cs109a_hw6_submission

October 25, 2017

1 CS 109A/STAT 121A/AC 209A/CSCI E-109A: Homework 6

2 Reg-Logistic Regression, ROC, and Data Imputation

Harvard University Fall 2017 Instructors: Pavlos Protopapas, Kevin Rader, Rahul Dave, Margo Levine

---

2.0.1 INSTRUCTIONS

- To submit your assignment follow the instructions given in canvas.
- Restart the kernel and run the whole notebook again before you submit.
- Do not include your name(s) in the notebook if you are submitting as a group.
- If you submit individually and you have worked with someone, please include the name of your [one] partner below.

Your partner's name (if you submit separately):
Enrollment Status (109A, 121A, 209A, or E109A):
Import libraries:

```
In [1]: import numpy as np
        import pandas as pd
        import matplotlib
        import matplotlib.pyplot as plt
        from sklearn.linear_model import LogisticRegressionCV
        import sklearn.metrics as metrics
        from sklearn.preprocessing import PolynomialFeatures
        from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
        from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
        from sklearn.neighbors import KNeighborsClassifier
        from sklearn.tree import DecisionTreeClassifier
        from sklearn.model_selection import cross_val_score
        from sklearn.metrics import accuracy_score

        from sklearn.tree import export_graphviz
        from IPython.display import Image
        from IPython.display import display
        %matplotlib inline

        from sklearn.model_selection import train_test_split
        from sklearn.metrics import confusion_matrix
        from sklearn.metrics import roc_curve, auc
        from sklearn.metrics import roc_auc_score
        import seaborn.apionly as sns
```

```
/Users/yijunshen/anaconda3/lib/python3.6/site-packages/seaborn/apionly.py:6: UserWarning: As s
  warnings.warn(msg, UserWarning)
```

2.1 Automated Breast Cancer Detection

In this homework, we will consider the problem of early breast cancer detection from X-ray images. Specifically, given a candidate region of interest (ROI) from an X-ray image of a patient's breast, the goal is to predict if the region corresponds to a malignant tumor (label 1) or is normal (label 0). The training and test data sets for this problem is provided in the file `hw6_dataset.csv`. Each row in these files corresponds to a ROI in a patient's X-ray, with columns 1-117 containing features computed using standard image processing algorithms. The last column contains the class label, and is based on a radiologist's opinion or a biopsy. This data was obtained from the KDD Cup 2008 challenge.

The data set contain a total of 69,098 candidate ROIs, of which only 409 are malignant, while the remaining are all normal.

Note: be careful of reading/treating column names and row names in this data set.

2.2 Question 1: Beyond Classification Accuracy

0. Split the data set into a training set and a testing set. The training set should be 75% of the original data set, and the testing set 25%. Use `np.random.seed(9001)`.

1. Fit a logistic regression classifier to the training set and report the accuracy of the classifier on the test set. You should use L_2 regularization in logistic regression, with the regularization parameter tuned using cross-validation.

   1. How does the fitted model compare with a classifier that predicts 'normal' (label 0) on all patients?
   2. Do you think the difference in the classification accuracies are large enough to declare logistic regression as a better classifier than the all 0's classifier? Why or why not?

For applications with imbalanced class labels, in this case when there are many more healthy subjects ($Y = 0$) than those with cancer ($Y = 1$), the classification accuracy may not be the best metric to evaluate a classifier's performance. As an alternative, we could analyze the confusion table for the classifier.

Compute the confusion table for both the fitted classifier and the classifier that predicts all 0's.

Using the entries of the confusion table compute the *true positive rate* and the *true negative rate* for the two classifiers. Explain what these evaluation metrics mean for the specific task of cancer detection. Based on the observed metrics, comment on whether the fitted model is better than the all 0's classifier.

What is the *false positive rate* of the fitted classifier, and how is it related to its true positive and true negative rate? Why is a classifier with high false positive rate undesirable for a cancer detection task?

