto',
                           random_state=10, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
```

[40]:
```
gbm_pred2 = gbm2.best_estimator_.predict(gu_Xtest)
gbm_pred2
```

[40]:
```
array([0, 0, 0, …, 0, 0, 0])
```

[41]:
```
prob9 = gbm2.best_estimator_.predict_proba(gu_Xtest)
prob9
```

[41]:
```
array([[0.947186  , 0.052814  ],
       [0.97310495, 0.02689505],
       [0.97898104, 0.02101896],
       …,
       [0.9711283 , 0.0288717 ],
       [0.87880754, 0.12119246],
       [0.98384301, 0.01615699]])
```

[42]:
```
gbm_matrix2 = metrics.confusion_matrix(gu_ytest, gbm_pred2)
gbm_matrix2
```

[42]:
```
array([[2044,    9],
       [ 113,   22]])
```

[43]:
```
print("", classification_report(gu_ytest, gbm_pred2,
                                target_names=target_names2))
```

```
                    precision    recall  f1-score   support

Still alive at 30 day       0.95      1.00      0.97      2053
      Died in 30 days       0.71      0.16      0.27       135

             accuracy                           0.94      2188
            macro avg       0.83      0.58      0.62      2188
         weighted avg       0.93      0.94      0.93      2188
```

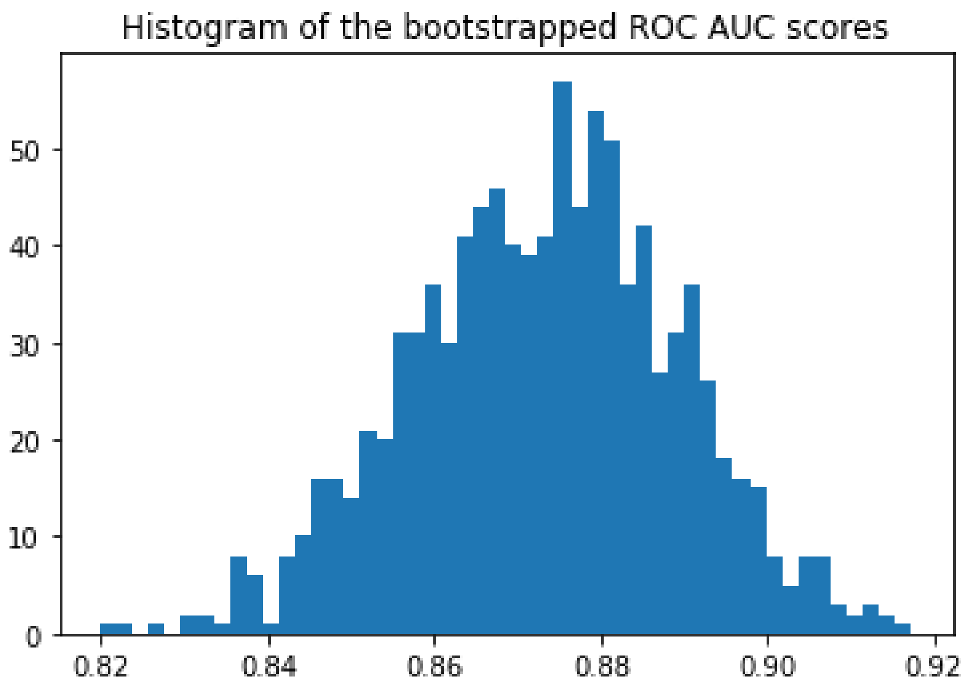
[44]:
```
gbm_probs2 = gbm2.best_estimator_.predict_proba(gu_Xtest)[:,1]
print(roc_auc_score(gu_ytest, gbm_probs2))
```

0.8720806047157728

[45]:
```
#Boostrapping calculated 95% CI
y_pred = gbm_probs2
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: 0.8721
```




```
Confidence interval for the score: [0.8468 - 0.8985]
```

[46]:
```
#pROC calculated 95% CI without bootstrapping
alpha = .95
y_pred = gbm_probs2
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: 0.8720806047157728
AUC COV: 0.000262844614256567
95% AUC CI: [0.84030472 0.90385649]
```

[ ]:

9 Support Vector Machine

[47]:

```
# mlp2 = MLPClassifier(solver='lbfgs', random_state=1)
# mlp2 = GridSearchCV(mlp2, cv=5, param_grid=params7, scoring = 'roc_auc',refit␣
 ↪= True,
#                      n_jobs=-1, verbose = 5, return_train_score=True)

# mlp2.fit(gu_Xtrains, gu_ytrain)
# mlp2.cv_results_
```

[48]:

```
from sklearn.svm import SVC
seed(0)
params0 = {'degree' : [2,3,4],
           'C': [0.5,1.0,1.5,2.0,2.5],
           'tol':[1e-5,1e-3,1e-1],
           'max_iter':[-1,-2,-3]}
svc = SVC(random_state=0,probability=True)
svc = GridSearchCV(svc, cv=5, param_grid=params0, scoring = 'roc_auc',refit =␣
 ↪True,
                   n_jobs=-1, verbose = 5, return_train_score=True)

svc.fit(gu_Xtrains, gu_ytrain)
svc.cv_results_
```

```
Fitting 5 folds for each of 135 candidates, totalling 675 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  48 tasks      | elapsed:    2.5s
```

```
[Parallel(n_jobs=-1)]: Done 360 tasks      | elapsed:    4.3s
[Parallel(n_jobs=-1)]: Done 675 out of 675 | elapsed:    6.5s finished
/usr/local/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)
```

[48]:

