Richard Albright
ISYE 6740
Fall 2020
Homework 6

1. **AdaBoost.** (30 points)

   Consider the following dataset, plotting in the following figure. The first two coordinates represent the value of two features, and the last coordinate is the binary label of the data.

   $$X_1 = (-1, 0, +1), X_2 = (-0.5, 0.5, +1), X_3 = (0, 1, -1), X_4 = (0.5, 1, -1),$$
   $$X_5 = (1, 0, +1), X_6 = (1, -1, +1), X_7 = (0, -1, -1), X_8 = (0, 0, -1).$$

   In this problem, you will run through $T = 3$ iterations of AdaBoost with decision stumps (as explained in the lecture) as weak learners.

   (a) (15 points) For each iteration $t = 1, 2, 3$, compute ϵ_t, α_t, Z_t, D_t by hand (i.e., show the calculation steps) and draw the decision stumps on the figure (you can draw this by hand).

   The details for the following summary are available in the attached jupyter notebook file `albright_richard_hw6q1.ipynb`. I first created a pandas dataframe of the values proived above.

	x1	x2	y
0	-1.0	0.0	1.0
1	-0.5	0.5	1.0
2	0.0	1.0	-1.0
3	0.5	1.0	-1.0
4	1.0	0.0	1.0
5	1.0	-1.0	1.0
6	0.0	-1.0	-1.0
7	0.0	0.0	-1.0

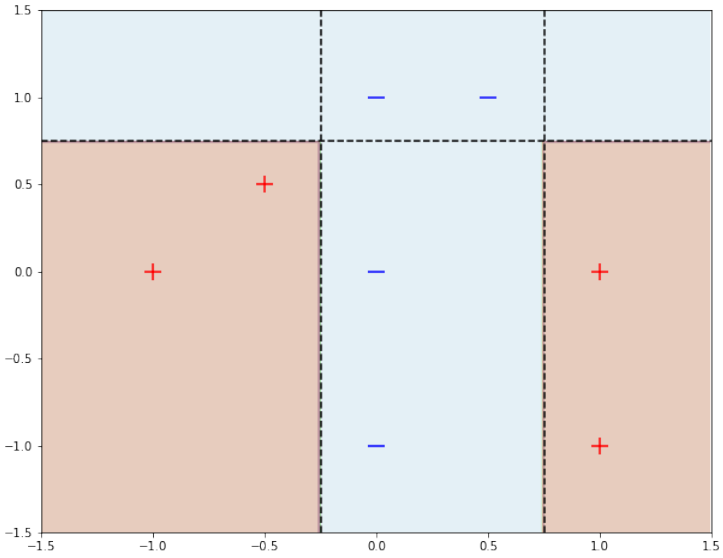
I then created the following functions

```
def learner1(x1, x2):
    if x1 < -0.25:
        return 1
    else:
        return -1
learner1 = np.vectorize(learner1)

def learner2(x1, x2):
    if x2 >= 0.75:
        return 1
    else:
        return -1
learner2 = np.vectorize(learner2)

def learner3(x1, x2):
    if x1 >= 0.75:
        return 1
    else:
        return -1
learner3 = np.vectorize(learner3)

def ada_boost(x1, x2, a):
    return np.sign(
        a[0] * learner1(x1, x2)
        + a[1] * learner2(x1, x2)
        + a[2] * learner3(x1, x2))

def errors(d, p, y):
    return (d * (p != y)).sum()

def alpha(e):
    return 0.5 * np.log((1 - e)/e)

def D(d, a, y, p):
    return d * np.exp(-a * y * p)

def Z(d):
    return d.sum()
```

I used the functions to calculate each iteration, resulting in the following table.

Iteration	e(t)	a(t)	Z(t)	Dt(1)	Dt(2)	Dt(3)	Dt(4)	Dt(5)	Dt(6)	Dt(7)	Dt(8)
1	0.250000	0.549306	0.866025	0.125000	0.125000	0.125000	0.125000	0.125	0.125	0.125000	0.125000
2	0.833333	-0.804719	0.745356	0.083333	0.083333	0.083333	0.083333	0.250	0.250	0.083333	0.083333
3	0.100000	1.098612	0.600000	0.050000	0.050000	0.050000	0.050000	0.150	0.150	0.250000	0.250000

I then plotted the decision boundary using a meshgrid.



(b) (15 points) What is the training error of this AdaBoost? Give a short explanation for why AdaBoost outperforms a single decision stump.