Hint: You may use the `metrics.confusion_matrix` function to compute the confusion matrix for a classification model.

```
In [2]: np.random.seed(9001)
        df = pd.read_csv('hw6_dataset.csv')
        df.head()
```

```
Out[2]:    -1.4399999999999999891e-01  -1.4299999999999999882e-01  \
        0                    -0.01100                       0.138
        1                     0.21200                      -0.313
        2                     0.21500                      -0.184
        3                     0.27900                      -0.197
        4                     0.00922                      -0.138

           -1.1600000000000000059e-01  -1.0299999999999999943e-01  \
        0                     -0.2230                     -0.1730
        1                      0.2660                      0.2320
        2                      0.0274                      0.0494
        3                      0.1270                      0.0973
        4                      0.1690                      0.1540

           2.2600000000000000064e-01  2.0999999999999999922e-01  \
        0                      0.188                      0.284
        1                     -1.190                     -1.150
        2                      0.443                      0.463
        3                     -0.213                     -0.150
        4                     -0.391                     -0.397

           -9.7999999999999999822e-01  -7.8000000000000000266e-01  \
        0                     -0.0522                      -0.256
        1                     -1.8100                      -1.560
        2                     -1.0500                      -0.941
        3                     -1.3200                      -0.994
        4                     -1.6900                      -1.450

           -4.7399999999999999769e-01  -4.4700000000000000084e-01  \
        0                       0.129                       0.427
        1                      -1.250                      -1.200
        2                      -0.531                      -0.394
        3                      -1.110                      -1.090
        4                      -0.546                      -0.527

                      ...              9.2500000000000000444e-01  \
        0           ...                          -0.593
        1           ...                          -0.816
        2           ...                           0.634
        3           ...                          -0.640
        4           ...                          -0.277

           5.160000000000000142e-01  3.439999999999999725e-01  \
        0                     0.452                   0.00785
        1                     1.570                   0.39400
        2                     0.111                   0.37100
        3                     0.485                   0.29500
        4                     0.699                   0.37100

           9.060000000000000275e-01  -1.129999999999999893e+00  \
        0                    -0.533                    -0.0789
        1                     1.340                    -1.1800
        2                     0.859                    -0.9930
        3                     0.403                    -1.1200
        4                     0.481                    -1.0600

           -5.520000000000000462e-01  5.530000000000000471e-01  \
        0                      0.705                     0.906
        1                     -2.700                    -0.926
        2                     -0.492                     0.363
        3                     -0.343                     0.468
        4                     -0.526                     0.550

           -4.169999999999999818e-01  2.560000000000000053e-01  \
        0                      0.216                   -0.0723
        1                     -2.650                   -0.0447
        2                      0.326                   -0.0528
        3                     -0.820                    0.4350
        4                     -0.284                    0.1550

           0.000000000000000000e+00
        0                       0.0
        1                       0.0
        2                       0.0
        3                       0.0
        4                       0.0

        [5 rows x 118 columns]
```

```
In [3]: df.describe()
```

```
Out[3]:        -1.439999999999999891e-01  -1.429999999999999882e-01  \
        count               69097.000000               69097.000000
        mean                   -0.000998                  -0.001317
```

```
std                     1.211814                   1.057315
min                  -316.000000                -147.000000
25%                    -0.064700                  -0.040900
50%                     0.024400                   0.103000
75%                     0.094200                   0.176000
max                     1.340000                   3.750000

       -1.160000000000000059e-01  -1.029999999999999943e-01  \
count                 69097.000000               69097.000000
mean                      0.000783                   0.000315
std                       1.025786                   1.033030
min                     -71.400000                 -81.400000
25%                      -0.253000                  -0.214000
50%                       0.095600                   0.085000
75%                       0.361000                   0.313000
max                       2.030000                   1.750000

       2.260000000000000064e-01  2.099999999999999922e-01  \
count                69097.000000              69097.000000
mean                    -0.003164                 -0.002645
std                      0.983603                  0.988942
min                     -3.080000                 -9.120000
25%                     -0.569000                 -0.553000
50%                     -0.099200                 -0.098800
75%                      0.450000                  0.434000
max                     28.000000                 26.900000

       -9.799999999999999822e-01  -7.800000000000000266e-01  \
count                 69097.000000               69097.000000
mean                      0.000860                  -0.000371
std                       0.999775                   0.998938
min                      -2.070000                  -1.910000
25%                      -0.532000                  -0.597000
50%                       0.013700                  -0.075300
75%                       0.582000                   0.513000
max                      11.200000                  22.200000

       -4.739999999999999769e-01  -4.470000000000000084e-01  \
count                 69097.000000               69097.000000
mean                      0.002183                   0.002589
std                       1.005921                   1.004499
min                      -1.570000                  -1.500000
25%                      -0.711000                  -0.728000
50%                      -0.227000                  -0.235000
75%                       0.488000                   0.498000
max                      16.700000                  22.500000

                 ...              9.250000000000000444e-01  \
count             ...                      69097.000000
mean              ...                          0.002552
std               ...                          1.003115
min               ...                        -15.200000
25%               ...                         -0.458000
50%               ...                         -0.102000
75%               ...                          0.322000
max               ...                         13.000000

       5.160000000000000142e-01  3.439999999999999725e-01  \
count              69097.000000              69097.000000
mean                  -0.001249                 -0.001666
std                    0.999683                  0.998781
min                   -7.460000                -32.900000
25%                   -0.491000                 -0.334000
50%                    0.161000                  0.205000
75%                    0.559000                  0.399000
max                    6.310000                 17.000000

       9.060000000000000275e-01  -1.129999999999999893e+00  \
count              69097.000000               69097.000000
mean                   0.002447                  -0.000894
std                    1.003225                   1.000271
min                   -7.780000                  -1.220000
25%                   -0.520000                  -0.857000
50%                   -0.269000                  -0.132000
75%                    0.134000                   0.645000
max                   10.600000                  17.700000

       -5.520000000000000462e-01  5.530000000000000471e-01  \
count               69097.000000              69097.000000
mean                    0.000008                 -0.001287
std                     0.999411                  1.000923
min                    -4.710000                 -5.450000
25%                    -0.247000                 -0.606000
50%                     0.321000                  0.269000
75%                     0.645000                  0.801000
max                     1.410000                  1.440000

       -4.169999999999999818e-01  2.560000000000000053e-01  \
count               69097.000000              69097.000000
mean                    0.000274                  0.000840
std                     0.999324                  1.024495
min                    -6.340000                -23.200000
25%                    -0.097800                 -0.259000
50%                     0.326000                 -0.142000
75%                     0.583000                  0.006700
max                     1.420000                 40.600000
```

```
       0.000000000000000000e+00
count                69097.000000
mean                     0.005919
std                      0.076709
min                      0.000000
25%                      0.000000
50%                      0.000000
75%                      0.000000
max                      1.000000

[8 rows x 118 columns]
```

```
In [4]: # Standardize the dataset
        X = df.iloc[:, :-1]
        y = df.iloc[:, -1]
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=

        X_train = (X_train - X_train.mean()) / X_train.std()
        X_test = (X_test - X_train.mean()) / X_train.std()

        X_train.shape, y_train.shape, X_test.shape, y_test.shape
```

```
Out[4]: ((51822, 117), (51822,), (17275, 117), (17275,))
```

```
In [5]: X_train.describe()
```

```
Out[5]:        -1.439999999999999891e-01  -1.429999999999999882e-01  \
        count               5.182200e+04               5.182200e+04
        mean               -1.784788e-16               2.960231e-17
        std                 1.000000e+00               1.000000e+00
        min                -2.262472e+02              -1.439716e+02
        25%                -4.436865e-02              -3.899734e-02
        50%                 1.942509e-02               1.022342e-01
        75%                 6.961522e-02               1.737315e-01
        max                 9.615820e-01               3.675138e+00

               -1.160000000000000059e-01  -1.029999999999999943e-01  \
        count               5.182200e+04               5.182200e+04
        mean               -2.973620e-18              -9.424320e-18
        std                 1.000000e+00               1.000000e+00
        min                -6.890080e+01              -7.938560e+01
