cs109_hw6_submission

October 26, 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.

Enrollment Status: CS109A
Import libraries:

```
In [2]: 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
        import statsmodels.api as sm

        from sklearn.tree import export_graphviz
```

```
from IPython.display import Image
from IPython.display import display
%matplotlib inline
```

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.

2.2.1 Part 1.1

In [5]:
```
# part 1, split into train and test, 75-25 split
from sklearn.model_selection import train_test_split

random = np.random.seed(9001)
df = pd.read_csv('hw6_dataset.csv', header=None)
itrain, itest = train_test_split(range(df.shape[0]), train_size=0.75)

# create training and testing sets based on the split
data_train = df.iloc[itrain, :]
data_test = df.iloc[itest, :]

print(data_train.shape)
print(data_test.shape)
```

```
(51823, 118)
(17275, 118)
```

In [9]:
```
# split into X and Y now
cols = data_train.columns.values
train_cols = np.delete(cols,117)

# take only predictor columns
X_train = data_train[train_cols]
X_test = data_test[train_cols]

# take 0,1 column denoting normal or cancerous
Y_train = data_train[117]
Y_test = data_test[117]

X_train.head()
```

Out[9]:

	0	1	2	3	4	5	6	7	8	9	...	107	108	109	110	111	112	113	114	115	116
20593	-0.1580	-0.5480	-0.0497	-0.0602	0.000228	-0.00127	-1.1000	-1.0200	-0.6480	-0.733	...	0.716	-0.459	0.653	0.4060	0.975	-1.130				
23898	0.0720	0.0986	0.0137	0.0117	0.360000	0.37000	-0.4050	-0.4460	0.5300	0.124	...	-0.810	0.235	0.683	-1.1300	-0.122	-0.473				
34913	0.0822	0.1580	0.0784	0.0820	-0.232000	-0.28900	-0.0837	0.0716	-0.9570	-0.720	...	0.430	-0.117	0.206	-1.5000	-0.329	0.568				
12869	-0.1490	-0.0349	-1.0100	-0.8390	1.060000	0.99100	1.4200	1.0000	-0.0523	0.190	...	0.765	-0.138	-0.604	0.0937	-0.429	0.639				
25561	0.0370	0.1990	0.3130	0.2840	-1.130000	-1.08000	0.1600	0.2410	-0.3070	0.351	...	1.080	-0.409	-0.671	-0.2360	-0.199	0.752				

```
20593 -0.9920  0.252 -0.546  0.1330
23898  0.9440  1.100  0.553 -0.0555
34913  0.8350  0.712  0.844 -0.2750
12869  0.0303 -0.709  0.505 -0.1890
25561  0.2950 -0.160  0.874 -0.1790

[5 rows x 117 columns]
```

2.2.2 Part 1.2

```
In [10]: # fit a logistic regression using L2 regularization, report accuracy
         # regularization parameter tuned using cross validation

         # 5-fold cross validation
         logreg = LogisticRegressionCV(cv = 5)
         logreg.fit(X_train, Y_train)

         # make predictions based on model
         y_hat_logreg_train = logreg.predict_proba(X_train)[:,1]
         y_hat_logreg_test = logreg.predict_proba(X_test)[:,1]

In [12]: # report accuracy scores
         print("On Test Set:")
         print("logreg accuracy in test = %0.5f" % accuracy_score(Y_test, y_hat_logreg_test>0.5
         print("zeros accuracy in test = %0.5f" % accuracy_score(Y_test, np.zeros(len(Y_test))

On Test Set:
logreg accuracy in test = 0.99583
zeros accuracy in test = 0.99467
```

2.2.3 ANALYSIS

Report the accuracy of the classifier on the test set. Test accuracy of the logistic regression classifier is 0.99583.

How does the fitted model compare with a classifier that predicts 'normal' (label 0) on all patients? Test accuracy of the zeros predictor is 0.99467, so this is slightly worse than the logistic regression classifier, but they are both extremely high accuracies. We are surprised the accuracies are so close, but this result makes sense because of the high number of 0's in the data set.

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, just looking at the accuracies, I don't think the differences are large enough to favor either classifier since they are both very high classification rates, and differ by about 0.001. If we ended up with a test set with even fewer cancer positive cases (just based on the random split), the 0's classifier might even report a better accuracy score. That being said, we know that a predictor of all 0's will have no chance at accurately guessing a cancer positive case, so it is likely that logistic regression is better.

