# Homework 3: October 5, 2016

Machine Learning

Susan Cherry

## Question 1

Assume that the convex halls of the two sets of data points $\{\mathbf{x_n}\}$ and $\{\mathbf{y_m}\}$ intersect. Denote the convex hall of $\{\mathbf{x}_n\}$ as $\mathbf{x} = \sum_n a_n \mathbf{x_n}$ and the convex hall of $\{\mathbf{y_m}\}$ as $\mathbf{y} = \sum_m b_m \mathbf{y}_m$, where $a_n, b_n \geq 0$ and $\sum_n a_n = \sum_m b_m = 1$.

Assume, to the contrary, that the two sets of points are linearly separable. Thus, there exists a $\mathbf{w}$ and $w_0$ such that $\mathbf{w}^T\mathbf{x}_n + w_0 > 0$ and $\mathbf{w}^T\mathbf{y}_m + w_0 < 0$. Then, we can have: $\sum_n a_n \mathbf{w}^T\mathbf{x}_n + w_0 > 0$ and $\sum_m b_m \mathbf{w}^T\mathbf{y}_m + w_0 < 0$, which leads us to $\mathbf{w}^T\mathbf{x} + w_0 > 0$ and $\mathbf{w}^T\mathbf{y} + w_0 < 0$. Let $\mathbf{z}$ be the intersection. $\mathbf{z}$ can be expressed as both $\sum_n a_n \mathbf{x_n}$ and $\sum_m b_m \mathbf{y}_m$, leading to $\mathbf{w}^T\mathbf{z} + w_0 > 0$ and $\mathbf{w}^T\mathbf{z} + w_0 < 0$ simultaneously. This is a contradiction, so the points are not linearly separable.

## Question 2

Now, the first 10 observations are worth twice the other observations. More generally, we could say that we are minimizing the weighted objective function $\sum_{i=1}^{10} g_i e^{-(M\lambda)_i} + \sum_{i>10} e^{-(M\lambda)_i}$, where $g$ is the weight assigned to the first 10 observations. Even more generally, we can first just consider minimizing the following training error: $\sum_i g_i e^{-(M\lambda)_i}$.

Using coordinate descent, we could minimize this loss function by doing the following: $j_t \in argmax_j[-\frac{\partial}{\partial \alpha}[\frac{1}{n}](\sum_{i=1} g_i e^{-(M\lambda)_i}] = j_t \in argmax_j[\frac{1}{n}(\sum_{i=1} g_i M_{ij} e^{-(M\lambda_t)_i}$. From here, we get the discrete probability distribution $d_{t,i} = \frac{(g_i e^{-M\lambda_t})_i}{Z_t}$ for $i = 1...n$, where $Z_t = \sum_{i=1}^n d_{i,t}$. We know that Adaboost minimizes exponential loss by coordinate descent.

Suppose that in Adaboost we have the weight $d_{t,i} = \frac{(g_i e^{-(M\lambda_t)_i})}{Z_t}$. Similarly, $d_{t+1,i} = \frac{(g_i e^{-(M\lambda_{t+1})_i})}{Z_{t+1}}$. Notice that $\lambda_{t+1} = \lambda_t + \alpha_t e_{j_t}$. Now we have $d_{t+1,i} = \frac{(ge^{-(M(\lambda_t + \alpha_t e_{j_t})_i})}{Z_{t+1}} \propto d_{t,i} e^{M_{i,j_t}\alpha_t}$.