        25%                -2.430020e-01              -2.082215e-01
        50%                 9.165048e-02               8.249608e-02
        75%                 3.456312e-01               3.037769e-01
        max                 1.957135e+00               1.705190e+00

               2.260000000000000064e-01  2.099999999999999922e-01  \
        count              5.182200e+04              5.182200e+04
```

```
mean              6.062501e-17              5.598355e-17
std               1.000000e+00              1.000000e+00
min              -3.115148e+00             -9.153742e+00
25%              -5.725497e-01             -5.517512e-01
50%              -9.644555e-02             -9.790508e-02
75%               4.596846e-01              4.382760e-01
max               2.836850e+01              2.701338e+01

       -9.799999999999999822e-01  -7.800000000000000266e-01  \
count               5.182200e+04               5.182200e+04
mean                2.056254e-17              -1.965846e-17
std                 1.000000e+00               1.000000e+00
min                -2.067937e+00              -1.908246e+00
25%                -5.326718e-01              -5.962096e-01
50%                 1.331122e-02              -7.549198e-02
75%                 5.790719e-01               5.149744e-01
max                 1.118709e+01               2.218406e+01

       -4.739999999999999769e-01  -4.470000000000000084e-01  \
count               5.182200e+04               5.182200e+04
mean               -1.369194e-17              -2.375897e-18
std                 1.000000e+00               1.000000e+00
min                -1.559753e+00              -1.493721e+00
25%                -7.059286e-01              -7.267595e-01
50%                -2.277771e-01              -2.383312e-01
75%                 4.848597e-01               4.948085e-01
max                 1.657378e+01               2.238057e+01

                  ...           -7.379999999999999893e-01  \
count             ...                        5.182200e+04
mean              ...                       -6.780626e-18
std               ...                        1.000000e+00
min               ...                       -1.939020e+00
25%               ...                       -6.410940e-01
50%               ...                       -6.167339e-02
75%               ...                        5.559156e-01
max               ...                        1.268888e+01

       9.250000000000000444e-01  5.160000000000000142e-01  \
count              5.182200e+04              5.182200e+04
mean              -2.451523e-17             -4.105867e-18
std                1.000000e+00              1.000000e+00
min               -1.500584e+01             -6.417456e+00
25%               -4.555016e-01             -4.881047e-01
50%               -1.041777e-01              1.597152e-01
75%                3.132661e-01              5.596040e-01
max                1.282375e+01              6.309006e+00
```

```
       3.4399999999999999725e-01  9.0600000000000000275e-01  \
count               5.182200e+04               5.182200e+04
mean               -7.341929e-18              -2.129309e-17
std                 1.000000e+00               1.000000e+00
min                -1.274658e+01              -6.573638e+00
25%                -3.396451e-01              -5.203783e-01
50%                 2.086718e-01              -2.700964e-01
75%                 4.039211e-01               1.305525e-01
max                 1.719840e+01               1.048114e+01

       -1.1299999999999999893e+00  -5.5200000000000000462e-01  \
count                5.182200e+04                5.182200e+04
mean                 5.221832e-17                4.601908e-17
std                  1.000000e+00                1.000000e+00
min                 -1.216458e+00               -4.696535e+00
25%                 -8.552714e-01               -2.425729e-01
50%                 -1.309030e-01                3.231522e-01
75%                  6.443508e-01                6.464236e-01
max                  1.586507e+01                1.399726e+00

       5.5300000000000000471e-01  -4.1699999999999999818e-01  \
count               5.182200e+04                5.182200e+04
mean                5.343733e-17                4.478641e-17
std                 1.000000e+00                1.000000e+00
min                -5.446999e+00               -6.336429e+00
25%                -6.056973e-01               -9.863111e-02
50%                 2.680619e-01                3.262189e-01
75%                 8.020676e-01                5.841278e-01
max                 1.443074e+00                1.350857e+00

       2.5600000000000000053e-01
count               5.182200e+04
mean                2.621268e-16
std                 1.000000e+00
min                -2.325851e+01
25%                -2.599988e-01
50%                -1.416979e-01
75%                 7.246089e-03
max                 3.749606e+01

[8 rows x 117 columns]
```

```
In [6]: X_test.describe()
```

```
Out[6]:        -1.4399999999999999891e-01  -1.4299999999999999882e-01  \
        count                17275.000000                17275.000000
        mean                     0.005101                    0.001880
        std                      0.147718                    1.159427
```

```
min                     -2.700000                  -147.000000
25%                     -0.063700                    -0.036950
50%                      0.025600                     0.105000
75%                      0.094800                     0.176000
max                      0.692000                     0.354000

       -1.160000000000000059e-01  -1.029999999999999943e-01  \
count                17275.000000               17275.000000
mean                    -0.002335                  -0.003269
std                      0.993605                   1.055627
min                    -67.000000                 -81.400000
25%                     -0.260000                  -0.219000
50%                      0.092000                   0.082000
75%                      0.361000                   0.312000
max                      1.820000                   1.550000

       2.260000000000000064e-01  2.099999999999999922e-01  \
count               17275.000000              17275.000000
mean                    0.001716                 -0.000099
std                     0.972808                  0.967692
min                    -2.970000                 -9.120000
25%                    -0.566000                 -0.553000
50%                    -0.094200                 -0.092300
75%                     0.454000                  0.438000
max                    24.200000                 23.300000

       -9.799999999999999822e-01  -7.800000000000000266e-01  \
count                17275.000000               17275.000000
mean                     0.002617                  -0.000426
std                      0.995725                   0.993559
min                     -2.060000                  -1.770000
25%                     -0.530000                  -0.597000
50%                      0.014200                  -0.072600
75%                      0.585000                   0.507000
max                     10.100000                  15.200000

       -4.739999999999999769e-01  -4.470000000000000084e-01  \
count                17275.000000               17275.000000
mean                     0.004258                   0.005597
std                      1.001117                   1.002221
min                     -1.570000                  -1.500000
25%                     -0.714000                  -0.727500
50%                     -0.224000                  -0.227000
75%                      0.483000                   0.497000
max                     15.300000                  11.500000

                   ...            -7.379999999999999893e-01  \
count              ...                         17275.000000
```

```
mean            ...                            0.000868
std             ...                            1.001180
min             ...                           -1.940000
25%             ...                           -0.653000
50%             ...                           -0.064600
75%             ...                            0.541000
max             ...                           11.700000

       9.250000000000000444e-01  5.160000000000000142e-01  \
count              17275.000000              17275.000000
mean                  -0.006485                 -0.002719
std                    0.971865                  0.997924
min                   -8.410000                 -7.460000
25%                   -0.465000                 -0.495000
50%                   -0.108000                  0.165000
75%                    0.320000                  0.560000
max                   10.700000                  5.500000

       3.439999999999999725e-01  9.060000000000000275e-01  \
count              17275.000000              17275.000000
mean                  -0.005862                 -0.005302
std                    1.029085                  0.979954
min                  -32.900000                 -7.780000
25%                   -0.330000                 -0.520000
50%                    0.203000                 -0.272000
75%                    0.399000                  0.127000
max                   12.400000                  8.950000

       -1.129999999999999893e+00  -5.520000000000000462e-01  \
count               17275.000000               17275.000000
mean                   -0.001169                   0.008673
std                     0.994330                   0.990811
min                    -1.220000                  -4.560000
25%                    -0.853000                  -0.251000
50%                    -0.130000                   0.325000
75%                     0.643000                   0.646000
max                    17.700000                   1.410000

       5.530000000000000471e-01  -4.169999999999999818e-01  \
count              17275.000000               17275.000000
mean                   0.004028                   0.005097
std                    1.003728                   0.996244
min                   -5.150000                  -6.180000
25%                   -0.599000                  -0.091500
50%                    0.280000                   0.328000
75%                    0.805000                   0.585000
max                    1.420000                   1.420000
```

```
       2.560000000000000053e-01
count              17275.000000
mean                   0.005348
std                    1.101653
min                  -16.900000
25%                   -0.257000
50%                   -0.142000
75%                    0.007340
max                   40.600000

[8 rows x 117 columns]
```

```
In [7]: reg_Cs = np.hstack((10.**np.arange(-5, 0), 10**np.arange(0, 6)))

        logit = LogisticRegressionCV(reg_Cs, cv=5)
        logit.fit(X_train, y_train)
        print("Logistic regression classifier on test set has accuracy:", logit.score(X_test, y
```

```
Logistic regression classifier on test set has accuracy: 0.995253256151
```

```
In [8]: y_test.values
        test_zeros = (y_test.values == 0).sum()
        test_ones = (y_test.values == 1).sum()
        test_total = y_test.values.shape[0]
        # test_zeros, test_ones, test_zeros+test_ones, test_total
        all_zero_accuracy = test_zeros/test_total
        print("A classifier that predicts 'normal' (label 0) on patients in the test set has ac
```

```
A classifier that predicts 'normal' (label 0) on patients in the test set has accuracy: 0.9946
```

