Machine Learning HW4

Ze Yang (zey@andrew.cmu.edu)

February 11, 2018

---

1 Asymmetric Classification Loss

(i) As we have derived in class, the expected prediction error is

$$\begin{aligned} EPE &= \mathbb{E}\left[\mathcal{L}(Y, \hat{f}(X)\right] \\ &= \mathbb{E}\left[\mathbb{E}\left[\mathcal{L}(Y, \hat{f}(X))\Big|X\right]\right] \\ &= \mathbb{E}\left[\mathbb{E}\left[\sum_{k\in\mathcal{G}} \mathcal{L}(k, \hat{f}(x))\mathbb{P}\left(Y=k|X=x\right)\right]\right] \end{aligned} \tag{1}$$

In this problem, $\mathcal{G} = \{0, 1\}$; $\mathcal{L}(k, \hat{k}) = L_{k\hat{k}}$. Minimizing pointwise inside the expectation:

$$\begin{aligned} \hat{f}(x) &= \underset{\hat{k}\in\mathcal{G}}{\operatorname{argmin}} \sum_{k\in\mathcal{G}} \mathcal{L}(k, \hat{k})\mathbb{P}\left(Y=k|X=x\right) \\ &= \underset{\hat{k}\in\mathcal{G}}{\operatorname{argmin}} \sum_{k\in\mathcal{G}} L_{k\hat{k}}\mathbb{P}\left(Y=k|X=x\right) \\ &= \underset{\hat{k}\in\{0,1\}}{\operatorname{argmin}}\ L_{0\hat{k}}\mathbb{P}\left(Y=0|X=x\right) + L_{1\hat{k}}\mathbb{P}\left(Y=1|X=x\right) \end{aligned} \tag{2}$$

We predict 1 $\iff$

$$\begin{aligned} & L_{10}\mathbb{P}\left(Y=1|X=x\right) \geq L_{01}\mathbb{P}\left(Y=0|X=x\right) \\ \iff\quad & L_{10}\mathbb{P}\left(Y=1|X=x\right) \geq L_{01}\left(1-\mathbb{P}\left(Y=1|X=x\right)\right) \\ \iff\quad & \mathbb{P}\left(Y=1|X=x\right) \geq \frac{L_{01}}{L_{10}+L_{01}} \end{aligned} \tag{3}$$

Therefore, our prediction rule is

$$\hat{f}(x) = \begin{cases} 1 & \text{if } \mathbb{P}\left(Y=1|X=x\right) \geq \frac{L_{01}}{L_{10}+L_{01}} \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

(ii) If $L_{10} > L_{01}$, $\frac{L_{01}}{L_{10}+L_{01}} < 0.5$. Consequently we make positive predictions more aggressively than we did in the 0-1 loss case. Namely, we predict "1" when $\mathbb{P}\left(Y=1|X=x\right)$ is greater than the threshold $\frac{L_{01}}{L_{10}+L_{01}}$, *which is less than a half.* By making more aggressive positive predition, we take the fact that false negatives are worse than false positives in to account, so it makes sense.

(iii) Like before, define $p(\boldsymbol{x}) := \mathbb{P}(Y = 1|X = \boldsymbol{x})$. In logistic regression, we model

$$\log \frac{p(\boldsymbol{x})}{1 - p(\boldsymbol{x})} = \boldsymbol{x}^\top \boldsymbol{\beta} \quad \Rightarrow \quad p(\boldsymbol{x}) = \frac{e^{\boldsymbol{x}^\top \boldsymbol{\beta}}}{1 + e^{\boldsymbol{x}^\top \boldsymbol{\beta}}} \tag{5}$$

Hence the decision boundary becomes

$$\frac{e^{\boldsymbol{x}^\top \boldsymbol{\beta}}}{1 + e^{\boldsymbol{x}^\top \boldsymbol{\beta}}} \geq \frac{L_{01}}{L_{10} + L_{01}} \quad \Rightarrow \quad \boldsymbol{x}^\top \boldsymbol{\beta} \geq \log \frac{L_{01}}{L_{10}} \tag{6}$$

Given $L_{10} > L_{01}$, the new decision boundary is $\log \frac{L_{01}}{L_{10}} < 0$. If we consider the one-dimensional case, this implies that the decision boundary is pulled to the left, closer to the negative samples. This also indicates that we are making more aggressive positive predictions than before.

