jgiebas_HW4

February 11, 2018

1 46-926, Statistical Machine Learning 1: Homework 4

Author : Jordan Giebas *Due Date*: Februrary 11th, 2018

```
In [3]: import warnings
        warnings.filterwarnings('ignore')
```

1.1 Question 1: *Best Classification for asymmetric losses*

If we assume that we know the joint distribution of (X, Y), we saw in class that the optimal classifier is given by the function,

$$f(x) = \begin{cases} 1 & \mathbb{P}(Y = 1|X = x) > 0.5 \\ 0 & otherwise \end{cases}$$

Suppose instead our loss is summarized in the following table,

1.1.1 Part (a)

Again assume that we know the joint distribution (X, Y). Find the best possible classification function $f(x)$ in a derivation similar to class.

Following the idea from class, we would like to solve the following optimization problem:

$$\begin{aligned} \hat{f}(x) &= \arg\min_g \sum_{k=0}^{1} \mathcal{L}(k, g)\mathbb{P}(Y = k|X = x) \\ &= \arg\min_g L_{k,g}\mathbb{1}_{\{k \neq g\}}\mathbb{P}(Y = k|X = x) \\ &= \min L_{0,1}\mathbb{P}(Y = 0|X = x) + L_{1,0}\mathbb{P}(Y = 1|X = x) \end{aligned}$$

Hence, we choose our function $\hat{f}(x)$ in such a way to minimize this quantity. Consider the following choice,

$$\hat{f}(x) = \begin{cases} 1 & L_{1,0}\mathbb{P}(Y = 1|X = x) \geq L_{0,1}\mathbb{P}(Y = 0|X = x) \\ 0 & otherwise \end{cases}$$

The above function is guarenteed to minimize the expected loss. Surely, if $L_{1,0}\mathbb{P}(Y = 1|X = x) \geq L_{0,1}\mathbb{P}(Y = 0|X = x)$, then we should impose that $f(x) = 1$ so that the second term in the equation vanishes leaving us with the lesser loss. The other direction follows from this logic as

well. **Note:** (a) I'm not necessarily sure how I would solve this if we weren't in a binary world and (b) I'm not sure how to exactly minimize that argument and arrive at the function presented through algebra, the above only offers a logical argument.

1.1.2 Part (b)

Suppose that $L_{1,0} > L_{0,1}$, how is the classification rule defined above different from the 0-1 loss defined in class?

As mentioned above, in the 0-1 case our classifier is given by,

$$f(x) = \begin{cases} 1 & \mathbb{P}\big(Y = 1|X = x\big) > 0.5 \\ 0 & otherwise \end{cases}$$

Note the obvious, that $0.5 = \frac{1}{2} = \frac{1}{1+1}$. Notice that this resembles a weighted average, and this nice case is essentially a consequence of the symmetry inherent in the problem considered in class (i.e. $L_{i,j} = \mathbb{1}_{i \neq j}$).

In our present setting, we have a similar notion of a weighted average amongst the loss coefficients. Consider the estimated function derived in part(a), and assume that $L_{1,0} > L_{0,1}$. For $\hat{f}(x) = 1$, it follows that $L_{1,0}\mathbb{P}(Y = 1|X = x) \geq L_{0,1}\mathbb{P}(Y = 0|X = x)$. Let $\alpha := \frac{L_{1,0}}{L_{0,1}}$, $p = \mathbb{P}(Y = 1|X = x)$ and note that $1 - p = \mathbb{P}(Y = 0|X = x)$. Then,

$$L_{1,0}\mathbb{P}(Y = 1|X = x) \geq L_{0,1}\mathbb{P}(Y = 0|X = x) := \alpha p \geq 1 - p$$

It follows that,

$$\begin{aligned} \alpha p \geq 1 - p &\implies (\alpha + 1)p \geq 1 - p \\ &\implies p \geq \frac{1}{\alpha + 1} \\ &\implies p \geq \frac{L_{1,0}}{L_{1,0} + L_{0,1}} \end{aligned}$$

Hence, we may our definition of $\hat{f}(x)$ as such,

$$\hat{f}(x) = \begin{cases} 1 & \mathbb{P}(Y = 1|X = x) \geq \frac{L_{1,0}}{L_{1,0}+L_{0,1}} \\ 0 & otherwise \end{cases}$$

In the symmetric case where $L_{i,j} = \mathbb{1}_{i \neq j}$, the rule above reduces to the fundamental case presented in class.