2.3 Answer:

2.3.1 2.A. How does the fitted model compare with a classifier that predicts 'normal' (label 0) on all patients?

- Using logistic regression, we get accuracy score on test set 99.53%, whereas using a "zero for all" classifier, we get test score of 99.47%.
- We can see that these two scores are very close.

2.3.2 2.B. Do you think the difference in the classification accuracies are large enough to declare logistic regression as a better classifier than the all 0's classifier? Why or why not?

- No, the difference between the classification accuracies are not large at all. So we cannot declare logistic regression is a better classfier than the all 0's classfier, if we only look at the two scores.
- This is because the dataset contains more than 69000 candidate ROIs, of which only 409 are malignant, while the remaining are all normal. So the ratio of people who does not have

breast cancer is extremely higher than those with malignant tumor. The class labels are very imbalanced.

```
In [9]: print("Confusion matrix for the fitted logistic regression classifier:")
        print(confusion_matrix(y_test,logit.predict(X_test)))
```

```
Confusion matrix for the fitted logistic regression classifier:
[[17181     2]
 [   80    12]]
```

```
In [10]: all_zero_confusion = np.array([[test_zeros, 0], [test_ones, 0]])
         print("Confusion matrix for the all-zero classifier:")
         print(all_zero_confusion)
```

```
Confusion matrix for the all-zero classifier:
[[17183     0]
 [   92     0]]
```

2.4 Answer:

2.4.1 4. Using the entries of the confusion table compute the true positive rate and the true negative rate for the two classifiers. Explain what these evaluation metrics mean for the specific task of cancer detection. Based on the observed metrics, comment on whether the fitted model is better than the all 0's classifier.