The training error in this AdaBoost was 0. The algorithm correctly classified all of the data points. A decision stump is a decision tree that splits on one value. AdaBoost outperforms a single decision stump by weighting the arrors of each decistion stump and encorporating those weights into the next iteration when calculating the next decision stump. This allows the algorthim to find the next decision stump by focusing on the points not classified correctly in the prior iteration.

2. **Linear regression: bias-variance tradeoff, CV, and variable selection** (30 points)

Consider a dataset with n data points (x^i, y^i), $x^i \in \mathbb{R}^n$, following from the following linear model:

$$y^i = {\beta^*}^T x^i + \epsilon^i, \quad i = 1, \ldots, m,$$

where ϵ^i are i.i.d. Gaussian noise with zero mean and variance σ^2, and β^* is the true parameter. Consider the ridge regression as follows:

$$\hat{\beta}(\lambda) = \arg\min_{\beta} \left\{ \frac{1}{m} \sum_{i=1}^{m} (y^i - \beta^T x^i)^2 + \lambda \|\beta\|_2^2 \right\}, \tag{1}$$

where $\lambda \geq 0$ is the regularized parameter.

(a) (5 points) Find the closed form solution for $\hat{\beta}(\lambda)$ and its distribution conditioning on $\{x^i\}$ (i.e., treat them as fixed).

$\frac{\partial \hat{\beta}(\lambda)}{\partial \beta} = \arg\min_\beta \left\{ -\frac{2}{m} Xy + \frac{2}{m} XX^T\beta + \frac{2\lambda}{m}\beta \right\}$

$\frac{\partial \hat{\beta}(\lambda)}{\partial \beta} = \arg\min_\beta \left\{ -2X^T(y - \beta^T X) + 2\lambda\beta \right\} = 0$

$\hat{\beta}(\lambda) = (X^T X + \lambda I)^{-1} X^T y$

$\mathbb{E}[\hat{\beta}] = (X^T X + \lambda I)^{-1} X^T X \beta^*$, where β^* represents the parameters of X, therefore replacing y.

$Var[\hat{\beta}] = \sigma^2 (X^T X + \lambda I)^{-1} X^T X [(X^T X + \lambda I)^{-1}]^T$

Since ϵ^i is i.i.d. Gaussian, y is also a Gaussian, implying that $\hat{\beta}$ is also Gaussian, so

$$\hat{\beta} \sim \mathcal{N}(\mathbb{E}[\hat{\beta}], Var[\hat{\beta}])$$

(b) (5 points) Calculate the bias $\mathbb{E}[x^T\hat{\beta}(\lambda)] - x^T\beta^*$ as a function of λ and some fixed test point x.

bias $\mathbb{E}[x^T\hat{\beta}(\lambda)] - x^T\beta^* = \mathbb{E}[x^T(X^TX + \lambda I)^{-1}X^T y] - x^T\beta^*$

Using Probability Theory (thank you simulation notes!)

$\mathbb{E}[\alpha y] = \alpha \mathbb{E}[y] = \alpha X \beta^*$

The above

$\mathbb{E}[x^T(X^TX + \lambda I)^{-1}X^T y] - x^T\beta^*$

becomes

$x^T(X^TX + \lambda I)^{-1}X^T X\beta^* - x^T\beta^*$

(c) (5 points) Calculate the variance term $\mathbb{E}\left[\left(x^T\hat{\beta}(\lambda) - \mathbb{E}[x^T\hat{\beta}(\lambda)]\right)^2\right]$ as a function of λ.

$Var[x^T\hat{\beta}] = x^T Var[\hat{\beta}] x$
$Var[\hat{\beta}] = \sigma^2 (X^TX + \lambda I)^{-1} X^T X [(X^TX + \lambda I)^{-1}]^T$
$Var[x^T\hat{\beta}] = x^T \sigma^2 (X^TX + \lambda I)^{-1} X^T X [(X^TX + \lambda I)^{-1}]^T x$

(d) (5 points) Use the results from parts (b) and (c) and the bias-variance decomposition to analyze the impact of λ in the mean squared error. Specifically, which term dominates when λ is small, and large, respectively?

$$MSE = [bias\mathbb{E}[x^T\hat{\beta}(\lambda)] - x^T\beta^*]^2 + \mathbb{E}\left[\left(x^T\hat{\beta}(\lambda) - \mathbb{E}[x^T\hat{\beta}(\lambda)]\right)^2\right] + \epsilon$$

$$= [x^T(X^TX + \lambda I)^{-1}X^TX\beta^*(\lambda) - x^T\beta^*]^2 + x^T\sigma^2(X^TX + \lambda I)^{-1}X^TX[(X^TX + \lambda I)^{-1}]^Tx$$

When λ is large the bias dominates, when λ is small, the variance dominates. This is what is known as the bias variance tradeoff.