Note: I noted above that I wasn't sure how to minimize the thing algebraically, and I'm still unsure. However, the results above are quite sensible and I can easily see how this would extend to a target space with a dimension greater than two.

1.1.3 Part (c)

Using the above notion of a weighted average, this becomes an easy exercise to determine the new rule for the logistic loss function. Recall in the symmetric case, we chose the rule $\langle x, \beta^T \rangle \geq 0$ (where $\langle x, y \rangle$ represent the inner product of x, y). We chose this rule by the following argument,

$$\mathbb{P}(Y = 1|X = x) = \frac{exp\{\langle x, \beta^T \rangle\}}{1 + exp\{\langle x, \beta^T \rangle\}} \geq 0.5$$

Solving

$$\frac{exp\{\langle x, \beta^T \rangle\}}{1 + exp\{\langle x, \beta^T \rangle\}} = 0.5$$

we find that $\langle x, \beta^T \rangle = 0$.

In a similar fashion, we now solve the equation,

$$\frac{exp\{\langle x, \beta^T \rangle\}}{1 + exp\{\langle x, \beta^T \rangle\}} = \frac{L_{1,0}}{L_{1,0} + L_{0,1}}$$

Upon doing the algebra, we see that

$$\langle x, \beta^T \rangle = log(\frac{L_{1,0}}{L_{0,1}})$$

, and this becomes our new *linear* decision boundary.

Hence, our estimated function is now

$$\hat{f}(x) = \begin{cases} 1 & \langle x, \beta^T \rangle \geq log(\frac{L_{1,0}}{L_{0,1}}) \\ 0 & otherwise \end{cases}$$

1.2 Question 2: *Prediction with Marketing Data*

Using a family of covariates regarding customer data and whether or not they invested, we would like to make a prediction whether a new customer will invest in the product or not.

In [4]:
```
# Load necessary modules
import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import sklearn
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_curve
%matplotlib inline

# Load the DataFrame
df = pd.read_csv('marketing.csv')

# Adjust for categorical data
df.y = np.where(df.y == 'yes', 1, 0)
df = pd.get_dummies(df)
df.tail()
```

Out[4]:

	age	balance	y	job_admin.	job_blue-collar	job_entrepreneur	\
45206	51	825	1	0	0	0	
45207	71	1729	1	0	0	0	
45208	72	5715	1	0	0	0	

```
45209   57      668  0           0                 1                 0
45210   37     2971  0           0                 0                 1

       job_housemaid  job_management  job_retired  job_self-employed  \
45206              0               0            0                  0
45207              0               0            1                  0
45208              0               0            1                  0
45209              0               0            0                  0
45210              0               0            0                  0

         ...     education_primary  education_secondary  education_tertiary  \
45206    ...                     0                   0                   1
45207    ...                     1                   0                   0
45208    ...                     0                   1                   0
45209    ...                     0                   1                   0
45210    ...                     0                   1                   0

       education_unknown  default_no  default_yes  housing_no  housing_yes  \
45206                  0           1            0           1            0
45207                  0           1            0           1            0
45208                  0           1            0           1            0
45209                  0           1            0           1            0
45210                  0           1            0           1            0

       loan_no  loan_yes
45206        1         0
45207        1         0
45208        1         0
45209        1         0
45210        1         0

[5 rows x 28 columns]
```

In [5]:
```
# Split the dataset into training and test data - as specified
X = df.drop('y', axis=1)
y = df.y
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.33,random_state=1)
```

1.2.1 Part (a)

Estimate the proportion of successes on the training data (i.e. the *base rate*) - this will be useful to keep in mind, as a 'benchmark'.