```
{'mean_fit_time': array([0.18540196, 0.16512508, 0.10027161, 0.00636835,
0.011408  ,
        0.00640464, 0.00484252, 0.00761418, 0.004704  , 0.17196598,
        0.1543386 , 0.08540335, 0.00465307, 0.01207347, 0.00430737,
        0.00397797, 0.00396605, 0.00802488, 0.16754427, 0.14290657,
        0.09345403, 0.00498338, 0.00888481, 0.00997963, 0.00808883,
        0.0064959 , 0.01369696, 0.18986068, 0.15622382, 0.09752955,
        0.01427655, 0.00909853, 0.01050215, 0.00775685, 0.00451374,
        0.00494375, 0.16934972, 0.14957738, 0.10681543, 0.00462661,
        0.0134491 , 0.00961742, 0.00875678, 0.00921769, 0.01324368,
        0.17771864, 0.15891609, 0.11198125, 0.0048141 , 0.00887904,
        0.0081162 , 0.01260376, 0.01269135, 0.00879664, 0.1729794 ,
        0.17033334, 0.11628809, 0.00977497, 0.00810337, 0.01259518,
        0.00517888, 0.00815959, 0.01047635, 0.18751884, 0.15756435,
        0.12989793, 0.00968699, 0.00917711, 0.00890985, 0.01459284,
        0.00995102, 0.00964823, 0.18468142, 0.16715102, 0.1215344 ,
        0.01424832, 0.00943203, 0.0088469 , 0.00914478, 0.00932856,
        0.01385231, 0.17395859, 0.17995033, 0.12344398, 0.00479822,
        0.00425143, 0.01044936, 0.01345501, 0.01370587, 0.00937605,
        0.18189578, 0.1617857 , 0.13018198, 0.00765572, 0.00885057,
        0.00932741, 0.00986443, 0.00426741, 0.00943685, 0.18498373,
        0.14978323, 0.121945  , 0.00947456, 0.00982594, 0.01547756,
        0.00447216, 0.01475086, 0.01064625, 0.18718858, 0.16806998,
        0.11111975, 0.01516066, 0.00911298, 0.00882025, 0.009515  ,
        0.01516213, 0.00997443, 0.18117323, 0.13921661, 0.10797253,
        0.00698838, 0.01322842, 0.01485782, 0.01534815, 0.00909338,
        0.01098657, 0.20106673, 0.18124571, 0.13345981, 0.01398358,
        0.00454082, 0.01458611, 0.00467801, 0.00424972, 0.01361704]),
 'std_fit_time': array([0.00848725, 0.01119252, 0.0057811 , 0.00082957,
0.00716746,
        0.00179574, 0.00085137, 0.00768214, 0.00035607, 0.00634435,
        0.01300377, 0.00401205, 0.00040175, 0.00989767, 0.0001645 ,
        0.0004344 , 0.00058992, 0.00822779, 0.0108452 , 0.01058227,
        0.01123612, 0.00187159, 0.00667158, 0.00983636, 0.00251094,
        0.00418156, 0.01011101, 0.01969538, 0.01100629, 0.00712422,
        0.00988601, 0.0087463 , 0.00743766, 0.00707995, 0.00135709,
        0.00127267, 0.01372623, 0.01418551, 0.00843769, 0.00068116,
        0.01019579, 0.00741849, 0.00810113, 0.00841863, 0.01015065,
        0.01182725, 0.0142521 , 0.01348738, 0.00086841, 0.00847861,
```

```
       0.00781157, 0.01012961, 0.01033028, 0.00930278, 0.00612014,
       0.0076139 , 0.00541977, 0.0094211 , 0.00757999, 0.00869422,
       0.00091247, 0.00605653, 0.00607782, 0.01518229, 0.0149688 ,
       0.00746361, 0.00954949, 0.0090895 , 0.00807134, 0.01056334,
       0.00729085, 0.00931335, 0.00927606, 0.00878707, 0.01065305,
       0.01044697, 0.00886738, 0.00893095, 0.00963046, 0.00924755,
       0.01112167, 0.01092518, 0.01410936, 0.01418105, 0.00067858,
       0.000149  , 0.00736129, 0.01133064, 0.01080346, 0.00975834,
       0.01002593, 0.00450806, 0.00801369, 0.00686558, 0.00634197,
       0.01024002, 0.00946756, 0.00028332, 0.0095398 , 0.00922454,
       0.02105057, 0.00488867, 0.0097109 , 0.01046812, 0.01266652,
       0.00018411, 0.01253121, 0.00987994, 0.02208216, 0.00972925,
       0.0234004 , 0.01250364, 0.00971536, 0.00884316, 0.00973546,
       0.01211441, 0.00913913, 0.01960502, 0.03427041, 0.03575138,
       0.00544525, 0.00929742, 0.012499  , 0.01314969, 0.01031478,
       0.01052743, 0.01259819, 0.01125997, 0.00197284, 0.01138635,
       0.00033816, 0.01213102, 0.00092279, 0.00045724, 0.01187802]),
 'mean_score_time': array([0.00787678, 0.00813813, 0.00661359, 0.00334158,
0.00330143,
       0.00320425, 0.00252223, 0.00250821, 0.00281754, 0.0073864 ,
       0.00746522, 0.00545125, 0.00265427, 0.00221181, 0.00271449,
       0.00236363, 0.00247936, 0.00233541, 0.00665402, 0.00749092,
       0.0051445 , 0.00267925, 0.00265188, 0.00262628, 0.00268431,
       0.00280676, 0.00258212, 0.00651951, 0.00751319, 0.00590034,
       0.00255022, 0.002426  , 0.00260658, 0.00234594, 0.00233126,
       0.00256047, 0.00726919, 0.00696082, 0.00666714, 0.00255022,
       0.00258737, 0.00254335, 0.00263739, 0.00259538, 0.00251942,
       0.00684757, 0.0073761 , 0.00658021, 0.0026298 , 0.00252967,
       0.00254445, 0.00248985, 0.00249152, 0.00230393, 0.00651298,
       0.00702057, 0.00681252, 0.00257082, 0.00254769, 0.00258279,
       0.0025054 , 0.00252776, 0.00509501, 0.0076848 , 0.00675354,
       0.00718651, 0.00255637, 0.0025908 , 0.00258222, 0.00253906,
       0.00260372, 0.00256705, 0.00682673, 0.00735283, 0.00702848,
       0.00253229, 0.00263042, 0.00256872, 0.00248003, 0.00252376,
       0.00262566, 0.0068573 , 0.00785637, 0.00684109, 0.00266538,
       0.00248895, 0.00269365, 0.00254269, 0.00255785, 0.00255413,
       0.00698214, 0.00788698, 0.00698242, 0.00259123, 0.00265512,
       0.0025867 , 0.00264392, 0.00259418, 0.00260024, 0.00764933,
       0.0072216 , 0.00661154, 0.00262008, 0.00260477, 0.0026917 ,
       0.00276151, 0.00256839, 0.00258312, 0.00673547, 0.00790801,
       0.00592222, 0.00260396, 0.00260434, 0.00260334, 0.00261846,
       0.00266166, 0.00257874, 0.00706878, 0.00654998, 0.00497141,
       0.00260124, 0.0026278 , 0.00252972, 0.00266623, 0.00232759,
       0.00273304, 0.00792794, 0.00729094, 0.00686145, 0.00268016,
       0.00292373, 0.00253935, 0.00288682, 0.0027626 , 0.00243959]),
 'std_score_time': array([2.87411737e-04, 4.63137535e-04, 1.51797732e-03,
1.46091466e-04,
```