---

2 Marketing Data

```
In [1]: import numpy as np
        import pandas as pd
        import sklearn as sk
        import matplotlib
        import matplotlib.pyplot as plt
        plt.style.use('ggplot')
        %matplotlib inline

        import scipy.stats
        from sklearn.model_selection import train_test_split
        from sklearn.metrics import accuracy_score, \
            roc_curve, confusion_matrix
        from sklearn.linear_model import LogisticRegression
        from pygam import LogisticGAM
        from pygam.utils import generate_X_grid
        from copy import copy
        from progressbar import ProgressBar
```

```
In [2]: df = pd.read_csv('marketing.csv')
        X = pd.get_dummies(df.iloc[:,:-1]).values
        y = np.where(df['y'] == 'yes', 1, 0)
        X_train, X_test, y_train, y_test = train_test_split(
            X, y, test_size=0.33, random_state=1)
        print(X_train.shape, X_test.shape)
        print(y_train.shape, y_test.shape)
```

```
(30291, 27) (14920, 27)
(30291,) (14920,)
```

2.1 Fraction of Success in Training Set

```
In [3]: sum(y_train)/len(y_train)
```

```
Out[3]: 0.11782377603908752
```

2.2 Logistic Regression

```
In [4]: logit_clf = LogisticRegression(C=10e5)
        logit_clf.fit(X_train, y_train)

        y_pred = logit_clf.predict(X_test)
        y_silly = np.zeros(len(y_test))