In [6]:
```
base_rate = y_train.sum()/len(y_train)
print("The base rate is:", base_rate*100.0,"%")
```

```
The base rate is: 11.7823776039 %
```

1.2.2 Part (b)

Fit a logistic regression to the training data using all of the features using `LogisticRegression` from `sklearn` with $C = 100000$.

```
In [7]: # Fit the linear model
        lmfit = LogisticRegression(C=100000).fit(X_train, y_train)

        # Perform the prediction
        y_pred = lmfit.predict(X_test)
```

```
In [8]: # Percent of 0's, 1's respectively from predicition
        print("Prop. of 0: ", y_pred.tolist().count(0)/len(y_pred.tolist())*100.0)
        print("Prop. of 1: ", y_pred.tolist().count(1)/len(y_pred.tolist())*100.0)
```

```
Prop. of 0:  99.97989276139411
Prop. of 1:  0.020107238605898123
```

```
In [9]: print("The misclassification rate is: ", np.mean(y_test!=y_pred)*100.0, "%")
```

```
The misclassification rate is:  11.5482573727 %
```

1.2.3 Part (c)

Using the 'silly' classifier that classifies each observation as 'no' (i.e. 0)

```
In [10]: silly = np.zeros(len(y_pred))
         print("The misclassification rate using the silly classifier is: ", np.mean(silly!=y_
```

```
The misclassification rate using the silly classifier is:  0.0201072386059 %
```

The silly classifier is noticeably better! Using the logisitic classifier we are incorrect about 11.5% of the time, whereas using the silly classifier we're wrong only 0.02% of the time! Wow, we suck!! Yes, this is truly discouraging for our classifier. **It seems that no matter the type/description of the client, they all say no!**

1.2.4 Part (d)

Using the logisitic model, find the 1000 clients that are most likely to score 'yes' (i.e have the highest probability that $Y = 1$)

```
In [11]: X_new = X_test.copy()
         X_new['client_prob'] = lmfit.predict_proba(X_test)[:,1]
         X_new['y_test'] = y_test
         X_new = X_new.sort_values(by='client_prob', ascending=False)
         good_custos = X_new.head(1000)
         print("Percent of clients with highest probability that actually say yes: %f" % (np.me
```