This is exactly the same weight update in the original Adaboost formula. Therefore since the weights only depend on the weighting vector during the previous time period we need only update $d_{1,i}$, or the initial weighting vector. In the normal Adaboost formula, we initialized $d_{1,i} = 1/n$ for all $i = 1...n$. Now in this problem, we initialize it to be $d_{1,i} \propto 2$ for $i = 1...10$and $d_{1,i} \propto 1$ for all $i > 10$. Each $d_{1,i}$ will be divided by a normalizing constant $Z_1 = \sum_i d_{1,i}$ so that the probability distribution adds up to 1. The rest of the Adaboost algorithm remains unchanged. So we have:

```
Assign $d_{1,i} = 2/Z_1$ for $i = 1...10$ and $d_{1,i} = 1/Z_1$ for all $i > 10$ where $Z_1 = \sum_i d_{1,i}$.
for t=1....T
    Train weak learning algorithm using data weighted by $d_{t,i}$. This produces weak classifier $h_t$.
    Choose $\alpha_t = \frac{1}{2}ln(\frac{1-\epsilon_t}{\epsilon_t})$, where $\epsilon = \sum_{i:M_{i,j_t}=-1} d_{t,i}$
    Update weights $d_{t+1,i} = \frac{d_{t,i}e^{M_{i,j_t}\alpha_t}}{Z_{t+1}}$, where $Z_{t+1} = \sum_i d_{t+1,i}$.
End
Output final classifier $H(x) = sign(\sum_t \alpha_t h_t(x))$ .
```

## Question 3

**Part A**: Yes, we know that there is a linear classifier that classifies the dataset perfectly and we know that Adaboost will find it. First, note that we are told that the weak learning assumption holds. This means that by the theorem in our notes, the training error decays exponentially fast. Each weak classifier performs better than random guessing. And from the theorem, we know that $\frac{1}{n}\sum_i 1_{[y \neq H(x_i)]} \leq e^{-2\gamma_{WLA}^2}$, where $\epsilon_t = \frac{1}{2} = \gamma_t$ and $\gamma_t > \gamma_{WLA}$ by the weak learning assumption.

We use the weak learning assumption as follows: We know $\gamma_t > \gamma_{WLA}$ for all $t$. Also, misclassification error is upper bounded by the exponential loss. This leads us to $\frac{1}{n}\sum_i 1_{[y \neq H(x_i)]} = \frac{1}{m}\sum_i e^{-(M\lambda_T)_i} \leq e^{-2\sum_t \gamma_t^2} \leq e^{-2\gamma_{WLA}^2 \mathbf{T}}$.
As $\mathbf{T}$ goes to infinity, $e^{-2\gamma_{WLA}^2 \mathbf{T}}$ goes to 0. Because the misclassification error is bounded by this term, the misclassification error also goes to 0, meaning that our data set is perfectly classified by Adaboost.

**Part B**: Now, the Weak Learning Assumption does not hold. Consider the following data points: $x_1 = 1, x_2 = -1$ with the following labels: $y_1 = 1, y_2 = 1$. Also consider the weak classifier $h_1(x_i)$, which classifies a point as 1 if $x_i < 1$. The classification error on this training set is then 0.5 because $h_1$ classifies $x_1$ correctly but $x_2$ incorrectly. According to this problem, we know have $h_2(x_i) = -h_1(x_i)$. Again the misclassification error is 0.5. In this case, $h_2$ classifies $x_1$ incorrectly but $x_2$ correctly. Now we have the following Matrix of Margins: $M = [\begin{smallmatrix} 1 & -1 \\ -1 & 1 \end{smallmatrix}]$. Here I have the rows as data points and the columns as classifiers. Now, all that is left to show is that Adaboost does not necessarily need to choose a linear combination of weak classifiers that correctly classifie the points. We know from the Adaboost formula that $H(x) = sign(\sum \alpha_t h_t(x))$.

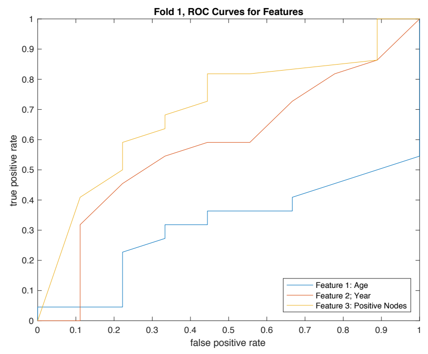
So we have $H(x_1) = \alpha_1 - \alpha_2$ and $H(x_2) = \alpha_2 - \alpha_1$. This is obviously a problem because this classifier will classify $x_1$ and $x_2$ with opposite labels, while their true labels are both 1. Adaboost therefore fails to perfectly classify the data set under these conditions.