Logistic Regression Classifier:

- True Positive Rate = $\frac{TP}{TP+FN} = \frac{12}{12+80} = 13.04\%$
- True Negative Rate = $\frac{TN}{TN+FP} = \frac{17181}{17181+2} = 99.99\%$

All-zero Classifier:

- True Positive Rate = $\frac{TP}{TP+FN} = \frac{0}{92+0} = 0\%$
- True Negative Rate = $\frac{TN}{TN+FP} = \frac{17183}{17183+0} = 100\%$
- True positive rate measures the proportion of positives that are correctly identified as such. In this case, true positive rate means the percentage of people with breast cancer who are correctly identified as having the condition.
- True negative rate measures the proportion of negatives that are correctly identified as such. In this case, true negative rate means the percentage of healthy people who are correctly identified as not having breast cancer.
- In this medical test condition, higher true positive rate is more important, because then we will be less likely to miss people who has breast cancer by telling them everything is fine.
- Since all-zero classifier tells everybody that they are healthy, it is very useless in this medical area.
- Therefore, the fitted logistic regression model is better thant the all 0's classifier.

2.4.2 5. What is the false positive rate of the fitted classifier, and how is it related to its true positive and true negative rate? Why is a classifier with high false positive rate undesirable for a cancer detection task?

- False Positive Rate of the fitted Logistic Regression Classfier = $\frac{FP}{FP+TN} = \frac{2}{2+17181} = 0.01\%$
- False Positive Rate = 1 - True Negative Rate
- False positive rate cannot be derived directly from true positive rate.
- For a caner detection task, high false positive rate is undesirable because it means false alarms, which may raise unnecessary worry and distress for healthy people. Those healthy people who get false positive results will go to hospital to do more tests, which is a waste of money and time.

2.5 Question 2: ROC Analysis

Another powerful diagnostic tool for class-imbalanced classification tasks is the Receiver Operating Characteristic (ROC) curve. Notice that the default logistic regression classifier in `sklearn` classifies a data point by thresholding the predicted class probability $\hat{P}(Y = 1)$ at 0.5. By using a different threshold, we can adjust the trade-off between the true positive rate (TPR) and false positive rate (FPR) of the classifier. The ROC curve allows us to visualize this trade-off across all possible thresholds.

1. Display the ROC curve for the fitted classifier on the *test set*. In the same plot, also display the ROC curve for the all 0's classifier. How do the two curves compare?

2. Compute the highest TPR that can be achieved by the classifier at each of the following FPR's, and the thresholds at which they are achieved. Based on your results, comment on how the threshold influences a classifier's FPR.

    - FPR = 0
    - FPR = 0.1
    - FPR = 0.5
    - FPR = 0.9

- Suppose a clinician told you that diagnosing a cancer patient as normal is *twice* as critical an error as diagnosing a normal patient as having cancer. Based on this information, what threshold would you recommend the clinician to use? What is the TPR and FPR of the classifier at this threshold?

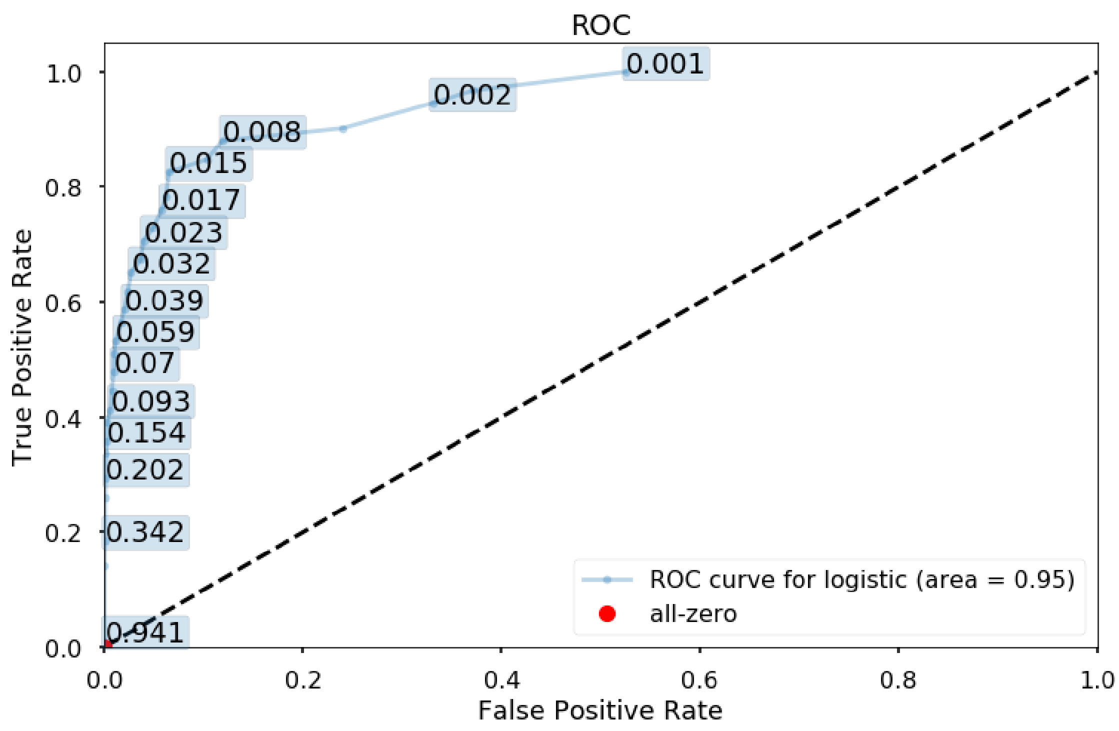
- Compute the area under the ROC curve (AUC) for both the fitted classifier and the all 0's classifier. How does the difference in the AUCs of the two classifiers compare with the difference between their classification accuracies in Question 1, Part 2(A)?