        misclf_logistic = 1-accuracy_score(
            y_test, y_pred, normalize=1)
        misclf_silly = 1-accuracy_score(
            y_test, y_silly, normalize=1)
        print('Logistic Regression Misclassification Rate: ',
              misclf_logistic)
        print('Guessing 0 Misclassification Rate: ',
              misclf_silly)
```

```
Logistic Regression Misclassification Rate:  0.115482573727
Guessing 0 Misclassification Rate:  0.11528150134
```

Indeed, silly guess achieves lower misclassification rate on this imbalanced dataset.

```
In [5]: # Confusion Matrix
        confusion_matrix(y_test, y_pred)
```

```
Out[5]: array([[13197,     3],
               [ 1720,     0]])
```

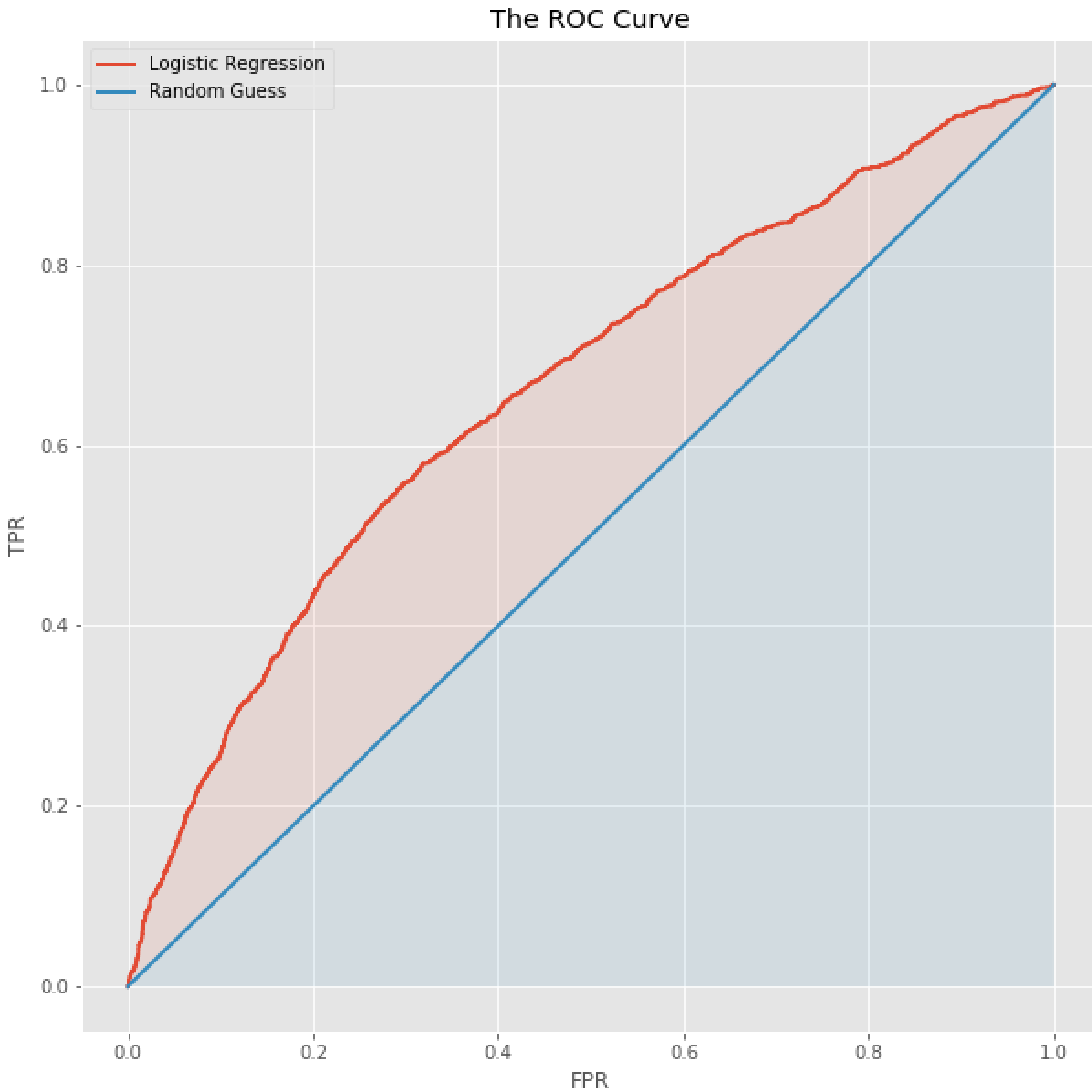
2.3 Top 1000 Clients from Probability Predition

```
In [6]: prob_pred_logit = logit_clf.predict_proba(X_test)[:,1]
        top1k_indices = np.argsort(-prob_pred_logit)[:1000]

        success_rate_clf = sum(
            y_test[list(top1k_indices)])/1000
        success_rate_random_pick = sum(
            y_test)/len(y_test)

        print('Success rate of calling top 1000 clients: ',
              success_rate_clf)
        print('Success rate of random call: ',
              success_rate_random_pick)
```

```
Success rate of calling top 1000 clients:  0.279
Success rate of random call:  0.11528150134
```

2.4 ROC Curve

```
In [24]: fpr, tpr, _ = roc_curve(y_test, prob_pred_logit)
         fig, ax = plt.subplots(1,1, figsize=(10,10))
         ax.plot(fpr, tpr, linewidth=2,
                 label='Logistic Regression')
         ax.plot([0,1], [0,1], linewidth=2,
                 label='Random Guess')
         ax.fill_between(fpr, tpr, fpr, alpha=0.1)
         ax.fill_between(tpr, [0]*len(tpr), tpr, alpha=0.1)
         ax.update({'xlabel':'FPR', 'ylabel':'TPR',
                    'title':'The ROC Curve'})
         _ = ax.legend()
```



3 Logistic GAM on Marketing Data

3.1 Fitting Logistic GAM

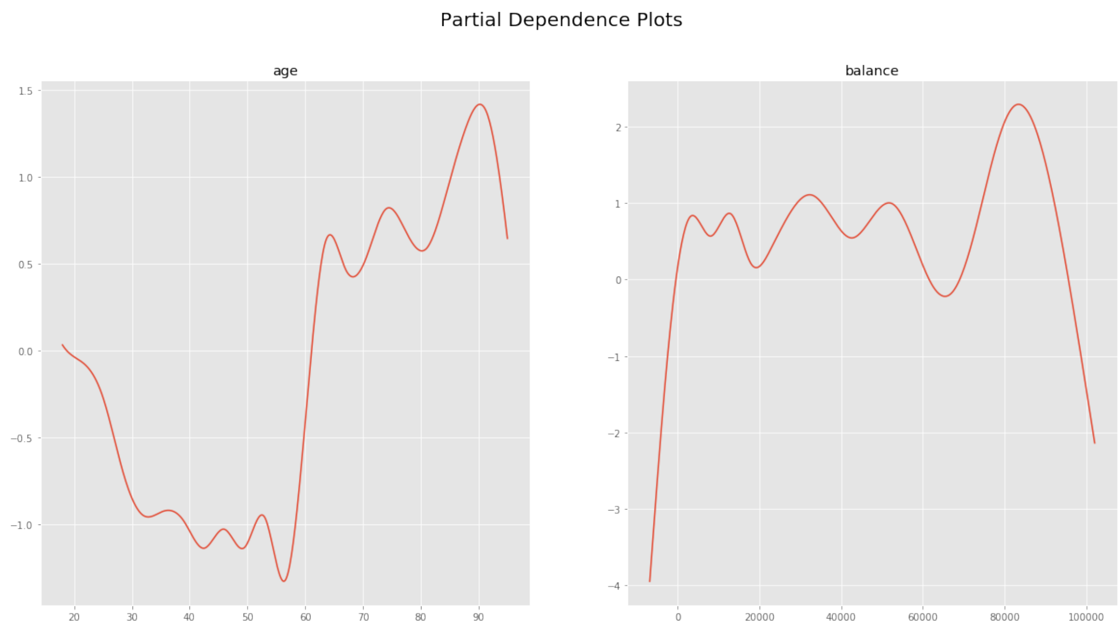
In [8]:
```
%%capture --no-stdout --no-display

gam_clf = LogisticGAM().gridsearch(
    X_train, y_train)

y_pred_gam = gam_clf.predict(X_test)
misclf_logitgam = 1-accuracy_score(
    y_test, y_pred_gam, normalize=1)
print('Logistic GAM Misclassification Rate: ',
```

```
              misclf_logitgam)

100% (11 of 11) |#########################| Elapsed Time: 0:00:54 Time: 0:00:54


Logistic GAM Misclassification Rate:  0.117292225201
```

```
In [9]: # Lambda used in logistic gam
        gam_clf.lam
```

```
Out[9]: 0.25118864315095796
```

```
In [10]: X_grid = generate_X_grid(gam_clf)
         X_names = ['age', 'balance']
         fig, axes = plt.subplots(1, 2, figsize=(20, 10))

         for i, ax in enumerate(axes):
             pdep, confi = gam_clf.partial_dependence(
                 X_grid, feature=i+1, width=.95)
             ax.plot(X_grid[:, i], pdep)
             ax.set_title(X_names[i])
         _ = fig.suptitle('Partial Dependence Plots', fontsize=20)
```



We indeed observe non-linear dependence upon `age` and `balance`. Namely, there is a jump at the age of 60; and for `balance`, there is a plateau in the middle, and plunges at both ends.

3.2 Model Evaluation

```
In [11]: print('Logistic Regression Misclassification Rate: ',
               misclf_logistic)
         print('Guessing 0 Misclassification Rate: ',
               misclf_silly)
         print('Logistic GAM Misclassification Rate: ',
               misclf_logitgam)
```

```
Logistic Regression Misclassification Rate:  0.115482573727
Guessing 0 Misclassification Rate:  0.11528150134
Logistic GAM Misclassification Rate:  0.117292225201
```

```
In [13]: prob_pred_gam = gam_clf.predict_proba(X_test)
         top1k_indices = np.argsort(-prob_pred_gam)[:1000]

         success_rate_gam = sum(
             y_test[list(top1k_indices)])/1000
         success_rate_random_pick = sum(
             y_test)/len(y_test)

         print('Success rate of calling top 1000 clients',
               '(Logistic GAM):',
               success_rate_gam)
         print('Success rate of calling top 1000 clients: ',
               '(Logistic Regression):',
               success_rate_clf)
         print('Success rate of random call:',
               success_rate_random_pick)
```

```
Success rate of calling top 1000 clients (Logistic GAM): 0.345
Success rate of calling top 1000 clients:  (Logistic Regression): 0.279
Success rate of random call: 0.11528150134
```

The Misclassification rate of Logistic GAM is no better than random guess or logistic regression (even a bit higher). However, when we look at the top 1000 clients from its probability predition, the ratio of success is 0.345, a significant improve from logistic regression.