```
       2.28482504e-04, 1.45025583e-04, 3.01728108e-04, 3.88204063e-04,
       2.97787787e-04, 1.33230418e-04, 1.50741387e-04, 1.40284709e-03,
       1.29308039e-04, 4.06348725e-04, 6.69399553e-05, 2.70533947e-04,
       3.90612194e-04, 2.75697109e-04, 1.13492480e-03, 2.27489716e-03,
       1.01891635e-03, 4.32107023e-04, 1.28469293e-04, 1.36422319e-04,
       1.25249737e-04, 2.08444639e-04, 1.30379819e-04, 1.16436994e-03,
       3.41501125e-04, 1.34112114e-03, 1.24667304e-04, 5.72289222e-05,
       5.00082387e-05, 3.50568350e-04, 3.87047285e-04, 1.13032898e-04,
       1.24469239e-03, 2.05522979e-03, 4.02804757e-04, 7.03929553e-05,
       1.05761944e-04, 1.13652274e-04, 2.02944662e-04, 1.32784671e-04,
       1.04190108e-04, 1.04594423e-03, 1.23916212e-03, 9.00013792e-04,
       1.66910675e-04, 1.47082460e-04, 1.11130826e-04, 7.45159943e-05,
       1.26946411e-04, 3.48031862e-04, 1.09898227e-03, 1.12088881e-03,
       6.95942116e-04, 9.98013907e-05, 1.52716017e-04, 3.55085235e-05,
       5.74205001e-05, 5.78172477e-05, 4.99197208e-03, 9.69989416e-04,
       1.33305929e-03, 1.29948681e-03, 5.73731214e-05, 1.03954503e-04,
       1.45231683e-04, 9.59858631e-05, 9.39989127e-05, 1.16002927e-04,
       1.13734730e-03, 6.64573389e-04, 4.05013272e-04, 7.14292513e-05,
       1.46481969e-04, 1.13885346e-04, 7.00958732e-05, 9.62902482e-05,
       9.73745844e-05, 1.03727851e-03, 3.15178216e-04, 9.95119948e-04,
       1.43059063e-04, 1.06038399e-04, 1.44095503e-04, 6.06445015e-05,
       5.29573983e-05, 2.54594429e-05, 1.39284034e-03, 2.00901098e-04,
       5.71676837e-04, 8.02758719e-05, 1.19940480e-04, 1.62898531e-04,
       1.58483671e-04, 9.89844801e-05, 8.72741492e-05, 4.42642815e-04,
       1.66324721e-03, 1.00788414e-03, 5.11221029e-05, 9.03261491e-05,
       1.45435755e-04, 1.08404563e-04, 6.08937294e-05, 3.46935575e-04,
       1.28425048e-03, 1.93802894e-03, 1.37961062e-03, 4.25710110e-05,
       1.14703235e-04, 1.12571791e-04, 8.95967504e-05, 1.27484927e-04,
       1.06348157e-04, 9.65184379e-04, 1.68632697e-03, 1.06728784e-03,
       1.06208410e-04, 7.97121990e-05, 7.77815685e-05, 7.36034877e-05,
       3.58041353e-04, 1.94363084e-04, 2.69574455e-04, 1.07379797e-03,
       1.04211311e-03, 9.94722314e-05, 1.29690766e-04, 4.85789060e-04,
       1.42674220e-04, 1.53350593e-04, 4.32757571e-04]),
 'param_C': masked_array(data=[0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5,
0.5,
                   0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5,
                   0.5, 0.5, 0.5, 0.5, 0.5, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
                   1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
                   1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.5,
                   1.5, 1.5, 1.5, 1.5, 1.5, 1.5, 1.5, 1.5, 1.5, 1.5, 1.5,
                   1.5, 1.5, 1.5, 1.5, 1.5, 1.5, 1.5, 1.5, 1.5, 1.5, 1.5,
                   1.5, 1.5, 1.5, 1.5, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0,
                   2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0,
                   2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.5, 2.5,
                   2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 2.5,
                   2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 2.5, 2.5,
                   2.5, 2.5, 2.5],
```

```
            mask=[False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False],
      fill_value='?',
           dtype=object),
 'param_degree': masked_array(data=[2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3,
3, 3, 3,
                  4, 4, 4, 4, 4, 4, 4, 4, 4, 2, 2, 2, 2, 2, 2, 2, 2, 2,
                  3, 3, 3, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4,
                  2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3,
                  4, 4, 4, 4, 4, 4, 4, 4, 4, 2, 2, 2, 2, 2, 2, 2, 2, 2,
                  3, 3, 3, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4,
                  2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3,
                  4, 4, 4, 4, 4, 4, 4, 4, 4],
            mask=[False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False],
      fill_value='?',
           dtype=object),
```

```
 'param_max_iter': masked_array(data=[-1, -1, -1, -2, -2, -2, -3, -3, -3, -1,
-1, -1, -2, -2,
                   -2, -3, -3, -3, -1, -1, -1, -2, -2, -2, -3, -3, -3, -1,
                   -1, -1, -2, -2, -2, -3, -3, -3, -1, -1, -1, -2, -2, -2,
                   -3, -3, -3, -1, -1, -1, -2, -2, -2, -3, -3, -3, -1, -1,
                   -1, -2, -2, -2, -3, -3, -3, -1, -1, -1, -2, -2, -2, -3,
                   -3, -3, -1, -1, -1, -2, -2, -2, -3, -3, -3, -1, -1, -1,
                   -2, -2, -2, -3, -3, -3, -1, -1, -1, -2, -2, -2, -3, -3,
                   -3, -1, -1, -1, -2, -2, -2, -3, -3, -3, -1, -1, -1, -2,
                   -2, -2, -3, -3, -3, -1, -1, -1, -2, -2, -2, -3, -3, -3,
                   -1, -1, -1, -2, -2, -2, -3, -3, -3],
             mask=[False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False],
       fill_value='?',
            dtype=object),
 'param_tol': masked_array(data=[1e-05, 0.001, 0.1, 1e-05, 0.001, 0.1, 1e-05,
0.001,
                   0.1, 1e-05, 0.001, 0.1, 1e-05, 0.001, 0.1, 1e-05,
                   0.001, 0.1, 1e-05, 0.001, 0.1, 1e-05, 0.001, 0.1,
                   1e-05, 0.001, 0.1, 1e-05, 0.001, 0.1, 1e-05, 0.001,
                   0.1, 1e-05, 0.001, 0.1, 1e-05, 0.001, 0.1, 1e-05,
                   0.001, 0.1, 1e-05, 0.001, 0.1, 1e-05, 0.001, 0.1,
                   1e-05, 0.001, 0.1, 1e-05, 0.001, 0.1, 1e-05, 0.001,
                   0.1, 1e-05, 0.001, 0.1, 1e-05, 0.001, 0.1, 1e-05,
                   0.001, 0.1, 1e-05, 0.001, 0.1, 1e-05, 0.001, 0.1,
                   1e-05, 0.001, 0.1, 1e-05, 0.001, 0.1, 1e-05, 0.001,
                   0.1, 1e-05, 0.001, 0.1, 1e-05, 0.001, 0.1, 1e-05,
                   0.001, 0.1, 1e-05, 0.001, 0.1, 1e-05, 0.001, 0.1,
                   1e-05, 0.001, 0.1, 1e-05, 0.001, 0.1, 1e-05, 0.001,
                   0.1, 1e-05, 0.001, 0.1, 1e-05, 0.001, 0.1, 1e-05,
                   0.001, 0.1, 1e-05, 0.001, 0.1, 1e-05, 0.001, 0.1,
                   1e-05, 0.001, 0.1, 1e-05, 0.001, 0.1, 1e-05, 0.001,
```