Hint: You may use the `metrics.roc_curve` function to compute the ROC curve for a classification model and the `metrics.roc_auc_score` function to compute the AUC for the model.

```
In [11]: # make_roc function from lab 7
         def make_roc(name, clf, ytest, xtest, ax=None, labe=5, proba=True, skip=0):
             initial=False
             if not ax:
                 ax=plt.gca()
```

```
            initial=True
        if proba:#for stuff like logistic regression
            fpr, tpr, thresholds=roc_curve(ytest, clf.predict_proba(xtest)[:,1])
        else:#for stuff like SVM
            fpr, tpr, thresholds=roc_curve(ytest, clf.decision_function(xtest))
        roc_auc = auc(fpr, tpr)
        if skip:
            l=fpr.shape[0]
            ax.plot(fpr[0:l:skip], tpr[0:l:skip], '.-', alpha=0.3, label='ROC curve for %
        else:
            ax.plot(fpr, tpr, '.-', alpha=0.3, label='ROC curve for %s (area = %0.2f)' %
        label_kwargs = {}
        label_kwargs['bbox'] = dict(
            boxstyle='round,pad=0.1', alpha=0.2,
        )
        if labe!=None:
            for k in range(0, fpr.shape[0],labe):
                #from https://gist.github.com/podshumok/c1d1c9394335d86255b8
                threshold = str(np.round(thresholds[k], 3))
                ax.annotate(threshold, (fpr[k], tpr[k]), **label_kwargs)
        if initial:
            ax.plot([0, 1], [0, 1], 'k--')
            ax.plot(0,0,'ro', label="all-zero")
            ax.set_xlim([0.0, 1.0])
            ax.set_ylim([0.0, 1.05])
            ax.set_xlabel('False Positive Rate')
            ax.set_ylabel('True Positive Rate')
            ax.set_title('ROC')
        ax.legend(loc="lower right")
        return ax

sns.set_context("poster")
plt.figure(figsize=(12,7.5))
ax=make_roc("logistic",logit, y_test, X_test, labe=10, skip=5)
```



2.6 Answer:

2.6.1 1. Display the ROC curve for the fitted classifier on the test set. In the same plot, also display the ROC curve for the all 0's classifier. How do the two curves compare?

- The ROC curve for logistic is clearly much better than the one for all-zero classifier, which covers about 0.93 area.
- The all o's classifier only gives a single point at (0,0), because it predicts everything to be negative, and hence both FPR and TPR will always be zero.

```
In [12]: fpr, tpr, thresholds=roc_curve(y_test, logit.predict_proba(X_test)[:,1])
         # fpr.shape, tpr.shape, thresholds.shape
         print("If FPR=0, the highest TPR is {}, and the threshold is {}"\
                .format(tpr[(np.abs(fpr-0)).argmin()], thresholds[(np.abs(fpr-0)).argmin()]))
         print("If FPR=0.1, the highest TPR is {}, and the threshold is {}"\
                .format(tpr[(np.abs(fpr-0.1)).argmin()], thresholds[(np.abs(fpr-0.1)).argmin()]
         print("If FPR=0.5, the highest TPR is {}, and the threshold is {}"\
                .format(tpr[(np.abs(fpr-0.5)).argmin()], thresholds[(np.abs(fpr-0.5)).argmin()]
         print("If FPR=0.9, the highest TPR is {}, and the threshold is {}"\
                .format(tpr[(np.abs(fpr-0.9)).argmin()], thresholds[(np.abs(fpr-0.9)).argmin()]
```

```
If FPR=0, the highest TPR is 0.010869565217391304, and the threshold is 0.9414417739229958
If FPR=0.1, the highest TPR is 0.8369565217391305, and the threshold is 0.009646251614429753
If FPR=0.5, the highest TPR is 0.9891304347826086, and the threshold is 0.0008238056503600531
If FPR=0.9, the highest TPR is 1.0, and the threshold is 2.155053140078295e-34
```

2.7 Answer:

2.7.1 2. Compute the highest TPR that can be achieved by the classifier at each of the following FPR's, and the thresholds at which they are achieved. Based on your results, comment on how the threshold influences a classifier's FPR.

- As we decrease the threshold from 1 to 0, the flase positive rate will increase, but we will also get a higher true positive rate. The change in threshold causes a trade-off between FPR and TPR.

2.8 Answer:

2.8.1 3. Suppose a clinician told you that diagnosing a cancer patient as normal is twice as critical an error as diagnosing a normal patient as having cancer. Based on this information, what threshold would you recommend the clinician to use? What is the TPR and FPR of the classifier at this threshold?

- Based on the clinician's point of view, a false negative is twice as critical an error as a false positive.
- Therefore, I will recommend trading a higher false positive rate for a higher true positive rate.
- My way to interpret twice is as the follows: Both FPR and non-TPR have some cost, but cost for non-TPR is twice more expensive. Therefore, we want to trade FPR for TPR, and the relation between them is about 2*FPR = 1-TPR.

```
In [13]: # print("For threshold=0.01, TPR is {}, and FPR is {}"\
         #       .format(tpr[(np.abs(thresholds-0.01)).argmin()], fpr[(np.abs(thresholds-0.01)
         tpr_penalty = [1-x for x in tpr]
         fpr_penalty = [x for x in fpr]
         dif = []
         for i, j in zip(tpr_penalty, fpr_penalty):
             d = abs(i - 2*j)
             dif.append(d)
         min_index = dif.index(min(dif))
         # print(min_index)
         # tpr[min_index], fpr[min_index], thresholds[min_index]
         print("For threshold={}, TPR is {}, and FPR is {}"\
                .format(thresholds[min_index], tpr[min_index], fpr[min_index]))
```

```
For threshold=0.01427641549682725, TPR is 0.8369565217391305, and FPR is 0.06832334283885236
```

- From the above calculation, I will recomment a threshold at about 0.014, which gives true positive rate (TPR) at 83.7%, whereas false positive rate (FPR) is at 6.83%.