(e) (5 points) Now suppose we have $m = 100$ samples. Write a pseudo-code to explain how to use cross validation to find the optimal λ.

Below is a description of the procedure using leave one out cross validation (LOOCV).

i. Define a range for the penalty parameter λ
ii. m = 100, for i in {1 to M}, Divide the set into samples {1, ..., m} leaving out i and, {i} respectively
iii. Fit the ridge regression model for each λ using the training set to obtain $\hat{\beta}_{-i}(\lambda)$ and $\hat{\sigma}^2_{-i}(\lambda)$
iv. Evaluate the prediction error of $\left||Y_i - X_i, *\hat{\beta}_{-i}\right|$.
v. repeat steps 1 through 3 until iteration is complete
vi. average the prediction performances of the test sets to get the cross validated log likelihood
vii. the value of λ that maximizes the cross validated log likelihood is the the optimal λ.

(f) (5 points) Explain if we would like to perform variable selection, how should we change the regularization term in Equation (2) to achieve this goal.

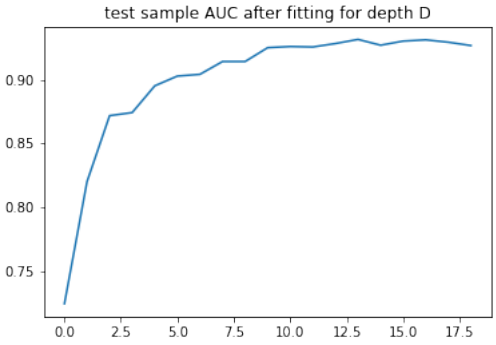
$$\theta^r = \arg\min_{\theta} L(\theta) = \left\{\frac{1}{m}\sum_{i=1}^{m}(y^i - \theta^T x^i)^2 + \lambda\|\theta\|_1\right\} \tag{2}$$

3. **Random forest and one-class SVM for email spam classifier** (40 points)

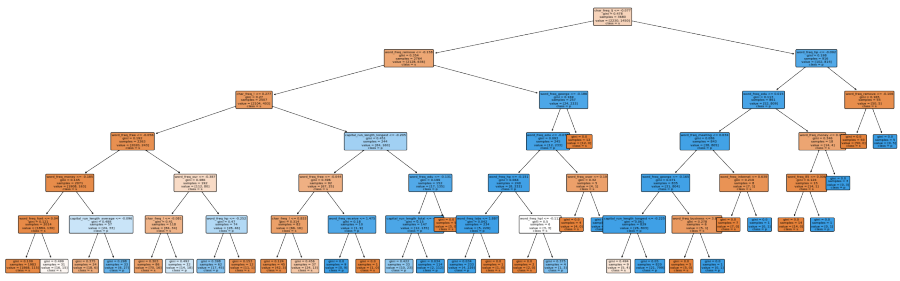
Your task for this question is to build a spam classifier using the UCR email spam dataset `https://archive.ics.uci.edu/ml/datasets/Spambase` came from the postmaster and individuals who had filed spam. Please download the data from that website. The collection of non-spam emails came from filed work and personal emails, and hence the word 'george' and the area code '650' are indicators of non-spam. These are useful when constructing a personalized spam filter. You are free to choose any package for this homework. Note: there may be some missing values. You can just fill in zero.

(a) (10 points) Build a CART model and visualize the fitted classification tree.