```
                  0.1, 1e-05, 0.001, 0.1, 1e-05, 0.001, 0.1],
            mask=[False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False, False,
                  False, False, False, False, False, False, False],
      fill_value='?',
           dtype=object),
'params': [{'C': 0.5, 'degree': 2, 'max_iter': -1, 'tol': 1e-05},
 {'C': 0.5, 'degree': 2, 'max_iter': -1, 'tol': 0.001},
 {'C': 0.5, 'degree': 2, 'max_iter': -1, 'tol': 0.1},
 {'C': 0.5, 'degree': 2, 'max_iter': -2, 'tol': 1e-05},
 {'C': 0.5, 'degree': 2, 'max_iter': -2, 'tol': 0.001},
 {'C': 0.5, 'degree': 2, 'max_iter': -2, 'tol': 0.1},
 {'C': 0.5, 'degree': 2, 'max_iter': -3, 'tol': 1e-05},
 {'C': 0.5, 'degree': 2, 'max_iter': -3, 'tol': 0.001},
 {'C': 0.5, 'degree': 2, 'max_iter': -3, 'tol': 0.1},
 {'C': 0.5, 'degree': 3, 'max_iter': -1, 'tol': 1e-05},
 {'C': 0.5, 'degree': 3, 'max_iter': -1, 'tol': 0.001},
 {'C': 0.5, 'degree': 3, 'max_iter': -1, 'tol': 0.1},
 {'C': 0.5, 'degree': 3, 'max_iter': -2, 'tol': 1e-05},
 {'C': 0.5, 'degree': 3, 'max_iter': -2, 'tol': 0.001},
 {'C': 0.5, 'degree': 3, 'max_iter': -2, 'tol': 0.1},
 {'C': 0.5, 'degree': 3, 'max_iter': -3, 'tol': 1e-05},
 {'C': 0.5, 'degree': 3, 'max_iter': -3, 'tol': 0.001},
 {'C': 0.5, 'degree': 3, 'max_iter': -3, 'tol': 0.1},
 {'C': 0.5, 'degree': 4, 'max_iter': -1, 'tol': 1e-05},
 {'C': 0.5, 'degree': 4, 'max_iter': -1, 'tol': 0.001},
 {'C': 0.5, 'degree': 4, 'max_iter': -1, 'tol': 0.1},
 {'C': 0.5, 'degree': 4, 'max_iter': -2, 'tol': 1e-05},
 {'C': 0.5, 'degree': 4, 'max_iter': -2, 'tol': 0.001},
 {'C': 0.5, 'degree': 4, 'max_iter': -2, 'tol': 0.1},
 {'C': 0.5, 'degree': 4, 'max_iter': -3, 'tol': 1e-05},
 {'C': 0.5, 'degree': 4, 'max_iter': -3, 'tol': 0.001},
 {'C': 0.5, 'degree': 4, 'max_iter': -3, 'tol': 0.1},
```

```
{'C': 1.0, 'degree': 2, 'max_iter': -1, 'tol': 1e-05},
{'C': 1.0, 'degree': 2, 'max_iter': -1, 'tol': 0.001},
{'C': 1.0, 'degree': 2, 'max_iter': -1, 'tol': 0.1},
{'C': 1.0, 'degree': 2, 'max_iter': -2, 'tol': 1e-05},
{'C': 1.0, 'degree': 2, 'max_iter': -2, 'tol': 0.001},
{'C': 1.0, 'degree': 2, 'max_iter': -2, 'tol': 0.1},
{'C': 1.0, 'degree': 2, 'max_iter': -3, 'tol': 1e-05},
{'C': 1.0, 'degree': 2, 'max_iter': -3, 'tol': 0.001},
{'C': 1.0, 'degree': 2, 'max_iter': -3, 'tol': 0.1},
{'C': 1.0, 'degree': 3, 'max_iter': -1, 'tol': 1e-05},
{'C': 1.0, 'degree': 3, 'max_iter': -1, 'tol': 0.001},
{'C': 1.0, 'degree': 3, 'max_iter': -1, 'tol': 0.1},
{'C': 1.0, 'degree': 3, 'max_iter': -2, 'tol': 1e-05},
{'C': 1.0, 'degree': 3, 'max_iter': -2, 'tol': 0.001},
{'C': 1.0, 'degree': 3, 'max_iter': -2, 'tol': 0.1},
{'C': 1.0, 'degree': 3, 'max_iter': -3, 'tol': 1e-05},
{'C': 1.0, 'degree': 3, 'max_iter': -3, 'tol': 0.001},
{'C': 1.0, 'degree': 3, 'max_iter': -3, 'tol': 0.1},
{'C': 1.0, 'degree': 4, 'max_iter': -1, 'tol': 1e-05},
{'C': 1.0, 'degree': 4, 'max_iter': -1, 'tol': 0.001},
{'C': 1.0, 'degree': 4, 'max_iter': -1, 'tol': 0.1},
{'C': 1.0, 'degree': 4, 'max_iter': -2, 'tol': 1e-05},
{'C': 1.0, 'degree': 4, 'max_iter': -2, 'tol': 0.001},
{'C': 1.0, 'degree': 4, 'max_iter': -2, 'tol': 0.1},
{'C': 1.0, 'degree': 4, 'max_iter': -3, 'tol': 1e-05},
{'C': 1.0, 'degree': 4, 'max_iter': -3, 'tol': 0.001},
{'C': 1.0, 'degree': 4, 'max_iter': -3, 'tol': 0.1},
{'C': 1.5, 'degree': 2, 'max_iter': -1, 'tol': 1e-05},
{'C': 1.5, 'degree': 2, 'max_iter': -1, 'tol': 0.001},
{'C': 1.5, 'degree': 2, 'max_iter': -1, 'tol': 0.1},
{'C': 1.5, 'degree': 2, 'max_iter': -2, 'tol': 1e-05},
{'C': 1.5, 'degree': 2, 'max_iter': -2, 'tol': 0.001},
{'C': 1.5, 'degree': 2, 'max_iter': -2, 'tol': 0.1},
{'C': 1.5, 'degree': 2, 'max_iter': -3, 'tol': 1e-05},
{'C': 1.5, 'degree': 2, 'max_iter': -3, 'tol': 0.001},
{'C': 1.5, 'degree': 2, 'max_iter': -3, 'tol': 0.1},
{'C': 1.5, 'degree': 3, 'max_iter': -1, 'tol': 1e-05},
{'C': 1.5, 'degree': 3, 'max_iter': -1, 'tol': 0.001},
{'C': 1.5, 'degree': 3, 'max_iter': -1, 'tol': 0.1},
{'C': 1.5, 'degree': 3, 'max_iter': -2, 'tol': 1e-05},
{'C': 1.5, 'degree': 3, 'max_iter': -2, 'tol': 0.001},
{'C': 1.5, 'degree': 3, 'max_iter': -2, 'tol': 0.1},
{'C': 1.5, 'degree': 3, 'max_iter': -3, 'tol': 1e-05},
{'C': 1.5, 'degree': 3, 'max_iter': -3, 'tol': 0.001},
{'C': 1.5, 'degree': 3, 'max_iter': -3, 'tol': 0.1},
{'C': 1.5, 'degree': 4, 'max_iter': -1, 'tol': 1e-05},
{'C': 1.5, 'degree': 4, 'max_iter': -1, 'tol': 0.001},
```