2.9 Answer:

2.9.1 4. Compute the area under the ROC curve (AUC) for both the fitted classifier and the all 0's classifier. How does the difference in the AUCs of the two classifiers compare with the difference between their classification accuracies in Question 1, Part 2(A)?

```
In [14]: logit_auc = roc_auc_score(y_test, logit.predict_proba(X_test)[:,1])
         print("The area under the ROC curve for the fitted logistic classifier is:", logit_auc
         print("The area under the ROC curve for the all 0's classifier is: 0.5")
```

```
The area under the ROC curve for the fitted logistic classifier is: 0.945432037226
The area under the ROC curve for the all 0's classifier is: 0.5
```

2.10 Question 3: Missing data

In this problem you are given a different data set, `hw6_dataset_missing.csv`, that is similar to the one you used above (same column definitions and same conditions), however this data set contains missing values.

Note: be careful of reading/treating column names and row names in this data set as well, it *may* be different than the first data set.

1. Remove all observations that contain and missing values, split the dataset into a 75-25 train-test split, and fit the regularized logistic regression as in Question 1 (use `LogisticRegressionCV` again to retune). Report the overall classification rate and TPR in the test set.
2. Restart with a fresh copy of the data in `hw6_dataset_missing.csv` and impute the missing data via mean imputation. Split the data 75-25 and fit the regularized logistic regression model. Report the overall classification rate and TPR in the test set.

3. Again restart with a fresh copy of the data in `hw6_dataset_missing.csv` and impute the missing data via a model-based imputation method. Once again split the data 75-25 and fit the regularized logistic regression model. Report the overall classification rate and TPR in the test set.

4. Compare the results in the 3 previous parts of this problem. Prepare a paragraph (5-6 sentences) discussing the results, the computational complexity of the methods, and conjecture and explain why you get the results that you see.

```
In [15]: df2 = pd.read_csv('HW6_dataset_missing.csv', index_col=0)
         df2.shape
```

```
Out[15]: (24999, 118)
```

```
In [16]: df2_drop = df2.dropna(axis=0)
         df2_drop.shape
```

```
Out[16]: (1436, 118)
```

```
In [17]: X2 = df2_drop.iloc[:, :-1]
         y2 = df2_drop.iloc[:, -1]
         X_train2, X_test2, y_train2, y_test2 = train_test_split(X2, y2, test_size=0.25, randon
         X_train2.shape, y_train2.shape, X_test2.shape, y_test2.shape
```

```
Out[17]: ((1077, 117), (1077,), (359, 117), (359,))
```

```
In [18]: logit = LogisticRegressionCV(reg_Cs, cv=3) #data size after removing missing values i
         logit.fit(X_train2, y_train2)
         print("Logistic regression classifier on test set with missing values removed has accu
               logit.score(X_test2, y_test2))
         print("Confusion matrix for the fitted logistic regression classifier:")
         print(confusion_matrix(y_test2,logit.predict(X_test2)))
```

```
Logistic regression classifier on test set with missing values removed has accuracy: 1.0
Confusion matrix for the fitted logistic regression classifier:
[[359]]
```

2.11 Answer:

2.11.1 1. Remove all observations that contain and missing values. Report the overall classification rate and TPR in the test set.

- With missing values removed, the TPR calculated from the above confusion matrix is N/A, because there is no positive case, and hence zero divide by zero is undefined.

```
In [19]: df3 = df2.copy()
         for col in df3:
             df3[col] = df2[col].fillna(df2[col].mean())
         df3.shape
```

```
Out[19]: (24999, 118)
```

```
In [20]: X3 = df3.iloc[:, :-1]
         y3 = df3.iloc[:, -1]
         X_train3, X_test3, y_train3, y_test3 = train_test_split(X3, y3, test_size=0.25, randon
         X_train3.shape, y_train3.shape, X_test3.shape, y_test3.shape
```

```
Out[20]: ((18749, 117), (18749,), (6250, 117), (6250,))
```

```
In [21]: logit = LogisticRegressionCV(reg_Cs, cv=5)
         logit.fit(X_train3, y_train3)
         print("Logistic regression classifier on test set with missing values imputed has accu
               logit.score(X_test3, y_test3))
         print("Confusion matrix for the fitted logistic regression classifier:")
         print(confusion_matrix(y_test3,logit.predict(X_test3)))
```

```
Logistic regression classifier on test set with missing values imputed has accuracy: 0.99488
Confusion matrix for the fitted logistic regression classifier:
[[6212    1]
 [  31    6]]
```

2.12 Answer:

2.12.1 2. Restart with a fresh copy and impute the missing data via mean imputation. Report the overall classification rate and TPR in the test set.

- TPR = $\frac{TP}{TP+FN} = \frac{6}{6+31} = 16.22\%$

In [22]:
```
df4 = df2.copy()
nan_cols = df4.columns[df4.isnull().any()].tolist()
print("columns that have missing values are: ", nan_cols)
# df4.isnull().sum()
# df4.head()
```

```
columns that have missing values are:  ['93', '94', '95', '96', '97', '98', '99', '100', '101'
```

In [23]:
```
# indices = df4['93'].index[df4['93'].apply(np.isnan)].tolist()
# indices[0]
# df4.iloc[8,92]
# np.isnan(df4.iloc[8,92])
# inds = df4['93'].index[df4['93'].apply(np.isnan)]
```