2.2.4 Part 1.3

```
In [14]: # part 3, compute confusion table
         # generate table for prediction and zeros, train and test
         train_conf = metrics.confusion_matrix(Y_train, y_hat_logreg_train>0.5)
         test_conf = metrics.confusion_matrix(Y_test, y_hat_logreg_test>0.5)
         train_zero_conf = metrics.confusion_matrix(Y_train, np.zeros(len(Y_train)))
         test_zero_conf = metrics.confusion_matrix(Y_test, np.zeros(len(Y_test)))

         print("Confusion Matrix for Training Data for Logistic")
         print(train_conf)
         print()
         print("Confusion Matrix for Training Data for Zeros")
         print(train_zero_conf)
         print()
         print("Confusion Matrix for Testing Data for Logistic")
         print(test_conf)
         print()
         print("Confusion Matrix for Testing Data for Zeros")
         print(test_zero_conf)
```

```
Confusion Matrix for Training Data for Logistic
[[51490    16]
 [  218    99]]

Confusion Matrix for Training Data for Zeros
[[51506     0]
 [  317     0]]

Confusion Matrix for Testing Data for Logistic
[[17177     6]
 [   66    26]]

Confusion Matrix for Testing Data for Zeros
[[17183     0]
 [   92     0]]
```

2.2.5 Part 1.4

```
In [15]: # pull values from the confusion tables for rate calculations (done in next cell)

         # from the logistic regression on the training set
         log_train_tn = train_conf[0][0]
         log_train_fp = train_conf[0][1]
         log_train_fn = train_conf[1][0]
         log_train_tp = train_conf[1][1]

         # from the logistic regression on the testing set
```

```
log_test_tn = test_conf[0][0]
log_test_fp = test_conf[0][1]
log_test_fn = test_conf[1][0]
log_test_tp = test_conf[1][1]

# from the zeros predictor on the training set
zero_train_tn = train_zero_conf[0][0]
zero_train_fp = train_zero_conf[0][1]
zero_train_fn = train_zero_conf[1][0]
zero_train_tp = train_zero_conf[1][1]

# from the zeros predictor on the testing set
zero_test_tn = test_zero_conf[0][0]
zero_test_fp = test_zero_conf[0][1]
zero_test_fn = test_zero_conf[1][0]
zero_test_tp = test_zero_conf[1][1]
```

In [19]:
```
# calculate and print out the true positive and true negative rates

# for the logistic regression on the training set
log_train_tpr = log_train_tp / (log_train_tp + log_train_fn)
log_train_tnr = log_train_tn / (log_train_tn + log_train_fp)

print("True positive rate for training data prediction: ", log_train_tpr)
print("True negative rate for training data prediction: ", log_train_tnr)
print()

# for the zeros predictor on the training set
zero_train_tpr = zero_train_tp / (zero_train_tp + zero_train_fn)
zero_train_tnr = zero_train_tn / (zero_train_tn + zero_train_fp)

print("True positive rate for training data zeros: ", zero_train_tpr)
print("True negative rate for training data zeros: ", zero_train_tnr)
print()

# for the logistic regression on the testing set
log_test_tpr = log_test_tp / (log_test_tp + log_test_fn)
log_test_tnr = log_test_tn / (log_test_tn + log_test_fp)

print("True positive rate for testing data prediction: ", log_test_tpr)
print("True negative rate for testing data prediction: ", log_test_tnr)
print()

# for the zeros predictor on the testing set
zero_test_tpr = zero_test_tp / (zero_test_tp + zero_test_fn)
zero_test_tnr = zero_test_tn / (zero_test_tn + zero_test_fp)

print("True positive rate for testing data prediction: ", zero_test_tpr)
```

```
        print("True negative rate for testing data prediction: ", zero_test_tnr)
```

```
True positive rate for training data prediction:  0.312302839117
True negative rate for training data prediction:  0.99968935658

True positive rate for training data zeros:  0.0
True negative rate for training data zeros:  1.0

True positive rate for testing data prediction:  0.282608695652
True negative rate for testing data prediction:  0.999650817669

True positive rate for testing data prediction:  0.0
True negative rate for testing data prediction:  1.0
```

2.2.6 ANALYSIS

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. FOR THE LOGISTIC CLASSIFIER

True positive rate for training data w logreg: 0.31230
True negative rate for training data w logreg: 0.99969
True positive rate for testing data w logreg: 0.28261
True negative rate for testing data w logreg: 0.99965
FOR THE ZEROS CLASSIFIER
True positive rate for training data w zeros: 0.0
True negative rate for training data w zeros: 1.0
True positive rate for testing data w zeros: 0.0
True negative rate for testing data w zeros: 1.0

True positive rate means the percentage of actual cancer positive people that are correctly labeled as cancerous by our predictor. True negative rate means the percentage of healthy people that are correctly labeled as healthy by our predictor. For cancer detection, TPR is especially important because our goal is to be able to detect cancer positive patients, so both of these models do a poor job (zeros classifier is not effective at all).