```
{'C': 1.5, 'degree': 4, 'max_iter': -1, 'tol': 0.1},
{'C': 1.5, 'degree': 4, 'max_iter': -2, 'tol': 1e-05},
{'C': 1.5, 'degree': 4, 'max_iter': -2, 'tol': 0.001},
{'C': 1.5, 'degree': 4, 'max_iter': -2, 'tol': 0.1},
{'C': 1.5, 'degree': 4, 'max_iter': -3, 'tol': 1e-05},
{'C': 1.5, 'degree': 4, 'max_iter': -3, 'tol': 0.001},
{'C': 1.5, 'degree': 4, 'max_iter': -3, 'tol': 0.1},
{'C': 2.0, 'degree': 2, 'max_iter': -1, 'tol': 1e-05},
{'C': 2.0, 'degree': 2, 'max_iter': -1, 'tol': 0.001},
{'C': 2.0, 'degree': 2, 'max_iter': -1, 'tol': 0.1},
{'C': 2.0, 'degree': 2, 'max_iter': -2, 'tol': 1e-05},
{'C': 2.0, 'degree': 2, 'max_iter': -2, 'tol': 0.001},
{'C': 2.0, 'degree': 2, 'max_iter': -2, 'tol': 0.1},
{'C': 2.0, 'degree': 2, 'max_iter': -3, 'tol': 1e-05},
{'C': 2.0, 'degree': 2, 'max_iter': -3, 'tol': 0.001},
{'C': 2.0, 'degree': 2, 'max_iter': -3, 'tol': 0.1},
{'C': 2.0, 'degree': 3, 'max_iter': -1, 'tol': 1e-05},
{'C': 2.0, 'degree': 3, 'max_iter': -1, 'tol': 0.001},
{'C': 2.0, 'degree': 3, 'max_iter': -1, 'tol': 0.1},
{'C': 2.0, 'degree': 3, 'max_iter': -2, 'tol': 1e-05},
{'C': 2.0, 'degree': 3, 'max_iter': -2, 'tol': 0.001},
{'C': 2.0, 'degree': 3, 'max_iter': -2, 'tol': 0.1},
{'C': 2.0, 'degree': 3, 'max_iter': -3, 'tol': 1e-05},
{'C': 2.0, 'degree': 3, 'max_iter': -3, 'tol': 0.001},
{'C': 2.0, 'degree': 3, 'max_iter': -3, 'tol': 0.1},
{'C': 2.0, 'degree': 4, 'max_iter': -1, 'tol': 1e-05},
{'C': 2.0, 'degree': 4, 'max_iter': -1, 'tol': 0.001},
{'C': 2.0, 'degree': 4, 'max_iter': -1, 'tol': 0.1},
{'C': 2.0, 'degree': 4, 'max_iter': -2, 'tol': 1e-05},
{'C': 2.0, 'degree': 4, 'max_iter': -2, 'tol': 0.001},
{'C': 2.0, 'degree': 4, 'max_iter': -2, 'tol': 0.1},
{'C': 2.0, 'degree': 4, 'max_iter': -3, 'tol': 1e-05},
{'C': 2.0, 'degree': 4, 'max_iter': -3, 'tol': 0.001},
{'C': 2.0, 'degree': 4, 'max_iter': -3, 'tol': 0.1},
{'C': 2.5, 'degree': 2, 'max_iter': -1, 'tol': 1e-05},
{'C': 2.5, 'degree': 2, 'max_iter': -1, 'tol': 0.001},
{'C': 2.5, 'degree': 2, 'max_iter': -1, 'tol': 0.1},
{'C': 2.5, 'degree': 2, 'max_iter': -2, 'tol': 1e-05},
{'C': 2.5, 'degree': 2, 'max_iter': -2, 'tol': 0.001},
{'C': 2.5, 'degree': 2, 'max_iter': -2, 'tol': 0.1},
{'C': 2.5, 'degree': 2, 'max_iter': -3, 'tol': 1e-05},
{'C': 2.5, 'degree': 2, 'max_iter': -3, 'tol': 0.001},
{'C': 2.5, 'degree': 2, 'max_iter': -3, 'tol': 0.1},
{'C': 2.5, 'degree': 3, 'max_iter': -1, 'tol': 1e-05},
{'C': 2.5, 'degree': 3, 'max_iter': -1, 'tol': 0.001},
{'C': 2.5, 'degree': 3, 'max_iter': -1, 'tol': 0.1},
{'C': 2.5, 'degree': 3, 'max_iter': -2, 'tol': 1e-05},
```

```
 {'C': 2.5, 'degree': 3, 'max_iter': -2, 'tol': 0.001},
 {'C': 2.5, 'degree': 3, 'max_iter': -2, 'tol': 0.1},
 {'C': 2.5, 'degree': 3, 'max_iter': -3, 'tol': 1e-05},
 {'C': 2.5, 'degree': 3, 'max_iter': -3, 'tol': 0.001},
 {'C': 2.5, 'degree': 3, 'max_iter': -3, 'tol': 0.1},
 {'C': 2.5, 'degree': 4, 'max_iter': -1, 'tol': 1e-05},
 {'C': 2.5, 'degree': 4, 'max_iter': -1, 'tol': 0.001},
 {'C': 2.5, 'degree': 4, 'max_iter': -1, 'tol': 0.1},
 {'C': 2.5, 'degree': 4, 'max_iter': -2, 'tol': 1e-05},
 {'C': 2.5, 'degree': 4, 'max_iter': -2, 'tol': 0.001},
 {'C': 2.5, 'degree': 4, 'max_iter': -2, 'tol': 0.1},
 {'C': 2.5, 'degree': 4, 'max_iter': -3, 'tol': 1e-05},
 {'C': 2.5, 'degree': 4, 'max_iter': -3, 'tol': 0.001},
 {'C': 2.5, 'degree': 4, 'max_iter': -3, 'tol': 0.1}],
'split0_test_score': array([0.71068841, 0.71105072, 0.66123188, 0.5       , 0.5
,
       0.5       , 0.5       , 0.5       , 0.5       , 0.71068841,
       0.71105072, 0.66123188, 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.71068841, 0.71105072,
       0.66123188, 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.71068841, 0.71086957, 0.69547101,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.71068841, 0.71086957, 0.69547101, 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.71068841, 0.71086957, 0.69547101, 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.71032609,
       0.71050725, 0.69655797, 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.71032609, 0.71050725,
       0.69655797, 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.71032609, 0.71050725, 0.69655797,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.70634058, 0.70634058, 0.7115942 , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.70634058, 0.70634058, 0.7115942 , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.70634058,
       0.70634058, 0.7115942 , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.70561594, 0.70561594,
       0.69855072, 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.70561594, 0.70561594, 0.69855072,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.70561594, 0.70561594, 0.69855072, 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ]),
'split1_test_score': array([0.63558352, 0.63558352, 0.64206712, 0.5       , 0.5
,
       0.5       , 0.5       , 0.5       , 0.5       , 0.63558352,
       0.63558352, 0.64206712, 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.63558352, 0.63558352,
```