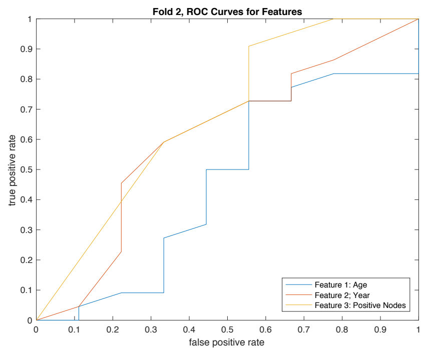
## Question 4

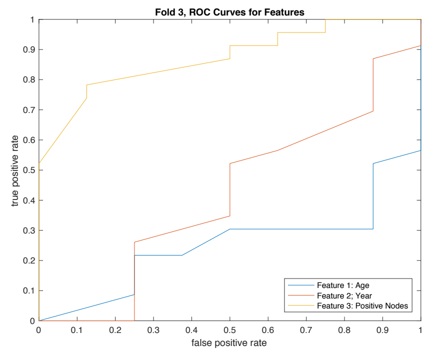
For this question, I use the Haberman's Survival Data Set, which contains information on whether patients who underwent surgery for cancer lived for more than 5 years after the procedure. The dataset contains three features: age at the time of the operation, year of operation, and number of positive auxiliary nodes detected. It has 306 observations. First, I recode the labels so that 1 correspond to a person living more than 5 years later and -1 corresponds to someone who died. There is no missing data in this dataset.

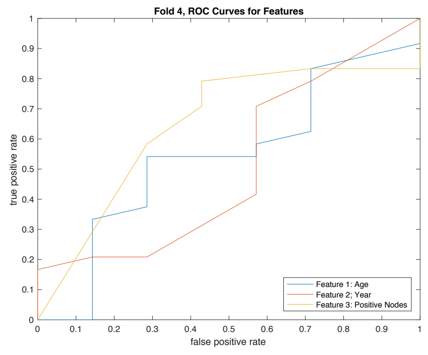
I will use the following algorithms: C.4 Decision Tree, Random Forest, Logistic Regression, Boosted Trees, and SVM. I will tune the Kernel Scale parameter in the SVM algorithm. This parameter takes either positive scalars or 'auto' as its input. Matlab divides all elements of X by the value of KernelScale and then applies the kernel norm to calculate the gram matrix. If 'auto' is the input, Matlab uses a heuristic procedure to determine the value of Kernel Scale. I try 5 values for this parameter: 0.5, 1.0, 1.5, 2.0, and 'auto'. For CART, a tune the MinLeafSize parameter, which means that each leaf must have at least that number of observations. 1 is the auto value that Matlab uses. Small MinLeafSize values lead to deep trees, while smaller lead to shallower trees. I try the following parameter values: 1, 2, 3, 4, 5.

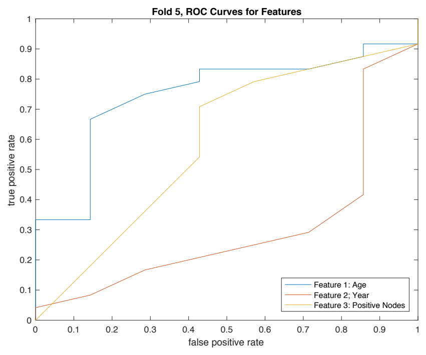
First, I split the data into 10 folds and do cross validation. I also do nested cross validation to tune the parameter values. Below are the ROC Curves for each feature for each of the 10 test folds. Most features appear to offer little advantage over random guessing, but the number of positive nodes is fairly good.