Based on the observed metrics, comment on whether the fitted model is better than the all 0's classifier. Yes, the logistic regression model is definitely better than the all 0's predictor. The 0's model has a TPR of 0.0, so it never guesses positive patients correctly, which is the main goal of a cancer test. Although the logreg model does not have a very high TPR (0.282), it is still much better than the 0 model (0). In this exercise, we care much less about whether we can accuraely identify a cancer-negative person, and more about being able to identify a cancer-positive person, so the 1.0 TNR for the zeros predictor isn't useful. The 0.0 TPR, on the other hand, is very alarming.

2.2.7 Part 1.5

```
In [18]: # calculate and print out the false positive (and also false negative) rates

         # for the logistic regression on the training set
```

```
log_train_fpr = log_train_fp / (log_train_fp + log_train_tn)
log_train_fnr = log_train_fn / (log_train_fn + log_train_tp)

print("False positive rate for training data prediction: ", log_train_fpr)
print("False negative rate for training data prediction: ", log_train_fnr)
print()

# for the zeros predictor on the training set
zero_train_fpr = zero_train_fp / (zero_train_fp + zero_train_tn)
zero_train_fnr = zero_train_fn / (zero_train_fn + zero_train_tp)

print("False positive rate for training data zeros: ", zero_train_fpr)
print("False negative rate for training data zeros: ", zero_train_fnr)
print()

# for the logistic regression on the testing set
log_test_fpr = log_test_fp / (log_test_fp + log_test_tn)
log_test_fnr = log_test_fn / (log_test_fn + log_test_tp)

print("False positive rate for testing data prediction: ", log_test_fpr)
print("False negative rate for testing data prediction: ", log_test_fnr)
print()

# for the zeros predictor on the testing set
zero_test_fpr = zero_test_fp / (zero_test_fp + zero_test_tn)
zero_test_fnr = zero_test_fn / (zero_test_fn + zero_test_tp)

print("False positive rate for testing data zeros: ", zero_test_fpr)
print("False negative rate for testing data zeros: ", zero_test_fnr)
```

```
False positive rate for training data prediction:  0.000310643420184
False negative rate for training data prediction:  0.687697160883

False positive rate for training data zeros:  0.0
False negative rate for training data zeros:  1.0

False positive rate for testing data prediction:  0.000349182331374
False negative rate for testing data prediction:  0.717391304348

False positive rate for testing data zeros:  0.0
False negative rate for testing data zeros:  1.0
```

2.2.8 ANALYSIS

What is the false positive rate of the fitted classifier, and how is it related to its true positive and true negative rate? False positive rate for training data prediction: 0.00031

False positive rate for training data zeros: 0.0

False positive rate for testing data prediction: 0.00035

False positive rate for testing data zeros: 0.0

False positive rate (FPR) is the percentage of cancer-negative patients that you incorrectly predict as cancer positive. FPR = 1 - TNR based on observing the setup of a confusion matrix. As for the relationship between FPR and TPR, these two metrics are usually positively correlated. If you have a high TPR, this means you are predicting a high number of positives in general (unless the model is incredibly accurate), so FPR will also likely be higher. However, in the above case for the logreg model, the TPR is lower in the test set while the FPR is higher. This represents the case in which there are an overwhelming number of one result (0's, or normal, for this example), leading to a high number of false positives in the test set since our data was not fit accurately enough.

Why is a classifier with high false positive rate undesirable for a cancer detection task? A classifier with high FPR is undesirable because of the high cost of a positive diagnosis. If we falsely predict positives frequently, these people will get second opinions, tests, treatments, etc. and this is all expensive. These people will also have unnecessary stress. Still, we would say it is better for cancer predictions to predict false postives than to predict false negatives, because false negative means a patient with cancer will not get treatment.

