46-926, Fall 2017: Homework #5

Due 5:20PM, Monday, February 19, 2018.

Homework should be submitted electronically on canvas before the deadline. Please submit your homework as a single file (pdf or html). We recommend using a jupyter notebook to prepare your homework.

Note: For this homework, try to do all problems in python.

Please post any questions on the Piazza discussion board.

1. *Fun with ROC curves and AUC!*

   Begin with the marketing data set from HW4 and the glm model that you fit at the beginning of the problem. You can just steal the code from the homework solutions if you like, to make sure that everything works.

   For each point in the test data, compute the predicted probabilities. From these probabilities, compute the logit transformation:

   $$\log\left(\frac{p(x_i)}{1-p(x_i)}\right)$$

   Plotting these scores on the logit scale will make for nicer pictures. It won't change the ROC curve or the classifications from thresholding, since the logit is just a monotonic function.

   (a) First, just make a histogram of your logit scores.

   > **Hint:** The plot and axis objects have a `hist` method. You can set the number of bins with the `bins` argument.

   (b) Now make separate histograms of the score for observations from each class. Stack these two histograms vertically in the same figure. Make sure to title each with the correct class, and to force the x-axis limits to be the same on both plots.

   You should see that the score has a different distribution for each class of observation. We are able to use our score to classify because the distribution of the score for class 'yes' tends to be larger than the distribution of the score for class 'no'.

Hint: You can set the x limits with the `range` argument of `hist`, which takes a two-element list or tuple.

You can set titles with the `set_title` method of the axis object.

(c) Suppose that we let $f_0(x)$ denote the distribution of the score for class '`no`', and $f_1(x)$ denote the distribution of the score for class '`yes`'. Furthermore, suppose we decide to threshold our scores at cutoff T, and classify all points with score $> T$ into class '`yes`'.

Write out the True Positive Rate (TPR) and False Positive Rate (FPR = 1 − specificity) in terms of intergrals of these two distributions, f_0, f_1. Redraw your two histograms, this time adding in a vertical line at $T = -1.5$. What part of the new plots correspond to the True Positive Rate and False Positive Rate? (You can describe this in words since labeling it your pdf might be hard.)

Hint: You can add vertical lines with the `axvline` method of the axis object.

(d) It is convenient to think about your score in terms of these class distributions. It will also let us prove a nice fact about ROC curves!

Write out the Area Under the Curve (AUC) in terms of an integral involving the TPR(T) and FPR(T).

Hints:

Realize that you are looking at a curve parametrized by $(FPR(T), TPR(T))$.

Remember from calculus: For a parametric curve x=f(t), y=g(t), the area under the curve is $\int_{t_0}^{t_f} g(t)f'(t)dt$. In your case, $t_0 = \infty$ and $t_f = -\infty$ (flipped because big T corresponds to small x-axis values).

(e) Simplify this expression to get it in terms of $\mathbb{P}(S_1 > S_0)$, if S_0 were the score of a random point from class '`no`' and S_1 were the score of a random point drawn from class '`yes`'.

When you succeed, you will have proven that the AUC is the probability that your score successfully ranks a random point from class 1 higher than a random point from class 0!

Hints:

As you simplify, try to introduce an indicator of the event $\{S_1 > S_0\}$. Remember that S_1 is drawn from f_1 and S_0 is drawn from f_0.

2. *Do-it-yourself sensitivity-specificity.* We will do a little more exploration of ROC curves and loss in this problem. As an example, we again consider the scores from logistic regression on the marketing data set.

   Note: All ROC curves in this problem should be produced by hand, rather than with a built-in python ROC function.

   (a) Write a function that takes in a boolean vector of guessed categories, a boolean vector of true categories, a loss for false positives and a loss for false negatives, and produces sensitivity, specificity, and total loss. You may find it useful to use boolean expressions like `(truth)&(~estimate)` in computing the desired quantities (where ~ is the elementwise "not" operator). The function should be laid out like this:

   ```
   #This function should take in numpy logical arrays for estimate and truth
   #and single values for the losses
   def eval_estimate(estimate, truth, loss_FP, loss_FN):
       #Things that you write go here
       return(sens, spec, loss)
   ```

   To check your function, run it on the scores from logistic regression thresholded at -1. Set $L_{FP} = 5$ (loss for a false positive) and $L_{FN} = 100$ (loss for a false negative) as in class, here and for the remainder of this problem.