```
       0.64206712, 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.63558352, 0.63539283, 0.63729977,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.63558352, 0.63539283, 0.63729977, 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.63558352, 0.63539283, 0.63729977, 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.63577422,
       0.63558352, 0.63558352, 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.63577422, 0.63558352,
       0.63558352, 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.63577422, 0.63558352, 0.63558352,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.63348589, 0.63329519, 0.62547674, 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.63348589, 0.63329519, 0.62547674, 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.63348589,
       0.63329519, 0.62547674, 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.62948131, 0.62909992,
       0.62471396, 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.62948131, 0.62909992, 0.62471396,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.62948131, 0.62909992, 0.62471396, 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ]),
 'split2_test_score': array([0.75330144, 0.75368421, 0.75349282, 0.5       , 0.5
,
       0.5       , 0.5       , 0.5       , 0.5       , 0.75330144,
       0.75368421, 0.75349282, 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.75330144, 0.75368421,
       0.75349282, 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.75425837, 0.75425837, 0.76401914,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.75425837, 0.75425837, 0.76401914, 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.75425837, 0.75425837, 0.76401914, 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.75406699,
       0.75406699, 0.74947368, 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.75406699, 0.75406699,
       0.74947368, 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.75406699, 0.75406699, 0.74947368,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.75425837, 0.75425837, 0.75444976, 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.75425837, 0.75425837, 0.75444976, 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.75425837,
       0.75425837, 0.75444976, 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.75330144, 0.75349282,
       0.75253589, 0.5       , 0.5       , 0.5       , 0.5       ,
```

```
        0.5       , 0.5       , 0.75330144, 0.75349282, 0.75253589,
        0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
        0.5       , 0.75330144, 0.75349282, 0.75253589, 0.5       ,
        0.5       , 0.5       , 0.5       , 0.5       , 0.5       ]),
 'split3_test_score': array([0.66564593, 0.66507177, 0.65837321, 0.5       , 0.5
,
        0.5       , 0.5       , 0.5       , 0.5       , 0.66564593,
        0.66507177, 0.65837321, 0.5       , 0.5       , 0.5       ,
        0.5       , 0.5       , 0.5       , 0.66564593, 0.66507177,
        0.65837321, 0.5       , 0.5       , 0.5       , 0.5       ,
        0.5       , 0.5       , 0.66564593, 0.66507177, 0.65645933,
        0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
        0.5       , 0.66564593, 0.66507177, 0.65645933, 0.5       ,
        0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
        0.66564593, 0.66507177, 0.65645933, 0.5       , 0.5       ,
        0.5       , 0.5       , 0.5       , 0.5       , 0.66602871,
        0.66583732, 0.66047847, 0.5       , 0.5       , 0.5       ,
        0.5       , 0.5       , 0.5       , 0.66602871, 0.66583732,
        0.66047847, 0.5       , 0.5       , 0.5       , 0.5       ,
        0.5       , 0.5       , 0.66602871, 0.66583732, 0.66047847,
        0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
        0.5       , 0.66526316, 0.66526316, 0.66851675, 0.5       ,
        0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
        0.66526316, 0.66526316, 0.66851675, 0.5       , 0.5       ,
        0.5       , 0.5       , 0.5       , 0.5       , 0.66526316,
        0.66526316, 0.66851675, 0.5       , 0.5       , 0.5       ,
        0.5       , 0.5       , 0.5       , 0.66239234, 0.66239234,
        0.66488038, 0.5       , 0.5       , 0.5       , 0.5       ,
        0.5       , 0.5       , 0.66239234, 0.66239234, 0.66488038,
        0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
        0.5       , 0.66239234, 0.66239234, 0.66488038, 0.5       ,
        0.5       , 0.5       , 0.5       , 0.5       , 0.5       ]),
 'split4_test_score': array([0.75062201, 0.75062201, 0.76708134, 0.5       , 0.5
,
        0.5       , 0.5       , 0.5       , 0.5       , 0.75062201,
        0.75062201, 0.76708134, 0.5       , 0.5       , 0.5       ,
        0.5       , 0.5       , 0.5       , 0.75062201, 0.75062201,
        0.76708134, 0.5       , 0.5       , 0.5       , 0.5       ,
        0.5       , 0.5       , 0.75062201, 0.75004785, 0.735311  ,
        0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
        0.5       , 0.75062201, 0.75004785, 0.735311  , 0.5       ,
        0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
        0.75062201, 0.75004785, 0.735311  , 0.5       , 0.5       ,
        0.5       , 0.5       , 0.5       , 0.5       , 0.74909091,
        0.74870813, 0.73186603, 0.5       , 0.5       , 0.5       ,
        0.5       , 0.5       , 0.5       , 0.74909091, 0.74870813,
        0.73186603, 0.5       , 0.5       , 0.5       , 0.5       ,
```

```
       0.5       , 0.5       , 0.74909091, 0.74870813, 0.73186603,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.74755981, 0.74755981, 0.74220096, 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.74755981, 0.74755981, 0.74220096, 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.74755981,
       0.74755981, 0.74220096, 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.74698565, 0.74679426,
       0.73741627, 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.74698565, 0.74679426, 0.73741627,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.74698565, 0.74679426, 0.73741627, 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ]),
 'mean_test_score': array([0.70313259, 0.7031672 , 0.69636454, 0.5       , 0.5
,
       0.5       , 0.5       , 0.5       , 0.5       , 0.70313259,
       0.7031672 , 0.69636454, 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.70313259, 0.7031672 ,
       0.69636454, 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.70332359, 0.7030926 , 0.697668  ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.70332359, 0.7030926 , 0.697668  , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.70332359, 0.7030926 , 0.697668  , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.70302157,
       0.70290519, 0.69475414, 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.70302157, 0.70290519,
       0.69475414, 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.70302157, 0.70290519, 0.69475414,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.7013422 , 0.70130401, 0.70041192, 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.7013422 , 0.70130401, 0.70041192, 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.7013422 ,
       0.70130401, 0.70041192, 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.69951599, 0.69943961,
       0.69557529, 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.69951599, 0.69943961, 0.69557529,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.69951599, 0.69943961, 0.69557529, 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ]),
 'std_test_score': array([0.04645699, 0.04664472, 0.05268863, 0.        , 0.
,
       0.        , 0.        , 0.        , 0.        , 0.04645699,
       0.04664472, 0.05268863, 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.04645699, 0.04664472,
       0.05268863, 0.        , 0.        , 0.        , 0.        ,
```