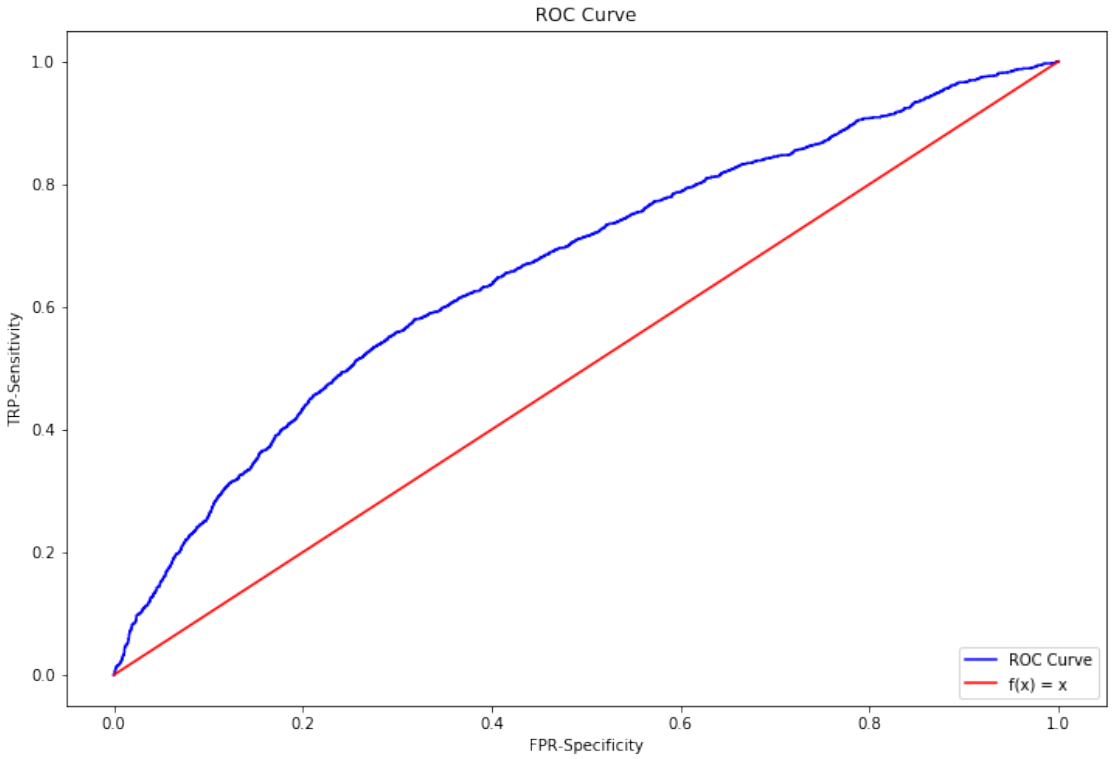
```
Percent of clients with highest probability that actually say yes: 27.900000
```

The percentage of customers that actually say yes out of the 1000 identified as having the highest probability is 27%! We would expect the base rate of 11.78% if we were pulling a random sample of 1000, so yes our model seems more useful!

1.2.5 Part (e)

An introduction to looking at the ROC curve, i.e. the curve plotting sensitivity vs. specificity. More to be discussed in class.

```
In [12]: # Follow the code in homework
         fpr, trp, _ = roc_curve(X_new.y_test, X_new.client_prob)

In [13]: matplotlib.rcParams['figure.figsize'] = (12,8)

         x = np.linspace(0,1.0,1000)
         plt.title("ROC Curve")
         plt.xlabel("FPR-Specificity")
         plt.ylabel("TRP-Sensitivity")
         plt.plot(fpr, trp, 'b-', label='ROC Curve')
         plt.plot(x, x, 'r-', label='f(x) = x')
         plt.legend(loc='lower right')

Out[13]: <matplotlib.legend.Legend at 0x111198358>
```



The above plot clearly shows that our ROC curve is above the diagonal (the equivalent of the random classifier) for all values of specificity. Hence, I think the interpretation is that we're making better guesses, True Positives, for any given value of True Negatives.

1.3 Question 3

Continuing with the market data example: let's improve the calssifier.

1.3.1 Part (a)

We doubt the that log odds of the `balance` and `age` variables grow linearly. Fit a Logistic GAM to the data, and we use smoothing splines to model the effect of balance and age. The partial dependence plots are plotted below.

```
In [14]: %%capture
         from pygam import LogisticGAM

         gam = LogisticGAM().gridsearch(X_train,y_train,lam = np.logspace(-1, 5, 30))
```