2.3 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

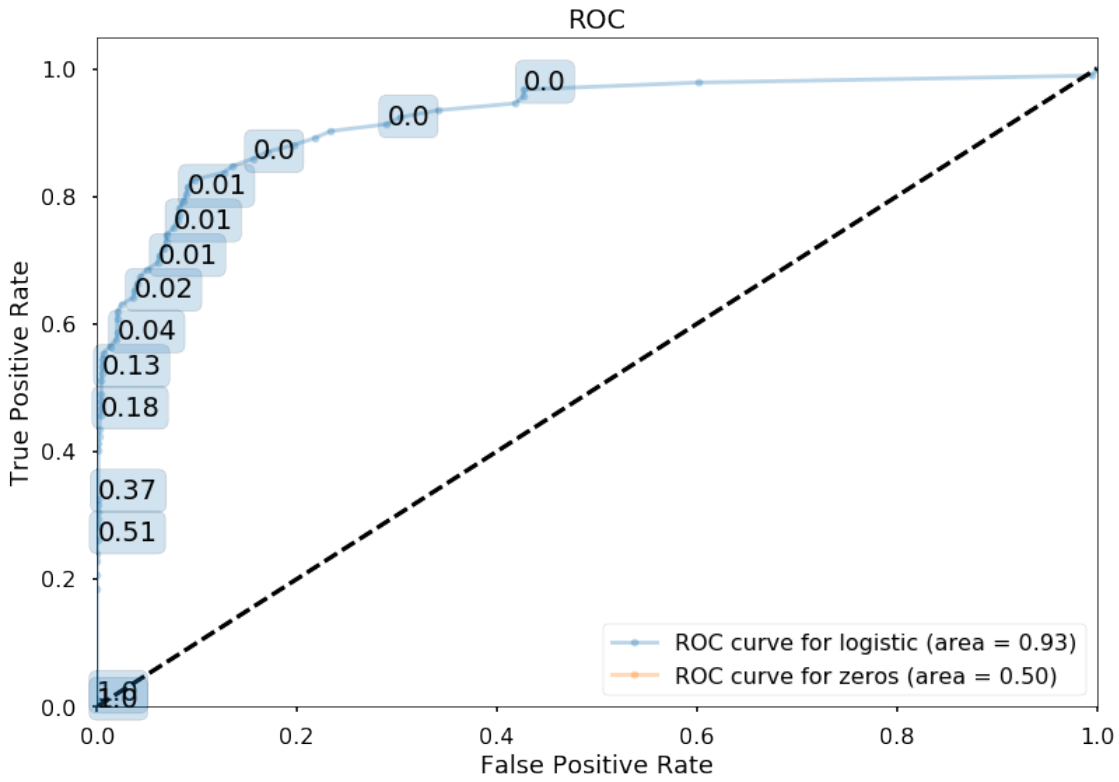
- Suppose a clinician told you that diagnosing a cancer patient as normal is *twice* as critical an error as diagnosing a normal patient as having cancer. Based on this information, what threshold would you recommend the clinician to use? What is the TPR and FPR of the classifier at this threshold?
-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

2.3.1 Part 2.1

In [23]:
```
# CODE TAKEN FROM LAB 7 CODE FOR GENERATING ROC CURVE!

from sklearn.metrics import roc_curve, auc

# takes in pred, which is a prediction (rather than a model)
def make_roc(name, pred, 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, pred)
    else:#for stuff like SVM, we won't see this here
        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 %s
    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=0.3', 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], 2))
            ax.annotate(threshold, (fpr[k], tpr[k]), **label_kwargs)
    if initial:
        ax.plot([0, 1], [0, 1], 'k--')
        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 arrays as well for later analysis
    return ax, fpr, tpr, thresholds
```

In [24]:
```
# display ROC curve for test set and for 0's classifier
import seaborn.apionly as sns

sns.set_context("poster")
# get fpr tpr and threshold as well
ax, fpr_1, tpr_1, thresholds_1=make_roc("logistic",logreg.predict_proba(X_test)[:,1],
ax, fpr_2, tpr_2, thresholds_2=make_roc("zeros",np.zeros(len(Y_test)), Y_test, X_test
```



2.3.2 ANALYSIS

How do the two curves compare? The logistic regression classifier ROC curve looks pretty standard, relative to the zeros classifier. The logreg curve does have a very sharp rise (and then a gradual flattening on top) because FPR = 1 - TNR, and the high number of 0's (cancer-negative people) in the data set means that TNR will be very high, ~0.99, for most corresponding TPR's (FPR is therefore very low, ~0, for most corresponding TPR's). For the zeros predictor, we can only have one case, a point at (0,0), as depicted, since we can't get a nonzero TPR or FPR without ever predicting a 1.