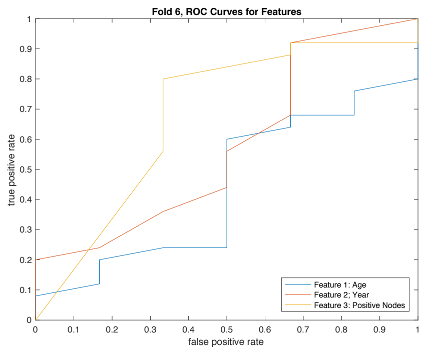


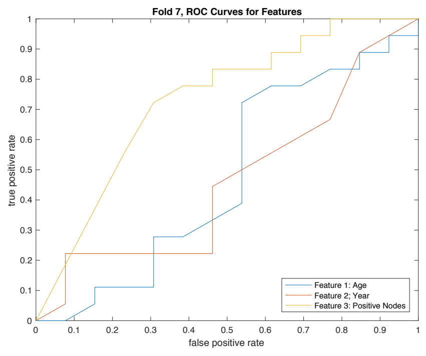


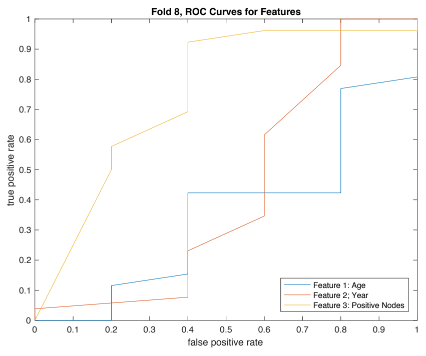

Fold 3, ROC Curves for Features
true positive rate
false positive rate
Feature 1: Age
Feature 2; Year
Feature 3: Positive Nodes

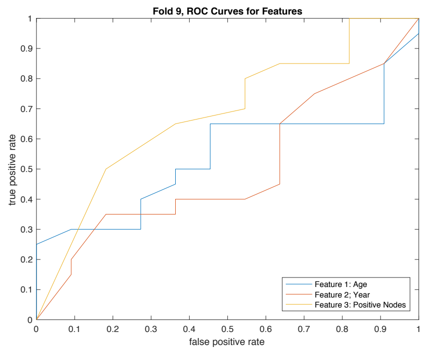

Fold 4, ROC Curves for Features
true positive rate
false positive rate
Feature 1: Age
Feature 2; Year
Feature 3: Positive Nodes

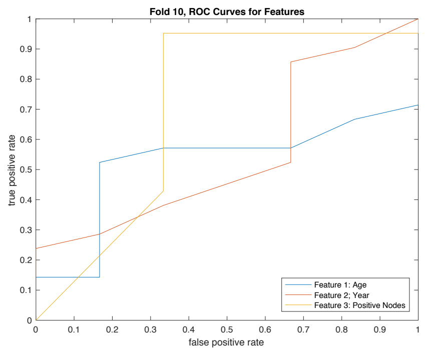

Fold 5, ROC Curves for Features
true positive rate
false positive rate
Feature 1: Age
Feature 2; Year
Feature 3: Positive Nodes


Fold 6, ROC Curves for Features
true positive rate
false positive rate
Feature 1: Age
Feature 2; Year
Feature 3: Positive Nodes


Fold 7, ROC Curves for Features
true positive rate
false positive rate
Feature 1: Age
Feature 2; Year
Feature 3: Positive Nodes

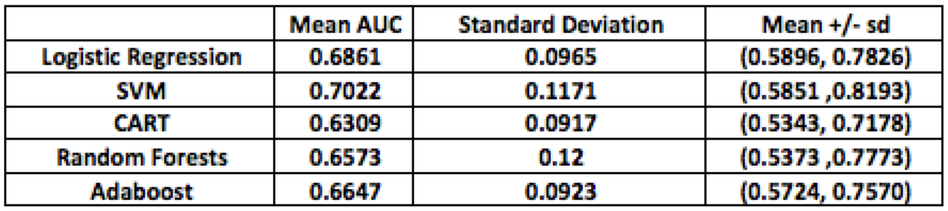

Fold 8, ROC Curves for Features
true positive rate
false positive rate
Feature 1: Age
Feature 2; Year
Feature 3: Positive Nodes


Fold 9, ROC Curves for Features
true positive rate
false positive rate
Feature 1: Age
Feature 2; Year
Feature 3: Positive Nodes