   (b) Apply your function to all the possible thresholds in your problem. These are just the midpoints between each pair of logit values. This gives all possible classifications that can be obtained by your scores. Make a plot showing all of the sensitivity-specificity pairs that you obtain. To get a curve that looks like a typical ROC curve, you'll actually want to plot $1 -$ specificity on the x-axis. Add a diagonal line to compare to.

   You may use the following code to produce breakpoints and apply your function if you like:

   ```
   unique_values = np.sort(np.unique(logits))
   midpoints = (unique_values[0:(len(unique_values)-1)]+
                unique_values[1:len(unique_values)])/2.0
   sens, spec, loss = np.vectorize( lambda x: eval_estimate(logits>x,
      np.array(y_test, dtype=bool), 5, 100))(midpoints)
   ```

(c) Color the points on your previous plot by the total loss that you computed for each point. Given a vector of `values`, you can color your points by those values in python with code like

```
from matplotlib import cm #For nice colors
plt.scatter(xvalues, yvalues, c=values, cmap=cm.inferno)
```

This will show you how your loss varies over your curve (with red values being better losses). I recommend using the negative of your loss values as the color values, so that bright colors indicate good losses.

(d) Find the minimum loss point on your curve (see `np.argmin`). Figure out its sensitivity and specificity. Plot your roc curve again without colors, and plot this minimizing point on top in a different color.

Draw a line on top of this curve which passes through that point and has slope given by $\frac{N}{P} \cdot \frac{L_{FP}}{L_{FN}}$, where N, P are the total number of negative and positive observations in the test set, and L_{FP} and L_{FN} are the False Positive and False Negative losses.

What do you notice?

Hint: To add a point to a plot, just call `scatter` with a single coordinate. You can set the color with `color` and scale the point size with `s`. You may have to make `s` quite large to be visible.

```
plt.scatter(x_coord, y_coord, color='green', s=200)
```

Hint: Plotting lines by slope and intercept is weirdly annoying in python. I stole this code from the internet to make it easier:

```
def abline(slope, intercept):
    """Plot a line from slope and intercept"""
    axes = plt.gca()
    x_vals = np.array(axes.get_xlim())
    y_vals = intercept + slope * x_vals
    plt.plot(x_vals, y_vals, '--')
```

(source `https://stackoverflow.com/a/43811762`)

(e) Prove this formula!

i. Write out the total loss as a function of N, P, the False Positive Rate (FPR), the False Negative Rate (FNR), and L_{FP}, L_{FN}.

ii. Rewrite FPR and FNR as integrals of f_0 and f_1, as in Problem 1c, with some set threshold T.

iii. Take a derivative of this expression with respect to T and set it equal to 0. Solve for $f_1(T)/f_0(T)$, the slope of the tangent to the parametric curve at the minimizing point.

3. Here we continue the last problem, making comparisons between different methods and different thresholds.

   (a) In your last homework, you derived the optimal threshold on the true probabilities for a given set of losses. Continuing with your logistic regression and from the last problem and the same losses (5 and 100), compute the optimal probability threshold for this problem.

   What is your loss at this optimal threshold? How does it compare to the minimum loss that you found in problem 2? (**Hint:** You can reuse your `eval_estimate` function.)

   (b) Starting with your ROC curve from (2d), add one more point for the sensitivity/specificity of your ideal point. How does it compare to the point you already plotted for your true minimum? Give a reason why these might not agree completely, and why the loss for your ideal classifier is not actually the best.

   (c) Make a calibration plot for your logistic classifier. You can use the `calibration_curve` function from `sklearn.calibration` and then plot the returned vectors.

   You do not need to plot error bars (since this function makes them more difficult to get). However, plot a histogram of the probabilities from your logistic regression on a second subfigure below your calibration plot. Force the histogram axes to match the calibration plot x axis.

   Where is your classifier well-calibrated? Where does it appear to have poor calibration, and why do you think this is?

   (d) Now we'll compare your GAM model from last time with this logistic model. Obtain another vector of predicted probabilities from the GAM model. Copy the relevant pieces of your code from problem 2 to calculate another ROC curve for this model. Plot both ROC curves together, along with their respective points of minimum loss. Draw the corresponding tangent lines for both curves (given by your loss ratios and population ratios as in problem 2).

   What do you notice about these optimal points?

   Hint: You should really just be reusing code for this problem.

4. **One more reasonably light problem coming tomorrow.**