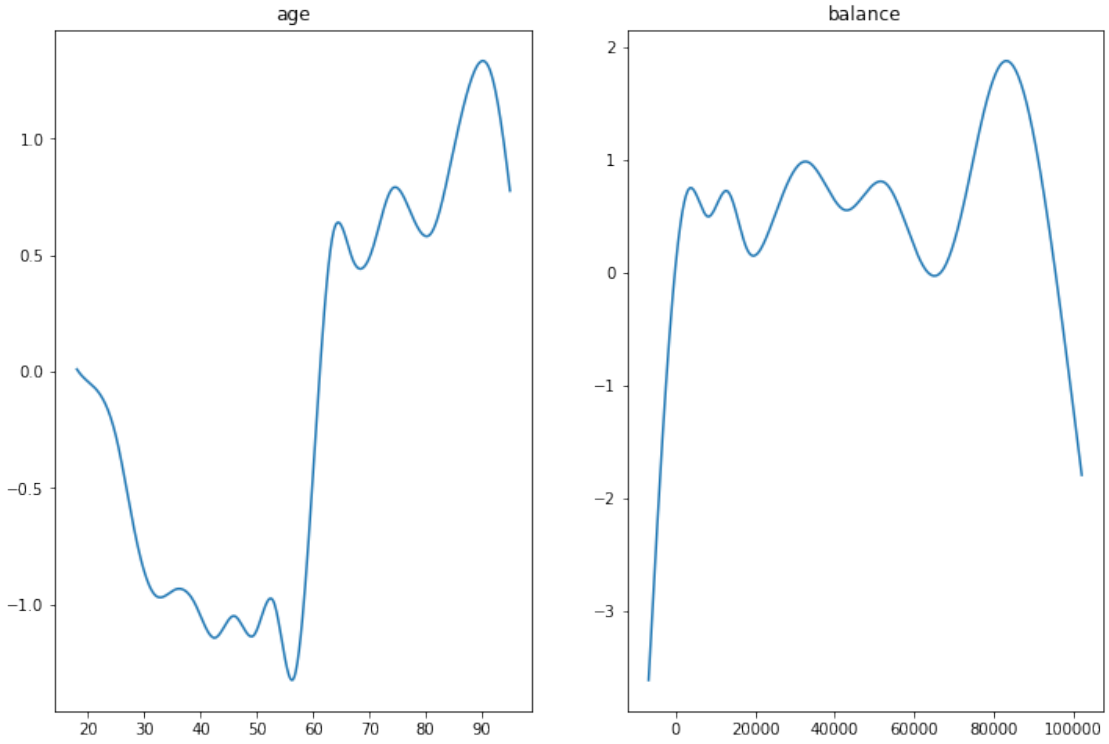
```
100% (30 of 30) |########################| Elapsed Time: 0:01:59 Time: 0:01:59
```

```
In [15]: gam.lam
```

```
Out[15]: 0.41753189365604015
```

```
In [16]: yhat_gam = gam.predict(X_test)
```

```
In [17]: from pygam.utils import generate_X_grid

         XX = generate_X_grid(gam)

         fig, axs = plt.subplots(1, 2)
         titles = X_test.columns[[0,1]].tolist()

         for j, ax in enumerate(axs):

             pdep, _ = gam.partial_dependence(XX, feature=j+1, width=0.95)

             ax.plot(XX[:,j], pdep)
             ax.set_title(titles[j])

         plt.show()
```



1.3.2 Part (b)

Evalute the predictions from the logistic GAM in terms of misclassifications, and calculate the proportion of scucesses in the top ranked 1000 teste observations. How does the performance of these metrics compare with the logistic regression case?

```
In [16]: print("Misclassification for LogisticGAM: ", np.mean(y_test != yhat_gam)*100.0, "%")
```

```
Misclassification for LogisticGAM:  11.7426273458 %
```

```
In [17]: X_new_2 = X_test.copy()
         X_new_2['client_prob'] = gam.predict_proba(X_test)
         X_new_2['y_test'] = y_test
         X_new_2 = X_new_2.sort_values(by='client_prob', ascending=False)
         good_custos_2 = X_new_2.head(1000)
         print("Percent of clients with highest probability that actually say yes: %f" % (np.me
```