Fold 10, ROC Curves for Features
true positive rate
false positive rate
Feature 1: Age
Feature 2; Year
Feature 3: Positive Nodes

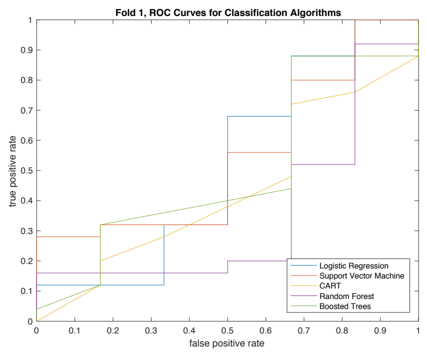
For nested cross validation, I find the parameter value that minimizes the average training error over the 9 validation folds. I use the classification error in this case, as specified by the lecture notes. I then use that parameter value to evaluate the test set. Then I repeat 10 times, using each fold in turn as the test fold. For SVM, Kernel Scale 1.5 and 2.0 each performed the best four out of the 10 times, while 1.0 performed the best two out of the ten times. 0.5 and 'auto' never performed the best. For decision trees, values 1 and 2 were each chosen three out of the 10 times. 4 was chosen twice and 3 and 5 where each chosen once.

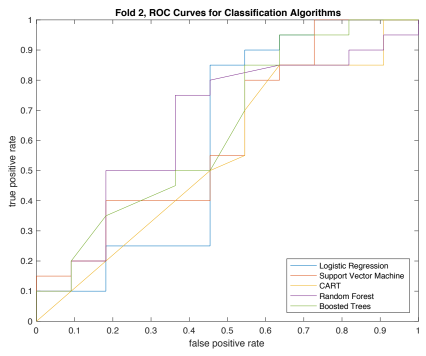
Next, I present the mean and standard deviation of the AUC values across the 10 test folds. See the chart below.

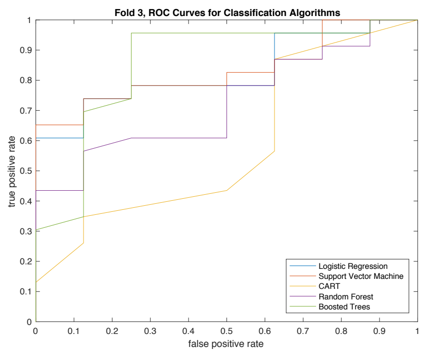


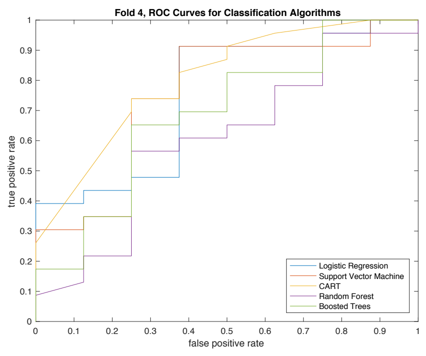
| | Mean AUC | Standard Deviation | Mean +/- sd |
|---|---|---|---|
| Logistic Regression | 0.6861 | 0.0965 | (0.5896, 0.7826) |
| SVM | 0.7022 | 0.1171 | (0.5851 ,0.8193) |
| CART | 0.6309 | 0.0917 | (0.5343, 0.7178) |
| Random Forests | 0.6573 | 0.12 | (0.5373 ,0.7773) |
| Adaboost | 0.6647 | 0.0923 | (0.5724, 0.7570) |

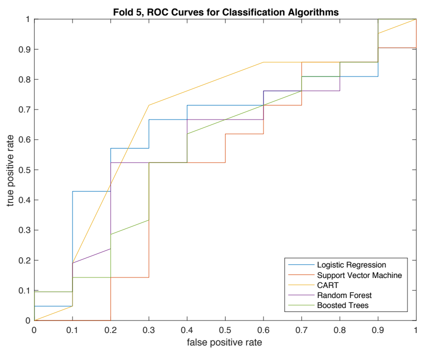
Below are the plotted ROC curves for each of the algorithms over the 10 test folds.