2.3.3 Part 2.2

```
In [22]: # part 2, compute highest TPR

         # collect the highest TPR and threshold for each FPR
         highest_tprs = []
         highest_thresholds = []

         # iterate through FPRS
         for fpr_goal in [0.0, 0.1, 0.5, 0.9]:
             for index, fpr in enumerate(fpr_1):
```

```
            # once we find the first FPR > goal, store the previous value
            if fpr > fpr_goal:
                highest_tprs.append(tpr_1[index-1])
                highest_thresholds.append(thresholds_1[index-1])
                break

    print("Highest TPR at FPR of 0.0: {}  \nCorresponding Threshold: {}".format(highest_tp
    print()
    print("Highest TPR at FPR of 0.1: {}  \nCorresponding Threshold: {}".format(highest_tp
    print()
    print("Highest TPR at FPR of 0.5: {}  \nCorresponding Threshold: {}".format(highest_tp
    print()
    print("Highest TPR at FPR of 0.9: {}  \nCorresponding Threshold: {}".format(highest_tp
```

```
Highest TPR at FPR of 0.0: 0.18478260869565216
Corresponding Threshold: 0.687817928852388

Highest TPR at FPR of 0.1: 0.8369565217391305
Corresponding Threshold: 0.0066056601429968775

Highest TPR at FPR of 0.5: 0.9782608695652174
Corresponding Threshold: 0.0003510132688753773

Highest TPR at FPR of 0.9: 0.9891304347826086
Corresponding Threshold: 9.23914653083274e-05
```

2.3.4 ANALYSIS

Based on your results, comment on how the threshold influences a classifier's FPR. The above results show us that as our target FPR goes up (from 0 to 0.9), the corresponding threshold is going down. Intuitively, as the threshold goes down, we will predict many more positives overall since the requirements to predict a positive wil be slacker, so we will see both TPR and FPR go up. In other words, as our threshold decreases, we are more likely to correctly identify cancer positive cases, but also more likely to predict a high number of positives incorrectly (FPR). If we increase the threshold, the opposite would occur (lower TPR's and FPR's).

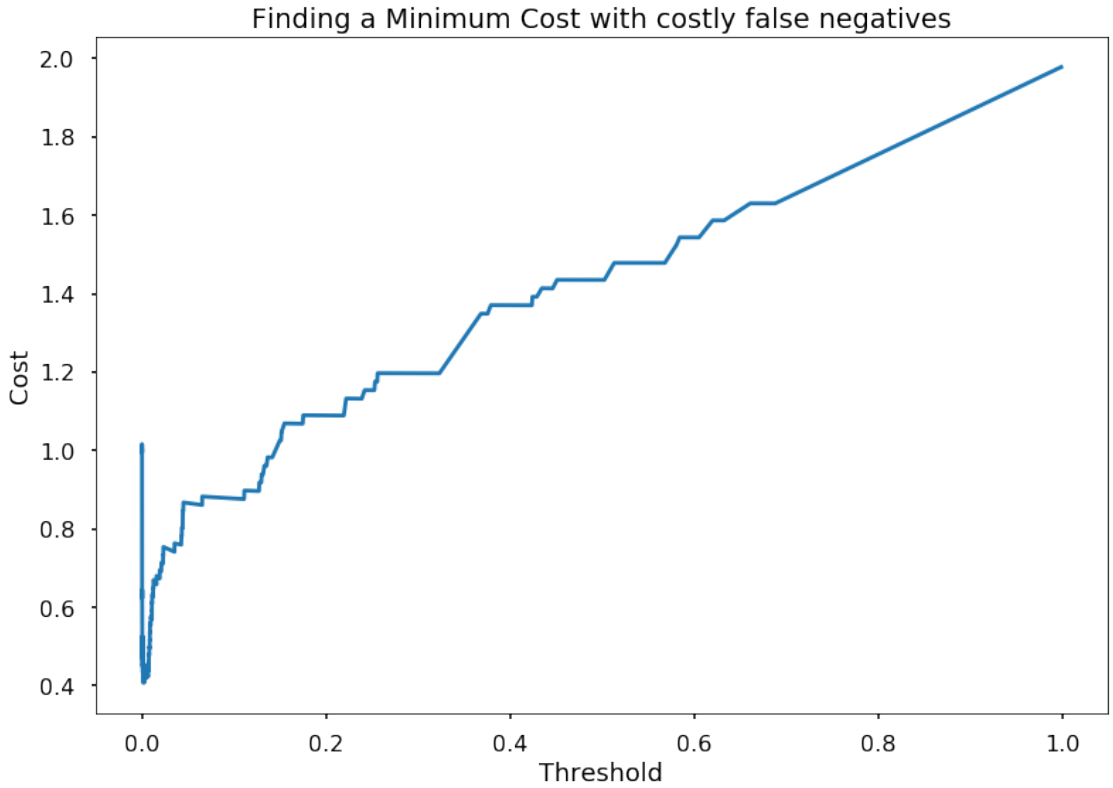
2.3.5 Part 2.3

Methodology: Overall, our goal is to minimize the total number of incorrect predictions, since we are trying to build the most accurate model possible. We know we generally want to minimize FNR and FPR, and once we are told FNR's are twice as costly, we can scale this term by 2x in our minimization function.