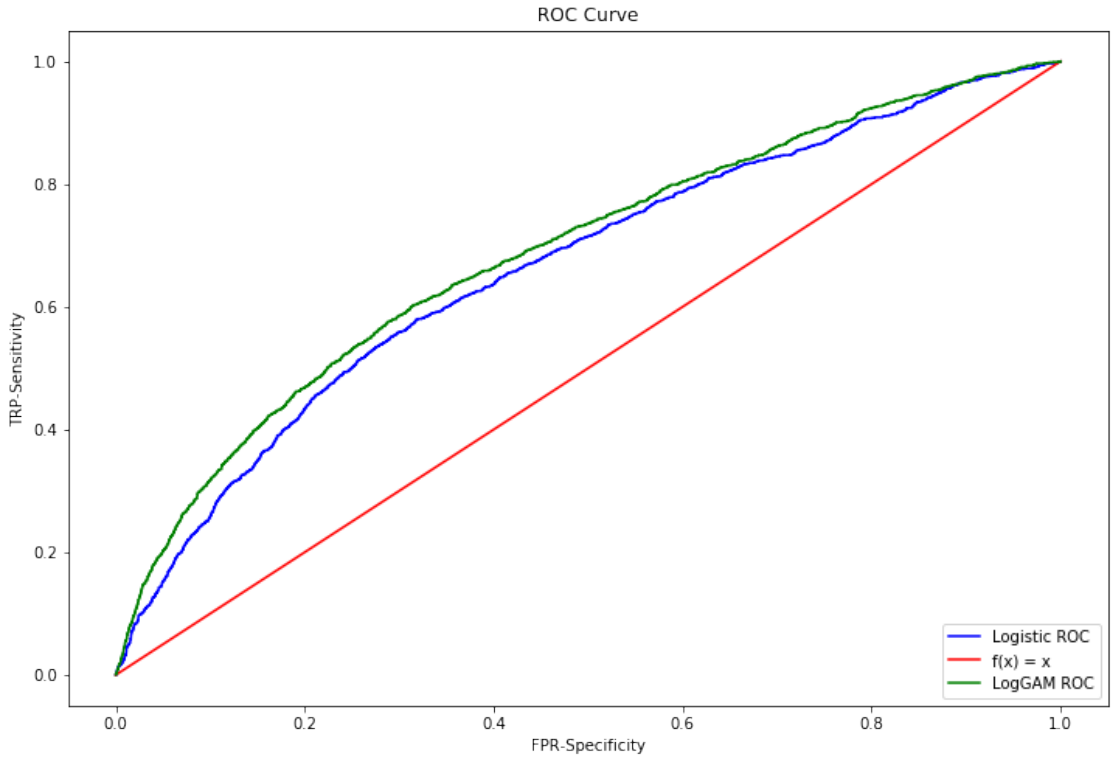
```
Percent of clients with highest probability that actually say yes: 34.500000
```

The misclassification rate is roughly the same as the base rate seen previously. However, when we re-selected the 1000 potential clients with the highest probability of saying 'yes', we see that the proportion of these clients that actually say yes is 34.5%. This is an improvement in 8.5% from the logisitic case!

1.3.3 Part (c)

Plot the ROC curve plot once again, superimposing the results from the first logistic case, this case, and the diagonal.

```
In [18]: fpr_gam, trp_gam, _ = roc_curve(X_new_2.y_test, X_new_2.client_prob)

         matplotlib.rcParams['figure.figsize'] = (12,8)

         x = np.linspace(0,1.0,1000)
         plt.title("ROC Curve")
         plt.xlabel("FPR-Specificity")
         plt.ylabel("TRP-Sensitivity")
         plt.plot(fpr, trp, 'b-', label='Logistic ROC')
         plt.plot(x, x, 'r-', label='f(x) = x')
         plt.plot(fpr_gam, trp_gam, 'g-', label='LogGAM ROC')
         plt.legend(loc='lower right')

Out[18]: <matplotlib.legend.Legend at 0x11718eb38>
```



This is beautiful. The plot above suggests that for any given value of True Negatives, the logistic GAM will provide more True Positives than using the more elementary logistic case.