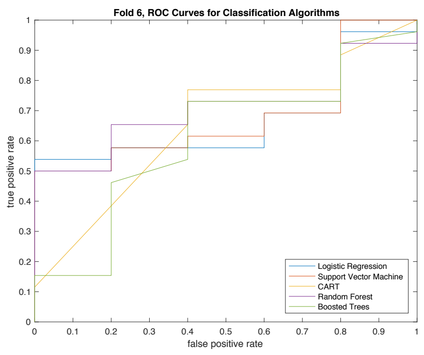


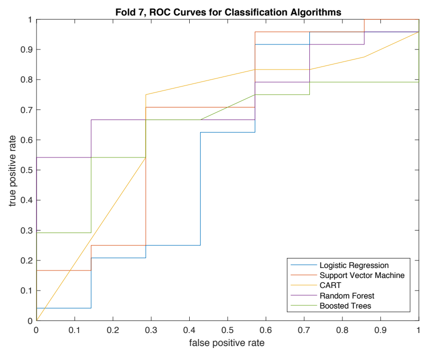


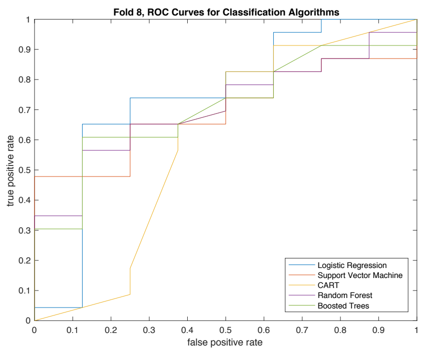


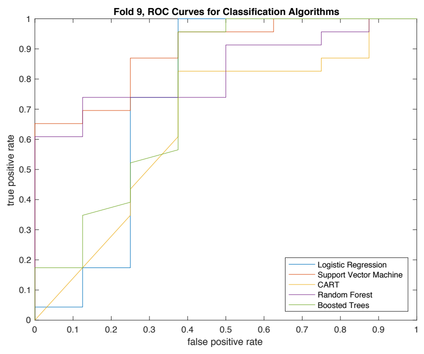


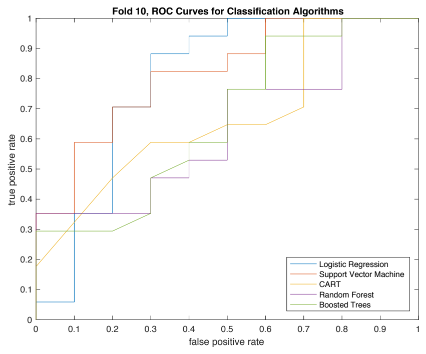












Discussion: I find that SVM has the best performance out of all of the algorithms, as measured by the AUC. It also has one of the largest standard deviations of AUC across the 10 test folds, however. CART has the worse average AUC and has the lowest standard deviation. Random Forest performs somewhat better than CART and Adaboost performs slightly better than Random Forest. Logistic Regression performs in the middle between SVM and Adaboost. These results aren't too surprising. I expect Random Forest to do better than CART, as Random Forest makes its decision by growing many decision trees and taking a majority vote, which adds variance and diversity to the model (as well as helps avoid the problem of overfitting that decision trees can have). Similarly, Adaboost is constructed as a linear combination of several weak classifiers, so I would also expect it to perform well. SVM performed very well, maybe because I was tuning one of its parameters, instead of just leaving the parameter at the default Matlab setting. The AUCs are fairly low for all of the algorithms, however (ranging only from 0.63 to 0.70). This is likely because my data set only has three features to predict death and only 300 observations. More observations and more features would likely improve the results. Overall, I find that combining features was better than the individual features alone in most cases. My ROC curves show that most individual features provide little advantage over random guessing (with perhaps the exception of number of positive nodes). The ROC curves for most of the features in most of the folds are around the random guessing line and there was a wide variance (for some folds, the features performed moderately well while for some they performed no better than random guessing). When I combine the features and use the algorithms, however, the ROC curves are pretty much always better than guessing and the AUC values are greater than 0.5.