```
       0.        , 0.        , 0.04666436, 0.04670289, 0.0472365 ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.04666436, 0.04670289, 0.0472365 , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.04666436, 0.04670289, 0.0472365 , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.04618664,
       0.04620279, 0.04253053, 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.04618664, 0.04620279,
       0.04253053, 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.04618664, 0.04620279, 0.04253053,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.0466069 , 0.04666253, 0.04779264, 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.0466069 , 0.04666253, 0.04779264, 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.0466069 ,
       0.04666253, 0.04779264, 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.04787488, 0.04799191,
       0.04682372, 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.04787488, 0.04799191, 0.04682372,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.04787488, 0.04799191, 0.04682372, 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ]),
 'rank_test_score': array([ 7,  4, 37, 46, 46, 46, 46, 46, 46,  7,  4, 37, 46,
46, 46, 46, 46,
       46,  7,  4, 37, 46, 46, 46, 46, 46, 46,  1, 10, 34, 46, 46, 46, 46,
       46, 46,  1, 10, 34, 46, 46, 46, 46, 46, 46,  1, 10, 34, 46, 46, 46,
       46, 46, 46, 13, 16, 43, 46, 46, 46, 46, 46, 46, 13, 16, 43, 46, 46,
       46, 46, 46, 46, 13, 16, 43, 46, 46, 46, 46, 46, 46, 19, 22, 25, 46,
       46, 46, 46, 46, 46, 19, 22, 25, 46, 46, 46, 46, 46, 46, 19, 22, 25,
       46, 46, 46, 46, 46, 46, 28, 31, 40, 46, 46, 46, 46, 46, 46, 28, 31,
       40, 46, 46, 46, 46, 46, 46, 28, 31, 40, 46, 46, 46, 46, 46, 46],
      dtype=int32),
 'split0_train_score': array([0.99435919, 0.99433529, 0.97523782, 0.5       ,
0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.99435919,
       0.99433529, 0.97523782, 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.99435919, 0.99433529,
       0.97523782, 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.99435919, 0.99433529, 0.9930207 ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.99435919, 0.99433529, 0.9930207 , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.99435919, 0.99433529, 0.9930207 , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.9944787 ,
       0.99446675, 0.99396482, 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.9944787 , 0.99446675,
       0.99396482, 0.5       , 0.5       , 0.5       , 0.5       ,
```

```
       0.5       , 0.5       , 0.9944787 , 0.99446675, 0.99396482,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.99498064, 0.99498064, 0.99514795, 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.99498064, 0.99498064, 0.99514795, 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.99498064,
       0.99498064, 0.99514795, 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.99554233, 0.99554233,
       0.99561404, 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.99554233, 0.99554233, 0.99561404,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.99554233, 0.99554233, 0.99561404, 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ]),
 'split1_train_score': array([0.98992651, 0.98993831, 0.9853498 , 0.5       ,
0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.98992651,
       0.98993831, 0.9853498 , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.98992651, 0.98993831,
       0.9853498 , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.98992651, 0.98992651, 0.99313493,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.98992651, 0.98992651, 0.99313493, 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.98992651, 0.98992651, 0.99313493, 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.99045732,
       0.99075221, 0.99279286, 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.99045732, 0.99075221,
       0.99279286, 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.99045732, 0.99075221, 0.99279286,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.99470375, 0.99469196, 0.99442066, 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.99470375, 0.99469196, 0.99442066, 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.99470375,
       0.99469196, 0.99442066, 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.99525815, 0.99525815,
       0.99529353, 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.99525815, 0.99525815, 0.99529353,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.99525815, 0.99525815, 0.99529353, 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ]),
 'split2_train_score': array([0.99226907, 0.99225729, 0.98568129, 0.5       ,
0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.99226907,
       0.99225729, 0.98568129, 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.99226907, 0.99225729,
       0.98568129, 0.5       , 0.5       , 0.5       , 0.5       ,
```

```
       0.5       , 0.5       , 0.99244585, 0.99246942, 0.99043062,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.99244585, 0.99246942, 0.99043062, 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.99244585, 0.99246942, 0.99043062, 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.99612275,
       0.99612275, 0.99492069, 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.99612275, 0.99612275,
       0.99492069, 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.99612275, 0.99612275, 0.99492069,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.99959931, 0.99959931, 0.99767837, 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.99959931, 0.99959931, 0.99767837, 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.99959931,
       0.99959931, 0.99767837, 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.99962288, 0.99962288,
       0.99832654, 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.99962288, 0.99962288, 0.99832654,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.99962288, 0.99962288, 0.99832654, 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ]),
 'split3_train_score': array([0.99305867, 0.99305867, 0.97720791, 0.5       ,
0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.99305867,
       0.99305867, 0.97720791, 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.99305867, 0.99305867,
       0.97720791, 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.99305867, 0.99304688, 0.98767294,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.99305867, 0.99304688, 0.98767294, 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.99305867, 0.99304688, 0.98767294, 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.99345935,
       0.99345935, 0.99098451, 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.99345935, 0.99345935,
       0.99098451, 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.99345935, 0.99345935, 0.99098451,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.99423716, 0.99423716, 0.9908431 , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.99423716, 0.99423716, 0.9908431 , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.99423716,
       0.99423716, 0.9908431 , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.99472034, 0.99472034,
       0.99358899, 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.99472034, 0.99472034, 0.99358899,
```

```
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.99472034, 0.99472034, 0.99358899, 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ]),
 'split4_train_score': array([0.98874537, 0.98850968, 0.97104438, 0.5       ,
0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.98874537,
       0.98850968, 0.97104438, 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.98874537, 0.98850968,
       0.97104438, 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.98874537, 0.98701299, 0.98657694,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.98874537, 0.98701299, 0.98657694, 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.98874537, 0.98701299, 0.98657694, 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.98913428,
       0.98912249, 0.99191553, 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.98913428, 0.98912249,
       0.99191553, 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.98913428, 0.98912249, 0.99191553,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.99014778, 0.99017135, 0.99266976, 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.99014778, 0.99017135, 0.99266976, 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.99014778,
       0.99017135, 0.99266976, 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.9907606 , 0.9907606 ,
       0.99067811, 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.9907606 , 0.9907606 , 0.99067811,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.9907606 , 0.9907606 , 0.99067811, 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ]),
 'mean_train_score': array([0.99167176, 0.99161985, 0.97890424, 0.5       , 0.5
,
       0.5       , 0.5       , 0.5       , 0.5       , 0.99167176,
       0.99161985, 0.97890424, 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.99167176, 0.99161985,
       0.97890424, 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.99170712, 0.99135822, 0.99016723,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.99170712, 0.99135822, 0.99016723, 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
       0.99170712, 0.99135822, 0.99016723, 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.5       , 0.99273048,
       0.99278471, 0.99291568, 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.5       , 0.99273048, 0.99278471,
       0.99291568, 0.5       , 0.5       , 0.5       , 0.5       ,
       0.5       , 0.5       , 0.99273048, 0.99278471, 0.99291568,
```