In [26]:
```
# here we want to minimize 2*FNR + FPR

    # generate cost function, minimize it
    fnr = 1-tpr_1
    cost = 2*fnr + fpr_1
```

```
min_index = np.argmin(cost)

# plot threshold vs cost
plt.plot(thresholds_1, cost)
plt.xlabel("Threshold")
plt.ylabel("Cost")
plt.title("Finding a Minimum Cost with costly false negatives")

print("Threshold: ", thresholds_1[min_index])
print("TPR: ", tpr_1[min_index])
print("FPR: ", fpr_1[min_index])
```

```
Threshold:  0.00151873509718
TPR:  0.913043478261
FPR:  0.233486585579
```



2.3.6 ANALYSIS

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? With this information, I would intuitively expect

a low threshold to encourage a high number of positive predictions so that we don't miss many positive predictions (low FNR). After writing the cost function as 2 * FNR + FPR, and then minimizing this function, we found that it was minimized at a threshold of 0.00152. This is low, like I expected! We can see here that the FNR is 0.08697 and FPR is 0.23349, and because of our cost function and how it scales FNRx2, I would expect FPR to be greater than FNR in order to get a minimum on cost, so this is also what I expected.

What is the TPR and FPR of the classifier at this threshold? At the threshold of 0.00152,
TPR: 0.91304
FPR: 0.23349

2.3.7 Part 2.4

```
In [28]: # compute area under ROC curve for classifier and 0's

         print("From Graph Above:")
         print("Area under LogReg Classifier Curve: 0.93")
         print("Area under Zeros Predictor Curve: 0.50")
```

```
From Graph Above:
Area under LogReg Classifier Curve: 0.93
Area under Zeros Predictor Curve: 0.50
```

2.3.8 ANALYSIS

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)? The AUC for the logistic predictor is 0.93. The AUC for the zeros predictor is 0.50. We found these values as a part of making the ROC functions. This shows a very clear, and massive difference between the values. In Q1, part 2A, we found that the classification accuracies were almost identical, giving us trouble determining which was best. Here, we have such a clear indicator that the logistic predictor is much better, since it has a much higher AUC. High AUC indicates a strong ability for the predictor to distinguish between the two classes, so the logistic model does well here, and the zeros predictor does not, even though it had a high classification accuracy.

2.4 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.

2.4.1 Part 3.1

In [29]:
```
# remove observations containing missing values
import copy

random = np.random.seed(9001)

# reload data
df_missing = pd.read_csv('hw6_dataset_missing.csv')
df_m0 = df_missing.drop('Unnamed: 0', axis = 1)
df_m1 = copy.deepcopy(df_m0)

# drop rows with NaN's
df_m1 = df_m1.dropna(axis = 0)

# split into 75-25 split and fit logistic model, report classification rate and TPR
itrain_m1, itest_m1 = train_test_split(range(df_m1.shape[0]), train_size=0.75)

# now split into test and train, as well as predictions and cancer +-
data_train_m1 = df_m1.iloc[itrain_m1, :]
data_test_m1 = df_m1.iloc[itest_m1, :]

cols = data_train_m1.columns.values
train_cols = np.delete(cols,117)

X_train_m1 = data_train_m1[train_cols]
X_test_m1 = data_test_m1[train_cols]

Y_train_m1 = data_train_m1['type']
Y_test_m1 = data_test_m1['type']
```

In [32]:
```
# fit the logistic regression with 5-fold cross validation and report accuracy
log_regression_m1 = LogisticRegressionCV(cv = 5, fit_intercept=True)
log_regression_m1.fit(X_train_m1, Y_train_m1)
y_hat_logreg_train_m1 = log_regression_m1.predict_proba(X_train_m1)[:,1]
```

```
        y_hat_logreg_test_m1 = log_regression_m1.predict_proba(X_test_m1)[:,1]

        print("DROPPING ROWS WITH NAN")
        print("  LogReg Accuracy in Test = %0.5f" % accuracy_score(Y_test_m1, y_hat_logreg_tes
        print()

        # get confusion matrix, then calculate rates
        test_cm_m1 = metrics.confusion_matrix(Y_test_m1, y_hat_logreg_test_m1 > 0.5)

        print(test_cm_m1)
        print()

        tp2_m1 = test_cm_m1[1][1]
        fn2_m1 = test_cm_m1[1][0]
        log_test_tpr_m1 = tp2_m1 / (tp2_m1 + fn2_m1)
        print("TPR for testing data prediction: ", log_test_tpr_m1)
```

```
/Users/georgeh85/anaconda/lib/python3.6/site-packages/sklearn/model_selection/_split.py:581: Wa
  % (min_groups, self.n_splits)), Warning)


DROPPING ROWS WITH NAN
  LogReg Accuracy in Test = 0.99721

[[358   0]
 [  1   0]]

TPR for testing data prediction:  0.0
```