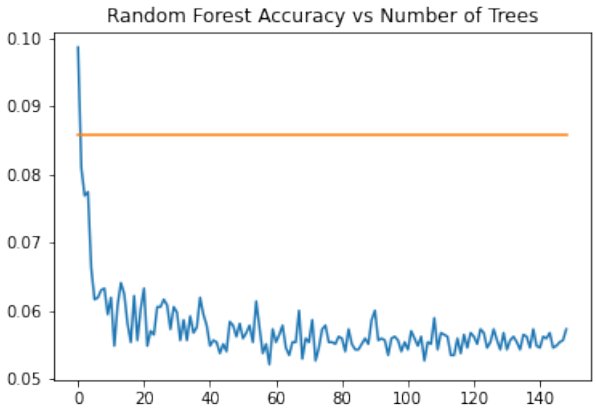
I first tuned my tree depth measuring the Area Under the ROC Curve and decided on a max depth of 6 (5 + 1 since index starts at 0).



```
|--- char_freq_$ <= 0.06
|   |--- word_freq_remove <= 0.05
|   |   |--- char_freq_! <= 0.51
|   |   |   |--- word_freq_free <= 0.20
|   |   |   |   |--- word_freq_money <= 0.01
|   |   |   |   |   |--- class: 0
|   |   |   |   |--- word_freq_money >  0.01
|   |   |   |   |   |--- class: 1
|   |   |   |--- word_freq_free >  0.20
|   |   |   |   |--- word_freq_our <= 0.05
|   |   |   |   |   |--- class: 0
|   |   |   |   |--- word_freq_our >  0.05
|   |   |   |   |   |--- class: 1
|   |   |--- char_freq_! >  0.51
|   |   |   |--- capital_run_length_longest <= 9.50
|   |   |   |   |--- word_freq_free <= 0.21
|   |   |   |   |   |--- class: 0
|   |   |   |   |--- word_freq_free >  0.21
|   |   |   |   |   |--- class: 1
|   |   |   |--- capital_run_length_longest >  9.50
|   |   |   |   |--- word_freq_edu <= 0.06
|   |   |   |   |   |--- class: 1
|   |   |   |   |--- word_freq_edu >  0.06
|   |   |   |   |   |--- class: 0
|   |--- word_freq_remove >  0.05
|   |   |--- word_freq_george <= 0.14
|   |   |   |--- word_freq_edu <= 0.12
|   |   |   |   |--- word_freq_hp <= 0.30
|   |   |   |   |   |--- class: 1
|   |   |   |   |--- word_freq_hp >  0.30
|   |   |   |   |   |--- class: 0
|   |   |   |--- word_freq_edu >  0.12
|   |   |   |   |--- word_freq_over <= 0.15
|   |   |   |   |   |--- class: 0
|   |   |   |   |--- word_freq_over >  0.15
|   |   |   |   |   |--- class: 1
|   |   |--- word_freq_george >  0.14
|   |   |   |--- class: 0
|--- char_freq_$ >  0.06
|   |--- word_freq_hp <= 0.40
|   |   |--- word_freq_edu <= 0.20
|   |   |   |--- word_freq_meeting <= 0.16
|   |   |   |   |--- word_freq_george <= 0.21
|   |   |   |   |   |--- class: 1
|   |   |   |   |--- word_freq_george >  0.21
|   |   |   |   |   |--- class: 0
|   |   |   |--- word_freq_meeting >  0.16
|   |   |   |   |--- word_freq_technology <= 0.17
|   |   |   |   |   |--- class: 0
|   |   |   |   |--- word_freq_technology >  0.17
|   |   |   |   |   |--- class: 1
|   |   |--- word_freq_edu >  0.20
|   |   |   |--- word_freq_money <= 0.43
|   |   |   |   |--- char_freq_; <= 0.19
|   |   |   |   |   |--- class: 0
|   |   |   |   |--- char_freq_; >  0.19
|   |   |   |   |   |--- class: 1
|   |   |   |--- word_freq_money >  0.43
|   |   |   |   |--- class: 1
|   |--- word_freq_hp >  0.40
|   |   |--- word_freq_remove <= 0.08
|   |   |   |--- class: 0
|   |   |--- word_freq_remove >  0.08
|   |   |   |--- class: 1
```



(b) (15 points) Now also build a random forest model. Partition the data to use the first 80% for training and the remaining 20% for testing. Compare and report the test error for your classification tree and random forest models on testing data. Plot the curve of test error (total misclassification error rate) versus the number of trees for the random forest, and plot the test error for the CART model (which should be a constant with respect to the number of trees).



The error rate for The Decision Tree algorithm is 0.0858. Iterating through random forests of trees between 1 and 150, the error rate drops from 0.0986 with one decision tree, and starts to converge at 12 trees with an error rate of 0.0549.

(c) (15 points) Now we will use a one-class SVM approach for spam filtering. Partition the data to use the first 80% for training and the remaining 20% for testing. Extract all *non-spam* emails from the training block (80% of data you have selected) to build the one-class kernel SVM using RBF kernel (you can turn the kernel bandwidth to achieve good performance). Then apply it on the 20% of data reserved for testing (thus this is a novelty detection situation), and report the total misclassification error rate on these testing data.

I normalized the testing set using sklearn's standard scaler. I randomly split the data into 80% train and 20% test. I calculated my gamma using the median trick on the 80% training data (including spam and not spam). I then also calculation my spam ratio (spam / total count) in order to tune the nu parameter. I fit only the not spam data from my 80% training sample into the One-Class SVM Classifier, then predicted the remaining 20% testing set. The total misclassification rate was 0.4234. The default gamma='scale' parameter in sklearn's one class SVM classifier performed better than using the median trick, and had a total misclassification rate of 0.4083.