```
         0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
         0.5       , 0.99473373, 0.99473608, 0.99415197, 0.5       ,
         0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
         0.99473373, 0.99473608, 0.99415197, 0.5       , 0.5       ,
         0.5       , 0.5       , 0.5       , 0.5       , 0.99473373,
         0.99473608, 0.99415197, 0.5       , 0.5       , 0.5       ,
         0.5       , 0.5       , 0.5       , 0.99518086, 0.99518086,
         0.99470024, 0.5       , 0.5       , 0.5       , 0.5       ,
         0.5       , 0.5       , 0.99518086, 0.99518086, 0.99470024,
         0.5       , 0.5       , 0.5       , 0.5       , 0.5       ,
         0.5       , 0.99518086, 0.99518086, 0.99470024, 0.5       ,
         0.5       , 0.5       , 0.5       , 0.5       , 0.5       ]),
 'std_train_score': array([0.00205486, 0.00211428, 0.0057545 , 0.        , 0.
,
         0.        , 0.        , 0.        , 0.        , 0.00205486,
         0.00211428, 0.0057545 , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.00205486, 0.00211428,
         0.0057545 , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.00206632, 0.00260308, 0.00268814,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.00206632, 0.00260308, 0.00268814, 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.00206632, 0.00260308, 0.00268814, 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.00257663,
         0.00252863, 0.00140467, 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.00257663, 0.00252863,
         0.00140467, 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.00257663, 0.00252863, 0.00140467,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.00300044, 0.00299327, 0.00230847, 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.00300044, 0.00299327, 0.00230847, 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.00300044,
         0.00299327, 0.00230847, 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.00281492, 0.00281492,
         0.0025199 , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.00281492, 0.00281492, 0.0025199 ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.00281492, 0.00281492, 0.0025199 , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ])}
```

[49]:
```
svc.best_estimator_
```

[49]:
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=2, gamma='auto_deprecated',
    kernel='rbf', max_iter=-1, probability=True, random_state=0, shrinking=True,
    tol=1e-05, verbose=False)
```

[50]:
```
svc_pred = svc.best_estimator_.predict(gu_Xtests)
svc_pred
```

[50]:
```
array([0, 0, 0, …, 0, 0, 0])
```

[51]:
```
prob_svc = svc.best_estimator_.predict_proba(gu_Xtests)
prob_svc
```

[51]:
```
array([[0.95247783, 0.04752217],
       [0.91819468, 0.08180532],
       [0.93822348, 0.06177652],
       …,
       [0.92257555, 0.07742445],
       [0.93326204, 0.06673796],
       [0.9271016 , 0.0728984 ]])
```

[52]:
```
svc_matrix = metrics.confusion_matrix(gu_ytest, svc_pred)
svc_matrix
```

[52]:
```
array([[2052,    1],
       [ 131,    4]])
```

[53]:
```
target_names2 = ['Still alive at 30 day', 'Died in 30 days']
print("", classification_report(gu_ytest, svc_pred,
                                target_names=target_names2))
```

```
                        precision    recall  f1-score   support

Still alive at 30 day       0.94      1.00      0.97      2053
      Died in 30 days       0.80      0.03      0.06       135

             accuracy                           0.94      2188
            macro avg       0.87      0.51      0.51      2188
         weighted avg       0.93      0.94      0.91      2188
```

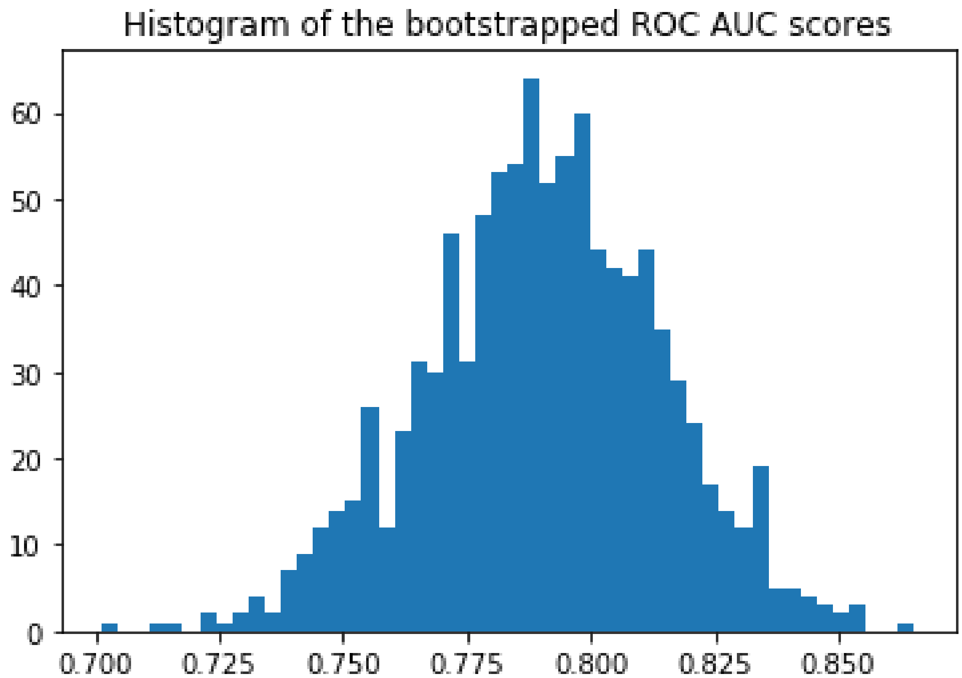
[54]:
```
svc_probs = svc.best_estimator_.predict_proba(gu_Xtests)[:,1]
print(roc_auc_score(gu_ytest, svc_probs))
```

```
0.7897854990889575
```

[55]:
```
#Boostrapping calculated 95% CI
y_pred = svc_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: 0.7898
```



Confidence interval for the score: [0.7489 - 0.8302]

[56]:
```
#pROC calculated 95% CI without bootstrapping
alpha = .95
y_pred = svc_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: 0.7897854990889575
AUC COV: 0.0005276809118318064
95% AUC CI: [0.74476257 0.83480843]
```

[ ]:

[ ]:

[ ]: