COMPSIC 617 Spring 2019

Homework 3

Yiteng Lu

1. Probit Classification

(a)

Given $Y = \begin{cases} 1 & \text{if } Z > 0 \\ 0 & \text{otherwise} \end{cases}$ and $Z = \mathbf{x}\beta + \epsilon$, then
$\Pr(Y = 1|\mathbf{x}) = \Pr(Z > 0) = \Pr(Z = \mathbf{x}\beta + \epsilon > 0)$

$\Rightarrow \Pr(\mathbf{x}\beta + \epsilon > 0) = \Pr(\epsilon > -\mathbf{x}\beta)$

Because ϵ belongs to a standard nomral $\mathcal{N}(0, 1)$,
$\Pr(\epsilon > -\mathbf{x}\beta) = 1 - \Pr(\epsilon < -\mathbf{x}\beta) = 1 - \Phi(-\mathbf{x}\beta) = \Phi(\mathbf{x}\beta)$

Therefore: $\Pr(Y = 1|\mathbf{x}) = \Phi(\mathbf{x}\beta)$

(b)

$Y \sim Bernoulli(p), p = \Phi(\mathbf{x}\beta)$:

$f(y|\mathbf{x}, \beta) = \Phi(\mathbf{x}\beta)^y (1 - \Phi(\mathbf{x}\beta))^{1-y}$, for $y = 0, 1$

$L(\beta) = \Pr(Y = y_1 \ldots Y = y_n | \beta, x_1 \ldots x_n) = \prod_{i=1}^{N} \Phi(\mathbf{x_i}\beta)^{y_i} (1 - \Phi(\mathbf{x_i}\beta))^{1-y_i}$

$$
\begin{aligned}
logL(\beta) &= log \prod_{i=1}^{N} \Phi(\mathbf{x_i}\beta)^{y_i} (1 - \Phi(\mathbf{x_i}\beta))^{1-y_i} \\
&= \sum_{i=1}^{N} log\Phi(\mathbf{x_i}\beta)^{y_i} (1 - \Phi(\mathbf{x_i}\beta))^{1-y_i} \\
&= \sum_{i=1}^{N} log\Phi(\mathbf{x_i}\beta)^{y_i} + log(1 - \Phi(\mathbf{x_i}\beta))^{1-y_i} \\
&= \sum_{i=1}^{N} log\Phi(\mathbf{x_i}\beta)^{y_i} + log(\Phi(-\mathbf{x_i}\beta))^{1-y_i} \\
&= \sum_{i=1}^{N} y_i \; log\Phi(\mathbf{x_i}\beta) + (1 - y_i) \; log(\Phi(-\mathbf{x_i}\beta))
\end{aligned}
$$

(c)

$$\frac{\partial - \log\, \Phi(a)}{\partial a} = -\frac{\phi(a)}{\Phi a}$$

$$\frac{\partial^2 - \log(\Phi(a))}{\partial a^2} = \frac{\partial - \frac{\phi(a)}{\Phi a}}{\partial a} = -\frac{-a\phi(a)\Phi(a) - (\phi(a))^2}{(\Phi(a))^2} = \frac{a\phi(a)\Phi(a) + (\phi(a))^2}{(\Phi(a))^2}$$

Given $a\Phi(a) + \phi(a) > 0 \text{ and } \phi(a) > 0\ \forall a$:

$$a\phi(a)\Phi(a) + (\phi(a))^2 = \phi(a)[a\Phi(a) + \phi(a)] \geq 0$$

The denominator is $(\Phi(x))^2$, so $\frac{\partial^2 - \log(\Phi(a))}{\partial a^2} = \frac{a\phi(a)\Phi(a) + (\phi(a))^2}{(\Phi(a))^2} \geq 0$ which indicates that $\log\, \Phi(a)$ is convex for all a.

(d)

$$\frac{\partial - \log\, \mathcal{L}(\beta)}{\partial \beta} = -\sum_{i=1}^{N} y_i \frac{x_i \phi(\mathbf{x}_i \beta)}{\Phi(\mathbf{x}_i \beta)} + (1 - y_i)\frac{-x_i \phi(\mathbf{x}_i \beta)}{1 - \Phi(\mathbf{x}_i \beta)}$$

2. Adaboost with L_2 Loss

(a)

The LHS:

$$\begin{aligned}
\underset{j=1,\ldots,m}{\arg\max}\left[-\frac{\partial}{\partial \alpha} L(\lambda + \alpha \mathbf{e}_j)\right]\Bigg|_{\alpha=0} &= \underset{j=1,\ldots,m}{\arg\max}\left[-\frac{\partial}{\partial \alpha}\frac{1}{n}\sum_{i=1}^{n}[y_i - h(\mathbf{x}_i)(\lambda + \alpha \mathbf{e}_j)]^2\right]\Bigg|_{\alpha=0} \\
&= \underset{j=1,\ldots,m}{\arg\max}\left[-\frac{2}{n}\sum_{i=1}^{n}[y_i - h(\mathbf{x}_i)(\lambda + \alpha \mathbf{e}_j)][-h(\mathbf{x}_i)\mathbf{e}_j]\right]\Bigg|_{\alpha=0} \\
&= \underset{j=1,\ldots,m}{\arg\max}\left[\sum_{i=1}^{n}[y_i - h(\mathbf{x}_i)(\lambda + \alpha \mathbf{e}_j)]h_j(\mathbf{x}_i)\right]\Bigg|_{\alpha=0}
\end{aligned}$$

$$= \arg\max_{j=1,\dots,m} \left[\sum_{i=1}^{n} [y_i - h(\mathbf{x}_i)\lambda] h_j(\mathbf{x}_i) \right]$$

$$= \arg\max_{j=1,\dots,m} \left[\sum_{i=1}^{n} y_i h_j(\mathbf{x}_i) - h_j(\mathbf{x}_i) h(\mathbf{x}_i)\lambda \right]$$

$$= \arg\max_{j=1,\dots,m} \left[\sum_{i=1}^{n} y_i h_j(\mathbf{x}_i) - \sum_{i=1}^{n} h_j(\mathbf{x}_i) h(\mathbf{x}_i)\lambda \right]$$

The RHS:

$$\arg\max_{j=1,\dots,m} h_j^T(\mathbf{X})(\mathbf{y} - H(\mathbf{X})\lambda)$$

$$= \arg\max_{j=1,\dots,m} h_j^T(\mathbf{X})\mathbf{y} - h_j^T(\mathbf{X}) H(\mathbf{X})\lambda$$

$$= \arg\max_{j=1,\dots,m} \left[h_j(\mathbf{x}_1) \cdots h_j(\mathbf{x}_n) \right] \cdot \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} - \left[h_j(\mathbf{x}_1) \cdots h_j(\mathbf{x}_n) \right] \cdot \begin{bmatrix} h_1(\mathbf{x}_1) & \dots & h_m(\mathbf{x}_1) \\ \vdots & \ddots & \vdots \\ h_1(\mathbf{x}_n) & \dots & h_m(\mathbf{x}_n) \end{bmatrix} \cdot \lambda$$

$$= \arg\max_{j=1,\dots,m} (h_j(\mathbf{x}_1) y_1 + h_j(\mathbf{x}_2) y_2 + \cdots h_j(\mathbf{x}_n) y_n) - \left[h_j(\mathbf{x}_1) h_1(\mathbf{x}_1) + \cdots + h_j(\mathbf{x}_n) h_1(\mathbf{x}_n) \ , \quad \cdot \right.$$

$$\cdot \lambda$$

$$= \arg\max_{j=1,\dots,m} \sum_{i=1}^{n} h_j(\mathbf{x}_i) y_i - \left[\sum_{i=1}^{n} h_j(\mathbf{x}_i) h_1(\mathbf{x_i}) \ , \quad \cdots \quad , \ \sum_{i=1}^{n} h_j(\mathbf{x}_i) h_m(\mathbf{x_i}) \right] \cdot \lambda$$

$$= \arg\max_{j=1,\dots,m} \sum_{i=1}^{n} h_j(\mathbf{x}_i) y_i - \sum_{i=1}^{n} h_j(\mathbf{x}_i) \left[h_1(\mathbf{x_i}) \ , \quad \cdots \quad , \ h_m(\mathbf{x_i}) \right] \cdot \lambda$$

$$= \arg\max_{j=1,\dots,m} \sum_{i=1}^{n} y_i h_j(\mathbf{x}_i) - \sum_{i=1}^{n} h_j(\mathbf{x}_i) h(\mathbf{x_i})\lambda$$

Therefore, $\arg\max_{j=1,\dots,m} \left[-\frac{\partial}{\partial \alpha} L(\lambda + \alpha \mathbf{e}_j) \right] \Big|_{\alpha=0} = \arg\max_{j=1,\dots,m} h_j^T(\mathbf{X})(\mathbf{y} - H(\mathbf{X})\lambda)$

(b)

Let $\frac{\partial L(\lambda+\alpha \mathbf{e}_j)}{\partial \alpha}\Bigg|_{\alpha^{(t)}}$ = 0, solve for $\alpha^{(t)}$

$$\begin{aligned}
\frac{\partial L(\lambda + \alpha \mathbf{e}_j^{(t)})}{\partial \alpha}\Bigg|_{\alpha^{(t)}} &= \frac{2}{n}\sum_{i=1}^{n}[y_i - h(\mathbf{x}_i)(\lambda + \alpha \mathbf{e}_j)](-h(\mathbf{x}_i)\mathbf{e}_j)\Bigg|_{\alpha^{(t)}} \\
&= \frac{2}{n}\sum_{i=1}^{n}[y_i - h(\mathbf{x}_i)(\lambda + \alpha^{(t)}\mathbf{e}_{j^{(t)}})](-h_{j^{(t)}}(\mathbf{x}_i)) \\
&= \frac{2}{n}\sum_{i=1}^{n}\left[-y_i h_{j^{(t)}}(\mathbf{x}_i) + h(\mathbf{x}_i)\lambda h_{j^{(t)}}(\mathbf{x}_i) + h(\mathbf{x}_i)\alpha^{(t)}\mathbf{e}_{j^{(t)}} h_{j^{(t)}}(\mathbf{x}_i)\right] \\
&= \frac{2}{n}(-\sum_{i=1}^{n} y_i h_{j^{(t)}}(\mathbf{x}_i) - h(\mathbf{x}_i)\lambda h_{j^{(t)}}(\mathbf{x}_i) + \sum_{i=1}^{n} h_{j^{(t)}}(\mathbf{x}_i)\alpha^{(t)} h_{j^{(t)}}(\mathbf{x}_i)) \\
&= \frac{2}{n}(-h_{j^{(t)}}^T(\mathbf{X})(\mathbf{y} - H(\mathbf{X})\lambda) + \sum_{i=1}^{n} h_{j^{(t)}}^2(\mathbf{x}_i)\alpha^{(t)}) \\
&= 0
\end{aligned}$$

$\sum_{i=1}^{n} h_{j^{(t)}}^2(\mathbf{x}_i)\alpha^{(t)} = h_{j^{(t)}}^T(\mathbf{X})(\mathbf{y} - H(\mathbf{X})\lambda)$

$\alpha^{(t)} = \frac{h_{j^{(t)}}^T(\mathbf{X})(\mathbf{y} - H(\mathbf{X})\lambda)}{\sum_{i=1}^{n} h_{j^{(t)}}^2(\mathbf{x}_i)}$

(c)

Given training data $D = (x_i, y_i)_{i=1}^n$ and maximum number of iterations T
Initialize weights: $\lambda_1 = \mathbf{0}$
For $t = 1, \ldots, T$ **do:**
$$j_t \in \underset{j=1,\ldots,m}{\arg\max}\, h_j^T(\mathbf{X})(\mathbf{y} - H(\mathbf{X})\lambda)$$

compute for the coefficient: $\alpha^{(t)} = \frac{h_{j(t)}^T(\mathbf{X})(\mathbf{y}-H(\mathbf{X})\lambda)}{\sum_{i=1}^n h_{j(t)}^2(\mathbf{x}_i)}$

update the weights for classifier: $\lambda_{t+1} = \lambda_t + \alpha \mathbf{e}_{jt}$

end for

output the $\lambda h(x_i) = \sum_{t=1}^T \frac{h_{j(t)}^T(\mathbf{X})(\mathbf{y}-H(\mathbf{X})\lambda)}{\sum_{i=1}^n h_{j(t)}^2(\mathbf{x}_i)} h_{j^{(t)}}(x_i)$

3.Comparing AdaBoost and Logistic Regression

(a)

In [59]:
```
import pandas as pd
import numpy as np
training_data = pd.read_csv('house_votes_84_train.csv').values
testing_data = pd.read_csv('house_votes_84_test.csv').values
```

In [69]:
```
class Adaboost:
    def __init__(self, training_data , testing_data):
        self.label = training_data.T[0]
        self.data = training_data[:,1:]
        self.M = np.zeros((self.data.shape[0], self.data.shape[1]))   #matri
x of margin for a weak classifier
        for i in range(self.data.shape[1]):
            self.M[:, i] = self.label * self.data[:,i]
        self.lambda_ = np.zeros(self.data.shape[1])
        self.test_data = testing_data[:,1:]
        self.test_label = testing_data.T[0]

    def train_adaboost(self):
```

```
        d= np.ones(self.data.shape[0])/self.data.shape[0]
        pre_error = 0;
        error = 1;
        error_index = []
        Flag = False
        count =1
        while (abs(error - pre_error )> 10**-5):
            if Flag:
                pre_error = error
            error = 0
            j = np.argmax(np.dot(d.T,self.M))
            for i in range(len(self.M.T[j])):
                if self.M.T[j][i] < 0:
                    error += d[i]
                    error_index.append(i)
            for k in range(len(d)):
                if k in error_index:
                    d[k] = 1/2*(d[k]/error)
                else:
                    d[k] = 1/2*(d[k]/(1-error))
            alpha = 1/2*np.log((1-error) / error)
            self.lambda_[j]+=alpha
            error_index.clear()
            Flag = True

    def predict(self,features):
        return np.sign(np.dot(self.lambda_, features))

    def accuracy(self):
        sum = 0;
        for x in range(self.test_data.shape[0]):
            if self.predict(self.test_data[x]) * self.test_label[x] > 0:
                sum += 1
        print(sum/self.test_data.shape[0])
        return sum/self.test_data.shape[0]
```

(b)

In [70]:
```
from scipy.optimize import minimize
from math import log
import sys
class LR:
    def __init__(self, training_data):
        self.label = training_data.T[0]
        self.data = training_data[:,1:]
        self.lambda_ = np.zeros(self.data.shape[1])
    def loss_fun(self, Lambda):
        sum = 0
        for i in range(self.data.shape[0]):
            sum += log(1+np.exp(-self.label[i] * np.dot(Lambda.T,self.data[i
])))
        return sum
    def maximization(self):
        self.lambda_ = minimize(self.loss_fun,self.lambda_, method='SLSQP',
tol =1e-6).x
    def logistic_function(self,features):
        probability = np.exp(np.dot(self.lambda_.T,features))/(1+np.exp(np.d
ot(self.lambda_.T,features)))
        return probability
    def predict(self,features):
        probability_1 = self.logistic_function(features)
        if probability_1 < 0.5:
            return -1
        else:
            return 1
```

In [79]:
```
##Notice: functions for calculating LR accuracy and making predictions for l
arge dataset  They are not used for LR
#impelementation!
def LR_accuracy(real_label, predict_label):
    misclassified_total = 0
    for i in real_label*np.array(predict_label):
        if i < 0:
            misclassified_total+=1
    print(1-misclassified_total/len(real_label))
    return 1-misclassified_total/len(real_label)

def predict_dataset(dataset, model):
    predicted_label = []
    for i in range(dataset[:,1:].shape[0]):
        label1 = model.predict(dataset[:,1:][i])
        predicted_label.append(label1)
    return predicted_label
```

(c)

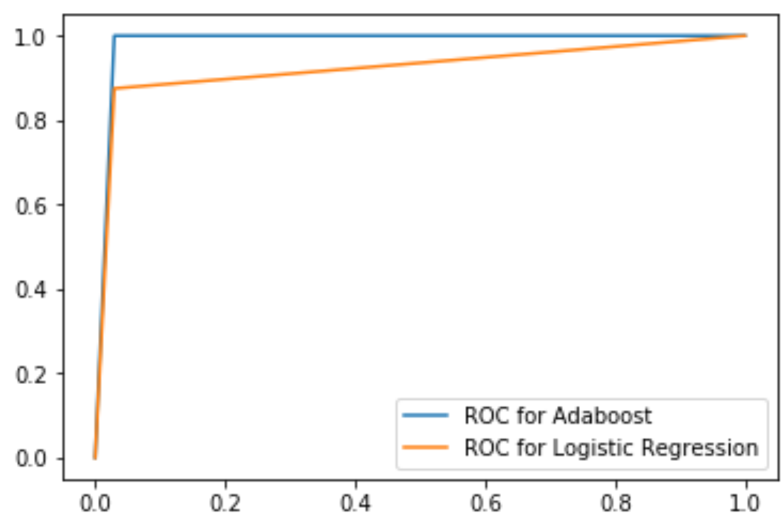
In [80]:
```
### adaboost train on training data set; test on both training and testing s
et
Adaboost_training_object = Adaboost(training_data, training_data)
Adaboost_training_object.train_adaboost()
print('The training accuracy for adaboost is ',end='')
_ = Adaboost_training_object.accuracy()
Adaboost_testing_object = Adaboost(training_data, testing_data)
Adaboost_testing_object.train_adaboost()
print('The testing accuracy for adaboost is ',end='')
_ = Adaboost_testing_object.accuracy()
ada_prediction_testset = predict_dataset(testing_data,Adaboost_testing_objec
t)
```

```
The training accuracy for adaboost is 0.9428571428571428
The testing accuracy for adaboost is 0.98
```

In [81]:
```
# same thing on the LR
Logistic_object = LR(training_data)
Logistic_object.maximization()
training_predict_label = predict_dataset(training_data, Logistic_object)
print("The training accuracy for LR is ", end='')
_=LR_accuracy(training_data.T[0], training_predict_label)
testing_predict_label = predict_dataset(testing_data,Logistic_object)
print("The testing accuracy for LR is ", end='')
_=LR_accuracy(testing_data.T[0], testing_predict_label)
```

```
The training accuracy for LR is 0.9636363636363636
The testing accuracy for LR is 0.94
```

In [78]:
```
## ROC curves
import sklearn.metrics
import matplotlib.pyplot as plt
fpr_ad,tpr_ad,_ = sklearn.metrics.roc_curve(testing_data.T[0], ada_predictio
n_testset)
fpr_lr,tpr_lr,_ = sklearn.metrics.roc_curve(testing_data.T[0] , testing_pred
ict_label)
plt.plot(fpr_ad,tpr_ad,label = "ROC for Adaboost")
plt.plot(fpr_lr,tpr_lr,label = "ROC for Logistic Regression")
plt.legend()
plt.show()
```



The logistic regression has higher training accuracy but we care more about the test accuracy which the adaboost is 4% accuracy higher than logistic regression. Therefore, given the test set, adaboost outperforms the logsitic regression.

In [65]:

```
### cross validation for adaboost
from sklearn.model_selection import train_test_split, KFold
from statistics import stdev, mean
cv= KFold(n_splits=10)
fold_number =1
accuracy_list =[]
print('AdaBoost for cross validation:')
for train, test in cv.split(training_data):
    Adaboost_object = Adaboost(training_data[train],training_data[test])
    Adaboost_object.train_adaboost()
    print('Accuracy for fold' + str(fold_number) + ':    ', end='')
    accuracy = Adaboost_object.accuracy()
    print('')
    accuracy_list.append(accuracy)
    fold_number +=1
mean_Ada = mean(accuracy_list)
std_Ada =stdev(accuracy_list)
print('The average accuracy for adaboost is', mean_Ada)
print('The stardard deviation for adaboost is', std_Ada)
```

```
AdaBoost for cross validation:
Accuracy for fold1:   0.9487179487179487

Accuracy for fold2:   0.9487179487179487

Accuracy for fold3:   0.9743589743589743

Accuracy for fold4:   0.9487179487179487

Accuracy for fold5:   0.9743589743589743

Accuracy for fold6:   0.9473684210526315

Accuracy for fold7:   0.9473684210526315

Accuracy for fold8:   0.9736842105263158

Accuracy for fold9:   0.7894736842105263

Accuracy for fold10:   0.9736842105263158

The average accuracy for adaboost is 0.9426450742240216
The stardard deviation for adaboost is 0.05533246287605921
```