2.4.2 Part 3.2

```
In [36]: # part 2, impute missing data via mean imputation. Split and fit model, report rates

        df_m2 = copy.deepcopy(df_m0)
        train_cols_m2 = list(df_m2.columns.values)
        train_cols_m2.remove('type')

        # impute in the mean value with fillna
        for col_name in train_cols:
            df_m2[col_name] = df_m0[col_name].fillna(df_m0[col_name].mean())

        # split into train and test, then predictors and target
        random = np.random.seed(9001)
        itrain2, itest2 = train_test_split(range(df_m2.shape[0]), train_size=0.75)
        data_train_m2 = df_m2.iloc[itrain2, :]
        data_test_m2 = df_m2.iloc[itest2, :]
```

```
X_train_m2 = data_train_m2[train_cols]
X_test_m2 = data_test_m2[train_cols]
Y_train_m2 = data_train_m2['type']
Y_test_m2 = data_test_m2['type']

# 5-fold cross validation
logreg_m2 = LogisticRegressionCV(cv = 5)#fit_intercept=True)
logreg_m2.fit(X_train_m2, Y_train_m2)
y_hat_logreg_train_m2 = logreg_m2.predict_proba(X_train_m2)[:,1]
y_hat_logreg_test_m2 = logreg_m2.predict_proba(X_test_m2)[:,1]

print("MEAN IMPUTATION MODEL")
print("LogReg Accuracy in Test = %0.5f" % accuracy_score(Y_test_m2, y_hat_logreg_test_
print()

# get confusion matrix, then calculate rates
test_conf_m2 = metrics.confusion_matrix(Y_test_m2, y_hat_logreg_test_m2 > 0.5)
print(test_conf_m2)
print()

true_pos_m2 = test_conf_m2[1][1]
false_negs_m2 = test_conf_m2[1][0]
pred_test_true_pos_rate_m2 = true_pos_m2 / (true_pos_m2 + false_negs_m2)
print("TPR for testing data prediction: ", pred_test_true_pos_rate_m2)
```

```
MEAN IMPUTATION MODEL
LogReg Accuracy in Test = 0.99552

[[6212    3]
 [  25   10]]

TPR for testing data prediction:  0.285714285714
```

2.4.3 Part 3.3

In [37]:
```
# part 3, impute via model-based imputtion. Split, fit, report.

# reload data
df_m3 = copy.deepcopy(df_m0)

# find indices of columns with NaNs and without
full_cols = []
missing_cols = []
for i in range(1,118):
    if (df_m3[str(i)].isnull().sum() == 0):
        full_cols.append(str(i))
    else:
```

```
            missing_cols.append(str(i))

    X_pred = df_m3[full_cols]
    X_missing = df_m3[missing_cols]
```

In [38]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LinearRegression

# first loop, impute values training on columns without NaNs.

for col_name in missing_cols:

    # get X train and test data for the linear model
    ds = pd.isnull(X_missing[col_name])
    ds_na = X_pred[ds]
    ds_filled = X_pred[[not x for x in ds]]

    # get Y train data for linear model
    cur_col = X_missing[col_name]
    cur_col_filled = cur_col[[not x for x in ds]]

    # train model and generate prediction
    regress = LinearRegression()
    regress.fit(ds_filled, cur_col_filled)
    pred = regress.predict(ds_na)

    # fill in df_m3 NaN with our prediction
    df_m3.ix[np.isnan(cur_col), col_name] = pred
```

```
/Users/georgeh85/anaconda/lib/python3.6/site-packages/ipykernel_launcher.py:23: DeprecationWarn
.ix is deprecated. Please use
.loc for label based indexing or
.iloc for positional indexing

See the documentation here:
http://pandas.pydata.org/pandas-docs/stable/indexing.html#deprecate_ix
```

In [39]:
```
# reload data
df_m4 = copy.deepcopy(df_m0)