In [24]:
```
# use knn with k=5
# randomly select from the nearest next 5 neighbors with equal probability
# Run this method three times till no more nan values
for t in range(3):
    for j in nan_cols:
        inds = df4[j].index[df4[j].apply(np.isnan)].tolist()
        j_int = int(j)-1
        for i in inds:
            if i+6 >= df4.shape[0]:
                df4.iloc[i,j_int] = df4.iloc[0,j_int]
            else:
                random_number = np.random.random()
                if random_number < 0.2:
                    if not np.isnan(df4.iloc[i+1, j_int]):
                        df4.iloc[i, j_int] = df4.iloc[i+1, j_int]
                    else:
                        df4.iloc[i, j_int] = df4.iloc[i+2, j_int]
                elif random_number >= 0.2 and random_number < 0.4:
                    if not np.isnan(df4.iloc[i+2, j_int]):
                        df4.iloc[i,j_int] = df4.iloc[i+2, j_int]
                    else:
                        df4.iloc[i, j_int] = df4.iloc[i+3, j_int]
                elif random_number >= 0.4 and random_number < 0.6:
                    if not np.isnan(df4.iloc[i+3, j_int]):
                        df4.iloc[i,j_int] = df4.iloc[i+3, j_int]
                    else:
                        df4.iloc[i, j_int] = df4.iloc[i+4, j_int]
```

```
                            elif random_number >= 0.6 and random_number < 0.8:
                                if not np.isnan(df4.iloc[i+4, j_int]):
                                    df4.iloc[i,j_int] = df4.iloc[i+4, j_int]
                                else:
                                    df4.iloc[i, j_int] = df4.iloc[i+5, j_int]
                            elif random_number >= 0.8:
                                if not np.isnan(df4.iloc[i+5, j_int]):
                                    df4.iloc[i,j_int] = df4.iloc[i+5, j_int]
                                else:
                                    df4.iloc[i, j_int] = df4.iloc[i+6, j_int]

        # check if there is any more nan values: want to see an empty list
        df4.columns[df4.isnull().any()].tolist()

Out[24]: []

In [25]: # df4.isnull().sum()

In [26]: X4 = df4.iloc[:, :-1]
         y4 = df4.iloc[:, -1]
         X_train4, X_test4, y_train4, y_test4 = train_test_split(X4, y4, test_size=0.25, rando
         X_train4.shape, y_train4.shape, X_test4.shape, y_test4.shape

Out[26]: ((18749, 117), (18749,), (6250, 117), (6250,))

In [27]: logit = LogisticRegressionCV(reg_Cs, cv=5)
         logit.fit(X_train4, y_train4)
         print("Logistic regression classifier on test set with missing values using knn model
               logit.score(X_test4, y_test4))
         print("Confusion matrix for the fitted logistic regression classifier:")
         print(confusion_matrix(y_test4,logit.predict(X_test4)))

Logistic regression classifier on test set with missing values using knn model imputation has 
Confusion matrix for the fitted logistic regression classifier:
[[6211    2]
 [  29    8]]
```

2.13 Answer:

2.13.1 3. Again restart with a fresh copy and impute the missing data via a model-based imputation method. Report the overall classification rate and TPR in the test set.

- TPR = $\frac{TP}{TP+FN} = \frac{8}{8+29} = 21.62\%$

2.14 Answer:

2.14.1 4. Compare the results in the 3 previous parts of this problem. Prepare a paragraph (5-6 sentences) discussing the results, the computational complexity of the methods, and conjecture and explain why you get the results that you see.

With missing values present in the dataset, we tried 3 methods: removing missing values, mean imputation and model-based (I chose knn model with k=5) imputation. In this problem, it is not a good idea to completely remove missing values, because doing this will significantly shrink the size of the dataset, and we can see then we have no more observations with patient who have breast cancer, and hence we cannot get any TPR information. Comparing the TPR in the test set, we can see that model-based imputation gives the highest true positive rate. This is because using model-based imputation plus some uncertainty, the value imputed will be more likely to represent real data. However, the computational complexity of the model-based imputation is much greater than the other two, and the TPR result is not that much higher than simple mean imputation. Therefore, I prefer use mean imputation in this problem than the other two methods.

2.15 APCOMP209a - Homework Question

This problem walks you through the derivation of the **likelihood equations** for a generalized linear model (GLM). Suppose that the random component of the GLM is in the univariate natural exponential family, so that

$$f(y_i|\theta_i) = h(y_i)e^{y_i\theta_i - b(\theta_i)}$$

Define the individual log-likelihood for each observation i as

$$l_i(\theta_i) \equiv \log f(y_i|\theta_i)$$

with linear predictor

$$\eta_i = x_i^T\beta = g(\mu_i)$$

for some link function g and where $\mu_i = E(Y_i)$.

1. Use the above expressions to write a simplified expression for the log-likelihood $l(\theta)$ for the entire dataset, $y_1, \dots, y_n$.
2. Use the chain rule to express $\frac{\partial l_i}{\partial \beta_j}$ in terms of the derivatives of l_i, θ_i, μ_i, and η_i. (*Hint*: Think carefully about which variables are related to which, and in what way. For example, for which of the above variables do you know the derivative with respect to β_j?)
3. Compute the derivatives for $\frac{\partial l_i}{\partial \theta_i}$ and $\frac{\partial \eta_i}{\partial \beta_j}$.
4. Express μ_i in terms of θ_i, and use this relationship to compute $\frac{\partial \theta_i}{\partial \mu_i}$. (*Hint*: Recall the cumulant function of a natural exponential family, and assume that you can write $\partial f/\partial g = (\partial g/\partial f)^{-1}$.)
5. Express η_i in terms of μ_i. Using the same hint as the above, compute $\frac{\partial \mu_i}{\partial \eta_i}$.
6. Put all of the above parts together to write an expression for $\frac{\partial l}{\partial \beta_j}$. Use matrix notation to write this expression as

   $$\nabla_\beta l(\beta) = XDV^{-1}(Y - \mu) = 0$$

   That is, compute the matrices D and V such that this equation holds.