In [66]:
```
### cross validation  for LR
fold_number =1
accuracy_list2=[]
print('Logisitic Regression for cross validation:')
for train, test in cv.split(training_data):
    LR_classifier = LR(training_data[train])
    LR_classifier.maximization()
    LR_predicted_labels = predict_dataset(training_data[test], LR_classifier
)
    print('Accuracy for fold'+ str(fold_number) + ':  ', end='')
    accuracy = LR_accuracy(training_data[test].T[0], LR_predicted_labels)
    print('')
    accuracy_list2.append(accuracy)
    fold_number +=1

mean_LR = mean(accuracy_list2)
std_LR =stdev(accuracy_list2)
print('The average accuracy for logistic regression is', mean_LR)
print('The stardard deviation for logistic regression is', std_LR)
```

```
Logisitic Regression for cross validation:
Accuracy for fold1:  0.9743589743589743

Accuracy for fold2:  0.9743589743589743

Accuracy for fold3:  0.9487179487179487

Accuracy for fold4:  0.9487179487179487

Accuracy for fold5:  0.9487179487179487

Accuracy for fold6:  0.9473684210526316

Accuracy for fold7:  0.9736842105263158

Accuracy for fold8:  0.8947368421052632

Accuracy for fold9:  0.8421052631578947

Accuracy for fold10:  0.9473684210526316

The average accuracy for logistic regression is 0.9400134952766531
The stardard deviation for logistic regression is 0.041452021216918246
```

Use the cross validation, we can see the average accuracy for both alogrithms are very close, adaboost slightly outperforms logisitic regression on the accuracy, something around 0.2% but LR has stardard deviation which is 0.5% lower than adaboost does. Therefore, the performances for both algorithm are almost the same for this real-world data.

(d)

In [47]:
```
# variable importance. randomly permute a feature on the test set and look i
nto the accuracy change
# adaboost
testing_dataframe =  pd.read_csv('house_votes_84_test.csv')
original_dataframe = testing_dataframe.copy(deep=True)
column_names =list(testing_dataframe)
print('The Testing accuracy with permuted features for Adaboost')
for name in column_names[1:]:
    testing_dataframe[name] = np.random.permutation(testing_dataframe[name])
    test_new_data = testing_dataframe.values
    Adaboost_permuted_object = Adaboost(training_data, test_new_data)
    print(name + ': ',end='')
    Adaboost_permuted_object.train_adaboost()
    _ = Adaboost_permuted_object.accuracy()
    testing_dataframe=original_dataframe.copy(deep=True)
```

```
The Testing accuracy with permuted features for Adaboost
bill_1: 0.98
bill_2: 0.98
bill_3: 0.98
bill_4: 0.62
bill_5: 0.96
bill_6: 0.98
bill_7: 0.98
bill_8: 0.98
bill_9: 0.98
bill_10: 0.98
bill_11: 0.98
bill_12: 0.96
bill_13: 0.98
bill_14: 0.98
bill_15: 0.98
bill_16: 0.98
```

In [48]:
```
# same thing for the LR
testing_dataframe =  pd.read_csv('house_votes_84_test.csv')
original_dataframe = testing_dataframe.copy(deep=True)
column_names =list(testing_dataframe)
print("The testing accuracy with permuted features for Logistic Regression")
for name in column_names[1:]:
    testing_dataframe[name] = np.random.permutation(testing_dataframe[name])
    test_new_data = testing_dataframe.values
    Logistic_model = LR(training_data)
    Logistic_model.maximization()
    testing_predict_label = predict_dataset(test_new_data,Logistic_model)
    print(name + ': ', end='')
    LR_accuracy(test_new_data.T[0], testing_predict_label)
    testing_dataframe=original_dataframe.copy(deep=True)
```

```
The testing accuracy with permuted features for Logistic Regression
bill_1: 0.96
bill_2: 0.94
bill_3: 0.9
bill_4: 0.62
bill_5: 0.94
bill_6: 0.98
bill_7: 0.96
bill_8: 0.92
bill_9: 0.94
bill_10: 0.96
bill_11: 0.94
bill_12: 0.96
bill_13: 0.96
bill_14: 0.96
bill_15: 0.94
bill_16: 0.94
```

I use the variable importance method to randomly permute the values for each features on the test data set, more specifically each bill, and find out the accuracy of both the adaboost and the logistic regression testing on the permuted-features data set. Both learning algorithms show that the fourth bill is extremely crucial that it is almost dominiating the importances of all the other bills because no matter which feature except bill_4 is permuted, the overall accuracy is around 95% (or higher) for both algorithms. However, once we shuffled the values for bill_4, adaboost acted worse than a weak classifier (below 0.5), and the performance of Logisitic Regression falls dramatically.