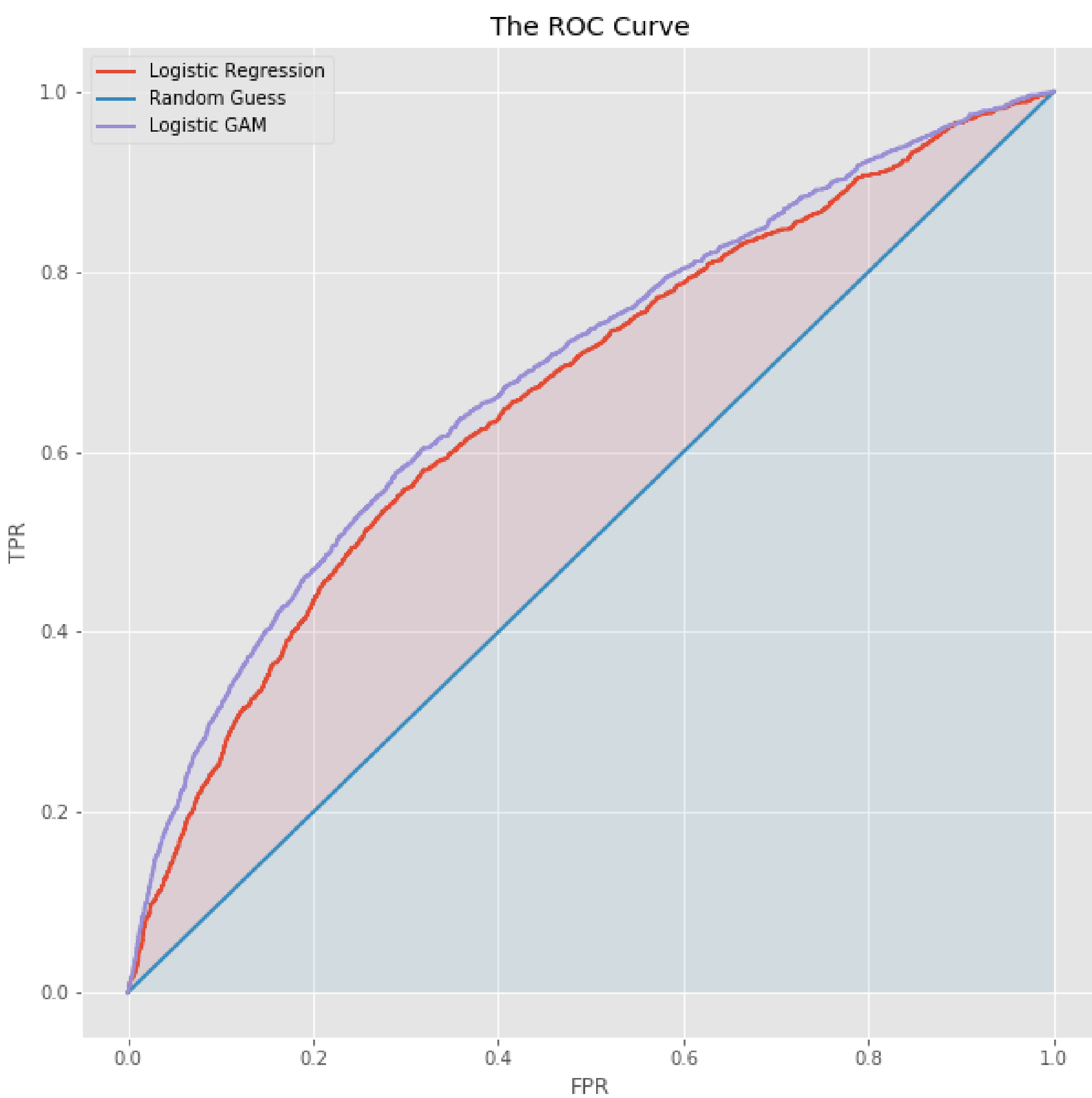
3.3 ROC Curve

```
In [32]: fpr1, fpr2

Out[32]: (array([  0.00000000e+00,   7.57575758e-05,   5.30303030e-04, ...,
                   9.99772727e-01,   9.99772727e-01,   1.00000000e+00]),
          array([  0.00000000e+00,   7.57575758e-05,   1.51515152e-04, ...,
                   9.99696970e-01,   9.99696970e-01,   1.00000000e+00]))
```

```
In [29]: len(fpr2),len(fpr1)
```

```
Out[29]: (3342, 3421)
```

```
In [36]: fpr1, tpr1, _ = roc_curve(y_test, prob_pred_logit)
         fpr2, tpr2, _ = roc_curve(y_test, prob_pred_gam)
         fig, ax = plt.subplots(1,1, figsize=(10,10))
         ax.plot(fpr1, tpr1, linewidth=2,
                 label='Logistic Regression')
         ax.plot([0,1], [0,1], linewidth=2,
                 label='Random Guess')
         ax.plot(fpr2, tpr2, linewidth=2,
                 label='Logistic GAM')
         ax.fill_between(fpr1, tpr1, fpr1, alpha=0.1)
         ax.fill_between(tpr1, [0]*len(tpr1), tpr1, alpha=0.1)
         ax.fill_between(fpr2, tpr2, fpr2, alpha=0.1)
         ax.update({'xlabel':'FPR', 'ylabel':'TPR',
                    'title':'The ROC Curve'})
         _ = ax.legend()
```



The ROC curve of logistic GAM swipes a larger area than that of logistic regression and random guess. We find it a better classifier.