# predictor columns
all_cols = list(df_m4.columns.values)
all_cols.remove("type")

# second loop, impute based on results of first loop
```

```
        for col_name in missing_cols:

            # train on all columns except the current one
            train_cols = copy.deepcopy(all_cols)
            train_cols.remove(str(col_name))
            training_data = df_m3[train_cols]

            # get X data for linear model using whole df_m3 (already has imputed values)
            ds = pd.isnull(df_m4[col_name])
            ds_filled = training_data[[not x for x in ds]]
            ds_na = training_data[ds]

            # get Y data for linear model
            cur_col = df_m4[col_name]
            cur_col_filled = cur_col[[not x for x in ds]]

            # train model and generate prediction
            regress = LinearRegression()
            regress.fit(ds_filled, cur_col_filled)
            pred = regress.predict(ds_na)

            # fill in df_m4 with more complete prediction
            df_m4.ix[np.isnan(cur_col), col_name] = pred

        # show that the values change from df_m3 to df_m4
        print(sum(df_m3['95']))
        print(sum(df_m4['95']))
```

```
/Users/georgeh85/anaconda/lib/python3.6/site-packages/ipykernel_launcher.py:32: DeprecationWarn
.ix is deprecated. Please use
.loc for label based indexing or
.iloc for positional indexing

See the documentation here:
http://pandas.pydata.org/pandas-docs/stable/indexing.html#deprecate_ix


106.041796093
107.270155519
```

```
In [41]: # split into train and test, then predictors and target
        random = np.random.seed(9001)
        itrain3, itest3 = train_test_split(range(df_m4.shape[0]), train_size=0.75)
        data_train4 = df_m4.iloc[itrain2, :]
        data_test4 = df_m4.iloc[itest2, :]
```

```
# get train_cols
train_cols_m3 = list(df_m4.columns.values)
train_cols_m3.remove('type')

X_train_4 = data_train4[train_cols]
X_test_4 = data_test4[train_cols]
Y_train_4 = data_train4['type']
Y_test_4 = data_test4['type']

# 5-fold cross validation
logreg4 = LogisticRegressionCV(cv = 5)#fit_intercept=True)
logreg4.fit(X_train_4, Y_train_4)
y_hat_logreg_train_4 = logreg4.predict_proba(X_train_4)[:,1]
y_hat_logreg_test_4 = logreg4.predict_proba(X_test_4)[:,1]

print("MODEL IMPUTATION METHOD")
print("logit single predictor accuracy in test = %0.5f" % accuracy_score(Y_test_4, y_l
print()

# get confusion matrix, then calculate rates
test_conf4 = metrics.confusion_matrix(Y_test_4, y_hat_logreg_test_4>0.5)
print(test_conf4)
print()

true_pos = test_conf4[1][1]
false_negs = test_conf4[1][0]
pred_test_true_pos_rate = true_pos / (true_pos + false_negs)
print("True positive rate for testing data prediction: ", pred_test_true_pos_rate)
```

```
MODEL IMPUTATION METHOD
logit single predictor accuracy in test = 0.99552

[[6214    1]
 [  27    8]]

True positive rate for testing data prediction:  0.228571428571
```

2.5 ANALYSIS

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. The method in which we drop all observations (rows) that have missing values is the one that gives us the best overall classification rate for the test set, at 0.99721, but it also gives us a TPR of 0 since we have dropped all of the actual cancer observations except one (we have one false negative), leading us to believe that classifcation rate is once again not a useful measure of the effectiveness of a model. On the other hand, we notice that the mean imputation and model-based imputation methods give us the same classification rates,

0.99552, and test TPR's of 0.28571 and 0.22857, respectively. These similar results make sense because, in columns with missing data, the model-based method will equally weight each observation and reported value in coming up with a prediction model, and this equal-weighting ends up being very close to simply taking the mean of the the reported values. Based on having the highest TPR, we think the mean-imputation method works best here. In terms of computational complexity, the first method (drop NaNs) is the fastest because we run through every observation once, simply checking for NaNs and dropping the entire row if one is found, while the second method (mean imputation) is slower because we run through every observation twice, once to check for missing values and the second time to calculate the mean of the existing values if we found any missing ones. The third method (model-based) is the slowest because, in the way we formulated it, you need to run through the data twice just to find which columns and rows contain data, and then perform linear regression (which depends on both the number of predictors and number of observations).

2.6 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, $y_1, \ldots, y_n$.

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.

7. If we use the canonical link function, how do your answers to part (6) simplify?

8. Finally, compute the above likelihood equations in the case of logistic regression, and show that this is equivalent to the solution given in lecture.

In [ ]: