ISYE 6740 HW 6

Mohammed Khalid Siddiqui

November 4, 2024

1 Conceptual Questions

1. What's the main difference between boosting and bagging? The random forest belongs to which type?

   Both boosting and bagging are ensemble methods used to improve the performance of machine learning models by combining predictions from multiple models.

   Bagging, also known as bootstrap aggregating, tries to reduce variance by training multiple models independently on different subsets of data which are created through sampling with replacement (also known as bootstrapping), and then aggregating predictions, either by averaging (regression) or voting (classification).

   Boosting, by contrast, focuses on reducing bias by training models in a sequential manner. Each model trained tries to correct the errors made by the previous models, thereby giving more weight to misclassified instances.

   Bagging trains each model independently and aggregates their outputs, which helps stabilize predictions and prevent overfitting. Boosting adjusts the weights of training samples based on the performance of prior models, leading to a more complex decision boundary that can capture intricate patterns.

   Bagging methods often use simple models (like decision trees), while boosting typically employs weak learners, which are models that perform slightly better than random guessing.

   Random forests, an extension of decision trees, belong to the bagging category. A random forest comprises of multiple decision trees using random subsets of the data and random subsets of features. It combines their predictions to achieve better accuracy and robustness against overfitting.

2. List several ways to prevent overfitting in CART.

   CART (Classification and Regression Trees) is a greedy decision tree algorithm used to predict outcomes in both classification and regression tasks. It constructs a tree structure by recursively partitioning the data based on feature values, where each internal node represents a feature test, and each leaf node signifies a predicted class or value.

   The algorithm tries to maximize the reduction in Gini impurity, another impurity measure, or variance—at each split until a stopping criterion such as maximum depth or minimum samples per leaf, is met.

   To prevent overfitting with CART, several strategies can be implemented.

   Pruning reduces the complexity of the tree by removing branches that offer little predictive power on unseen data, improving generalization.

   Setting a maximum depth limits how deep the tree can grow, ensuring it captures only essential patterns.

Establishing minimum samples per leaf and minimum samples for splitting prevents the creation of leaves or splits based on too few data points.

Cross-validation can be employed to assess the model's performance across different data subsets, guiding parameter adjustments for better generalization.

Carefully selecting input features helps to reduce irrelevant or noisy features, simplifying the model and enhancing its predictive capability.

By applying these techniques, CART can improve its ability to generalize to new data.

3. Explain how we control the data-fit complexity in the regression tree. Name at least one hyperparameter that we can turn to achieve this goal.

   In a regression tree, we control data-fit complexity primarily by tuning hyperparameters that dictate how the tree is constructed.

   Two key hyperparameters used for this purpose are the maximum depth of the tree, and the minimum number of samples required to split an internal node.

   By setting a lower maximum depth, we prevent the tree from growing too complex and overfitting to the training data.

   Similarly, increasing the minimum samples required for a split ensures that each node has enough data to make reliable predictions, thus reducing variance.

   These adjustments help achieve a better balance between fitting the training data and generalizing to unseen data.

4. Explain how OOB errors are constructed and how to use them to understand a good choice for the number of trees in a random forest. Is OOB a test error or training error, and why?

   The Out-of-Bag (OOB) error is a method for estimating the error of a random forest model by using data that was not included in each individual tree's training sample. In a random forest, each tree is trained on a bootstrap sample of the data, meaning that some data points are left out, or "out-of-bag," for that tree.

   To compute the OOB error, each data point is passed through only those trees in which it was not included during training, and the predicted label is determined based on these trees. The OOB error is the average error rate across all data points based on these out-of-bag predictions.

   To determine a good choice for the number of trees, we can observe the OOB error rate as the forest size increases. Initially, as more trees are added, the OOB error typically decreases and then stabilizes once enough trees are included, indicating a sufficient model complexity.

   OOB error is a test error because it uses data that each tree has not seen during training, which makes it an unbiased estimate of the model's predictive accuracy on new data.

5. Explain what the bias-variance tradeoff means in the linear regression setting.

   In linear regression, the bias-variance tradeoff describes the balance between two types of errors that affect a model's predictive performance.

   Bias refers to the error introduced by assumptions made in the model, specifically the tendency of the model to simplify or underfit the data. In linear regression, a high-bias model might miss important relationships in the data, leading to systematically inaccurate predictions.

   Variance refers to the model's sensitivity to fluctuations in the training data. A high-variance model captures noise as well as signal, resulting in overfitting and poor generalization to new data.

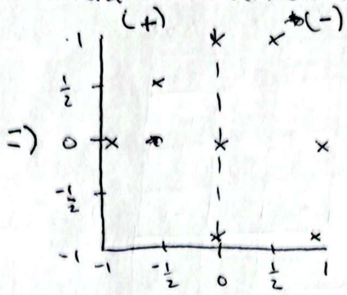
   The tradeoff arises because reducing bias by making the model more complex typically increases variance, and vice versa. In linear regression, the simplest models may have high bias but low variance, while more complex polynomial regression models might have low bias but high variance. The goal is to find the model complexity that minimizes the total error, which includes both bias and variance, achieving a good balance for accurate predictions on unseen data.

2 Adaboost

1. For each iteration t = 1, 2, 3, compute Epsilon, Alpha, Z and D by hand for each iteration t (i.e., show the calculation steps) and draw the decision stumps on the figure (you can draw this by hand).

point	x_1	x_2	y_i
X_1	-1	0	1
X_2	$-\frac{1}{2}$	$\frac{1}{2}$	1
X_3	0	1	-1
X_4	$\frac{1}{2}$	1	-1
X_5	1	0	1
X_6	1	-1	1
X_7	0	-1	-1
X_8	0	0	-1

Iteration 1: vertical line at $x_1 = 0$ $D_1(i) = \frac{1}{8}$ for each i

$\Rightarrow$ 

$h_1(i) \to \{X_1, X_2\} : 1$

$\{X_3, X_4, X_5, X_6, X_7, X_8\} = -1$

Misclassified: $\{X_5, X_6\}$

$\epsilon_1 = \sum_{i=1}^{8} \frac{1}{8} \cdot \mathbb{I}\{\text{misclassified}\}$

$= \frac{1}{8} \cdot 2 = \frac{1}{4}$

$\alpha_1 = \frac{1}{2} \ln\left(\frac{1-\frac{1}{4}}{\frac{1}{4}}\right) = \frac{1}{2}\ln(3) \sim 0.54931$

$Z_1 = \sum_{i=1}^{8} \frac{1}{8} \cdot e^{-0.54931 \cdot \overset{\text{actual}}{y_i} \cdot \underset{\text{predicted}}{h_t(x_i)}}$ ← If y_i and $h_t(x_i)$ are mismatched, then -1, else 1

$e^{-0.54931} \approx 0.57735 \to y_i = h_t(x_i)$

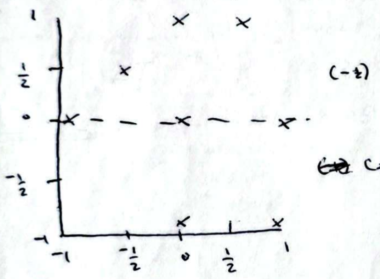
$e^{0.54931} \approx 1.73206 \to y_i \neq h_1(x_i)$ $Z_1 = \frac{1}{4} \cdot e^{0.54931} + \frac{3}{4} e^{-0.54931}$

$Z_1 \approx 0.86603$

$D_2(1) \approx 0.14434 \cdot 0.57735 \approx 0.08335$ $D_2(2) = 0.08335$ $D_2(3) = 0.08335$ $D_2(4) = 0.08335$

$D_2(5) = 0.14434 \cdot 1.73206 \approx 0.25001$ $D_2(6) = 0.25001$ $D_2(7) = 0.08335$ $D_2(8) = 0.08335$

Iteration 2: horizontal line at $x_2 = 0$



$h_2(i) \to \{X_1, X_5, X_6, X_7, X_8\} = 1$

$\{X_2, X_3, X_4\} = -1$

$\alpha_2 = \frac{1}{2}\ln\left(\frac{1-0.25005}{0.25005}\right)$

≈ 0.54917

$e^{-0.54917} \approx 0.57743 \to y_i = h_2(x)$

$e^{0.54917} \approx 1.73182 \to y_i \neq h_2(x)$

Misclassified: $\{X_2, X_7, X_8\}$

$\epsilon_2 = D_2(2) + D_2(7) + D_2(8)$

$= 0.08335 \cdot 3$

$= 0.25005$

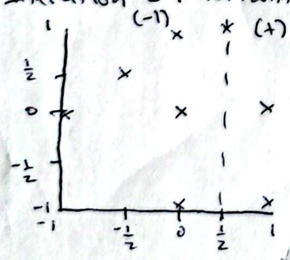
$Z_2 = 3 \cdot 0.08335 \cdot 1.73182 + 2 \cdot 0.25001 \cdot 0.57743 + 3 \cdot 0.08335 \cdot 0.57743 \approx 0.86615$

$D_3(1) \approx \frac{0.08335 \cdot 0.57743}{0.86615} \approx 0.05557$ $D_3(2) = \frac{0.08335 \cdot 1.73182}{0.86615} \approx 0.16665$ $D_3(3) \approx 0.05557$

$D_3(4) = 0.05557$ $D_3(5) = \frac{0.25001 \cdot 0.57743}{0.86615} \approx 0.16667$ $D_3(6) = 0.16667$ $D_3(7) = 0.16665$

$D_3(8) \approx 0.16665$

Iteration 3: horizontal line at $x_1 = \frac{1}{2}$



$h_3(i) \to \{X_5, X_6\} = 1$

$\{X_1, X_2, X_3, X_4, X_7, X_8\} = -1$

Misclassified: $\{X_1, X_2\}$

$\epsilon_3 = D_3(1) + D_3(2)$

$= ~~0.16667 \cdot 2~~$ 0.22222

$= ~~0.33334~~$

$\alpha_3 = \frac{1}{2}\ln\left(\frac{1-\epsilon_3}{\epsilon_3}\right)$

$= \frac{1}{2}\ln(~~[illegible]~~ 3.50004)$

$\approx ~~[illegible]~~$ 0.62639

$e^{-\alpha_3} \approx ~~0.33334~~$ $0.53452 \to y_i = h_3(x_i)$

$e^{\alpha_3} \approx ~~2.99993~~$ $1.87085 \to y_i \neq h_3(x_i)$

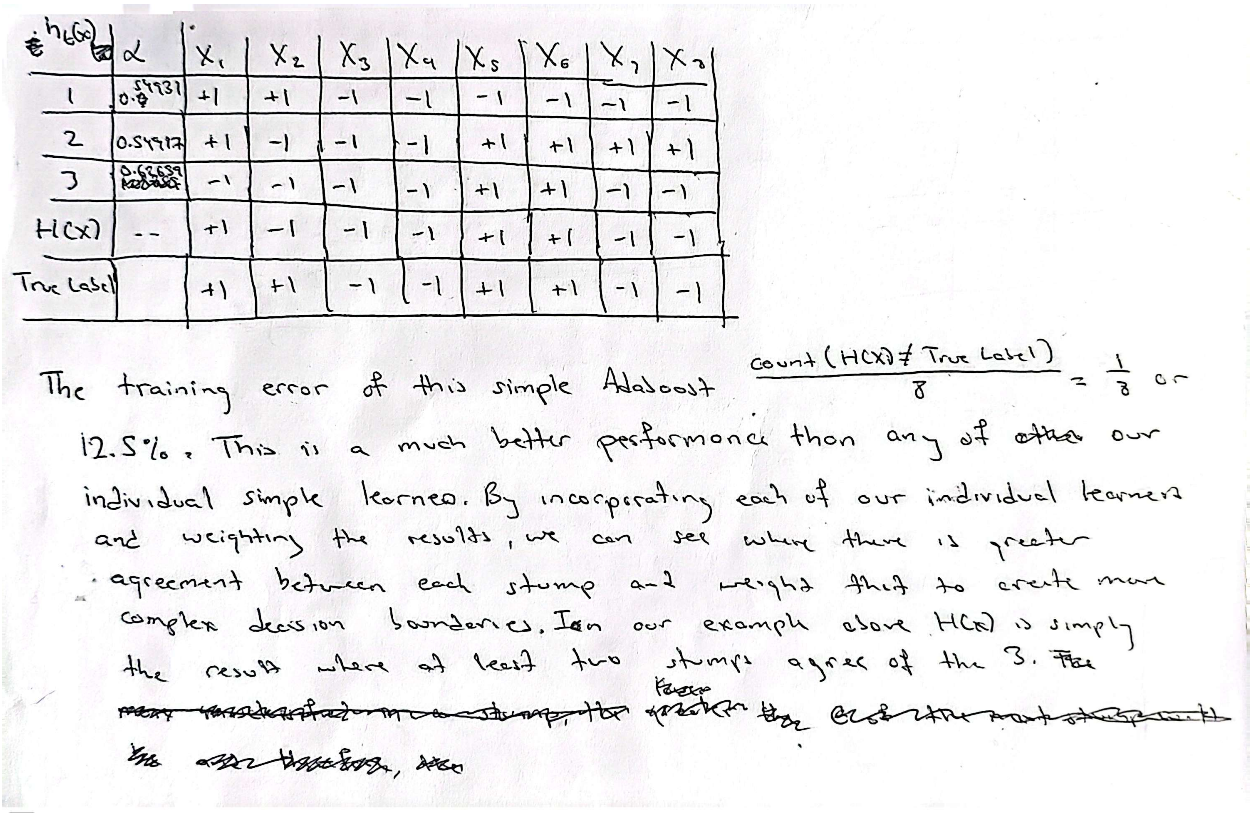
$Z_3 = 0.05557 \cdot ~~2.99993~~$ 1.87085 $+ 0.16665 \cdot ~~2.99993~~$ 1.87085
$+ 2 \cdot 0.05557 \cdot ~~0.33334~~$ 0.53452 $+ 2 \cdot 0.11667 \cdot ~~0.33334~~$ 0.53452 $+ 2 \cdot 0.11665 \cdot ~~0.33334~~$ 0.53452

$\approx ~~1.23934~~$ 0.83148

$H(x) = \text{sign}(0.54931 \cdot h_1(x) + 0.54917 \cdot h_2(x) + 1.09859 \cdot h_3(x))$

t	ϵ_t	α_t	Z_t	$D_t(1)$	$D_t(2)$	$D_t(3)$	$D_t(4)$	$D_t(5)$	$D_t(6)$	$D_t(7)$	$D_t(8)$
1	0.25	0.54931	0.86603	0.125	0.125	0.125	0.125	0.125	0.125	0.125	0.125
2	0.25005	0.54917	0.86615	0.08335	0.08335	0.08335	0.08335	0.25001	0.25001	0.08335	0.08335
3	~~0.33334~~ 0.22222	~~[illegible]~~ 0.62639	~~[illegible]~~ 0.83148	0.05557	0.16665	0.05557	0.05557	0.16667	0.16667	0.16665	0.16665

2. What is the training error of this AdaBoost? Give a short explanation for why AdaBoost outperforms a single decision stump.



$h_t(x)$	α	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8
1	0.54931	+1	+1	-1	-1	-1	-1	-1	-1
2	0.54917	+1	-1	-1	-1	+1	+1	+1	+1
3	0.62659	-1	-1	-1	-1	+1	+1	-1	-1
H(x)	--	+1	-1	-1	-1	+1	+1	-1	-1
True Label		+1	+1	-1	-1	+1	+1	-1	-1

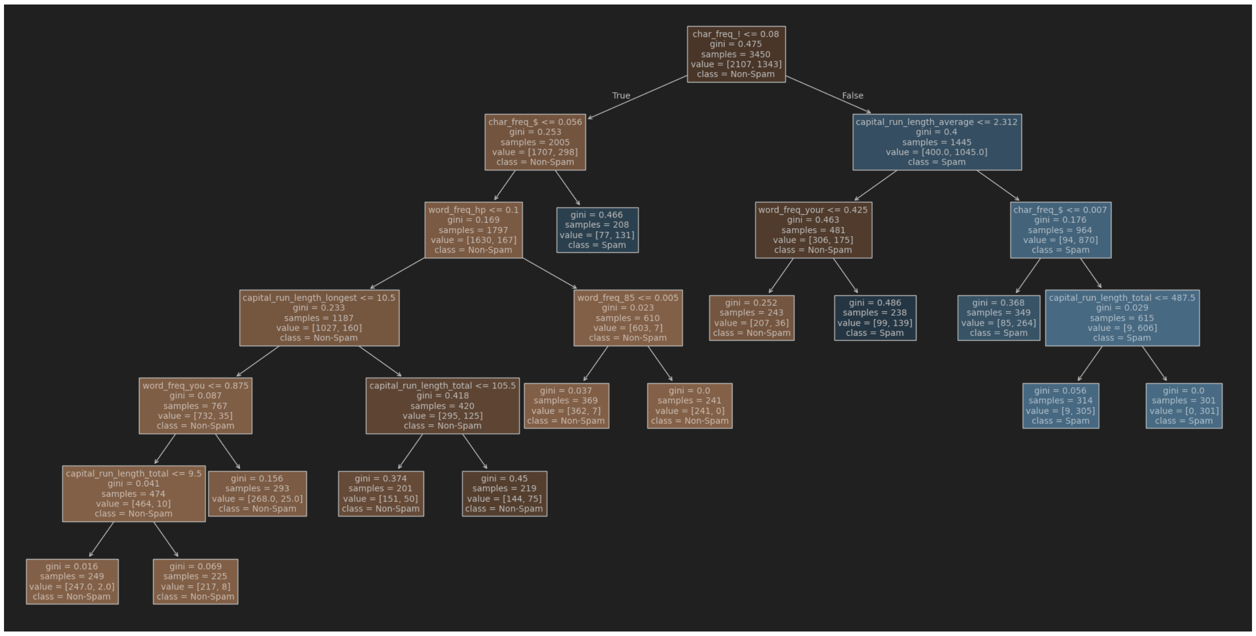
The training error of this simple Adaboost $\frac{\text{count}(H(x) \neq \text{True Label})}{8} = \frac{1}{8}$ or 12.5%. This is a much better performance than any of ~~other~~ our individual simple learners. By incorporating each of our individual learners and weighting the results, we can see where there is greater agreement between each stump and weight that to create more complex decision boundaries. In our example above H(x) is simply the result where at least two stumps agree of the 3. ~~The~~ [illegible]

Scanned with CamScanner

3 Random forest and one-class SVM for email spam classifier

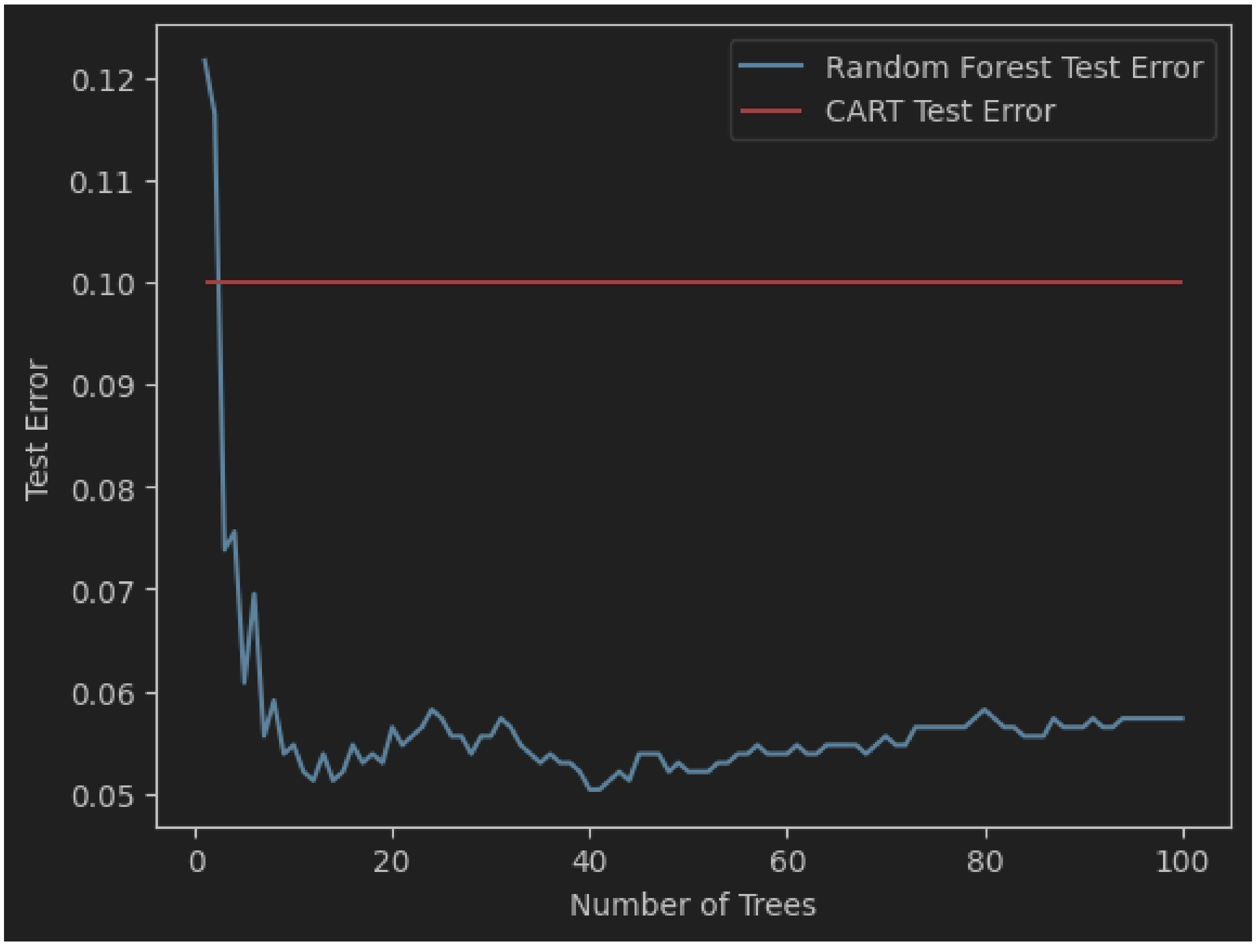
1. Build a CART model and visualize the fitted classification tree. Please adjust the plot size/font as appropriate to ensure it is legible. Pruning this tree is not required, and if done reasoning should be stated as to why.

   The below CART Tree was generated where the minimum samples for a leaf was set to 200. The purpose for this was simply to generate the image below such that it can be displayed. For analytical purposes, a decision tree was created without any pruning or hyper parameter changes applied.



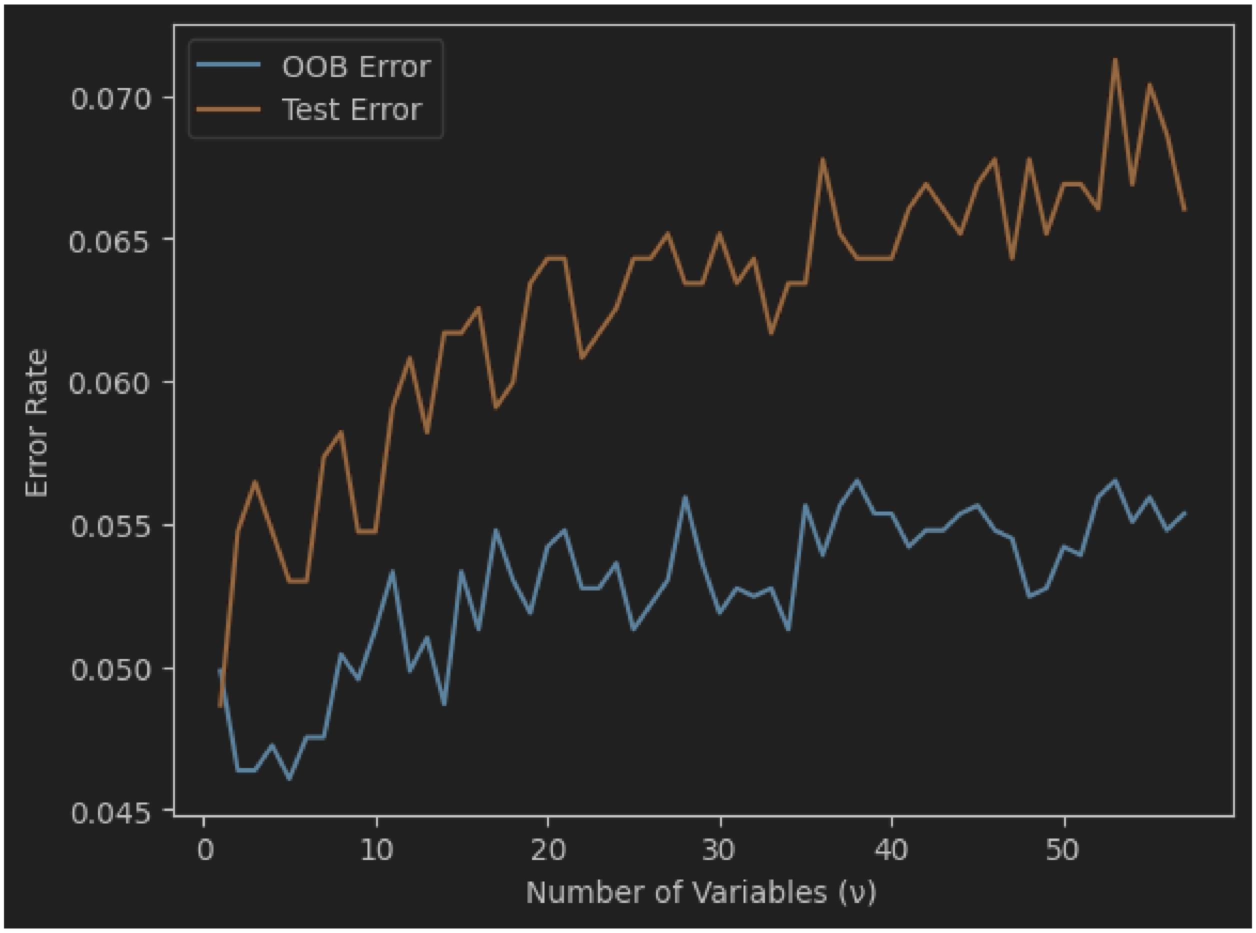
2. Now, also build a random forest model. Randomly shuffle the data and partition to use 75% for training and the remaining 25% for testing. Compare and report the test error for your classification tree and random forest models on testing data. Plot the curve of test error (total misclassification error 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).

   Our baseline CART model has a test error of 0.09991311902693312. The test error of our Random Forest model drop significantly when we first start increasing the number of trees and as we increase the number of trees, it tends to remain between 0.05 and 0.06. The figure plotting the test-errors of our random forest model with an increasing number of trees is plotted alongside the CART test error.



3. Fit a series of random-forest classifiers to the data to explore the sensitivity to the parameter v (the number of variables selected at random to split). Plot both the OOB error as well as the test error against a suitably chosen range of values for .

   For the maximum features parameters, it seems the fewer, the better. As we increase the number of maximum allowed features to consider when determining the best split, the OOB Error and Test Error seem to increase.



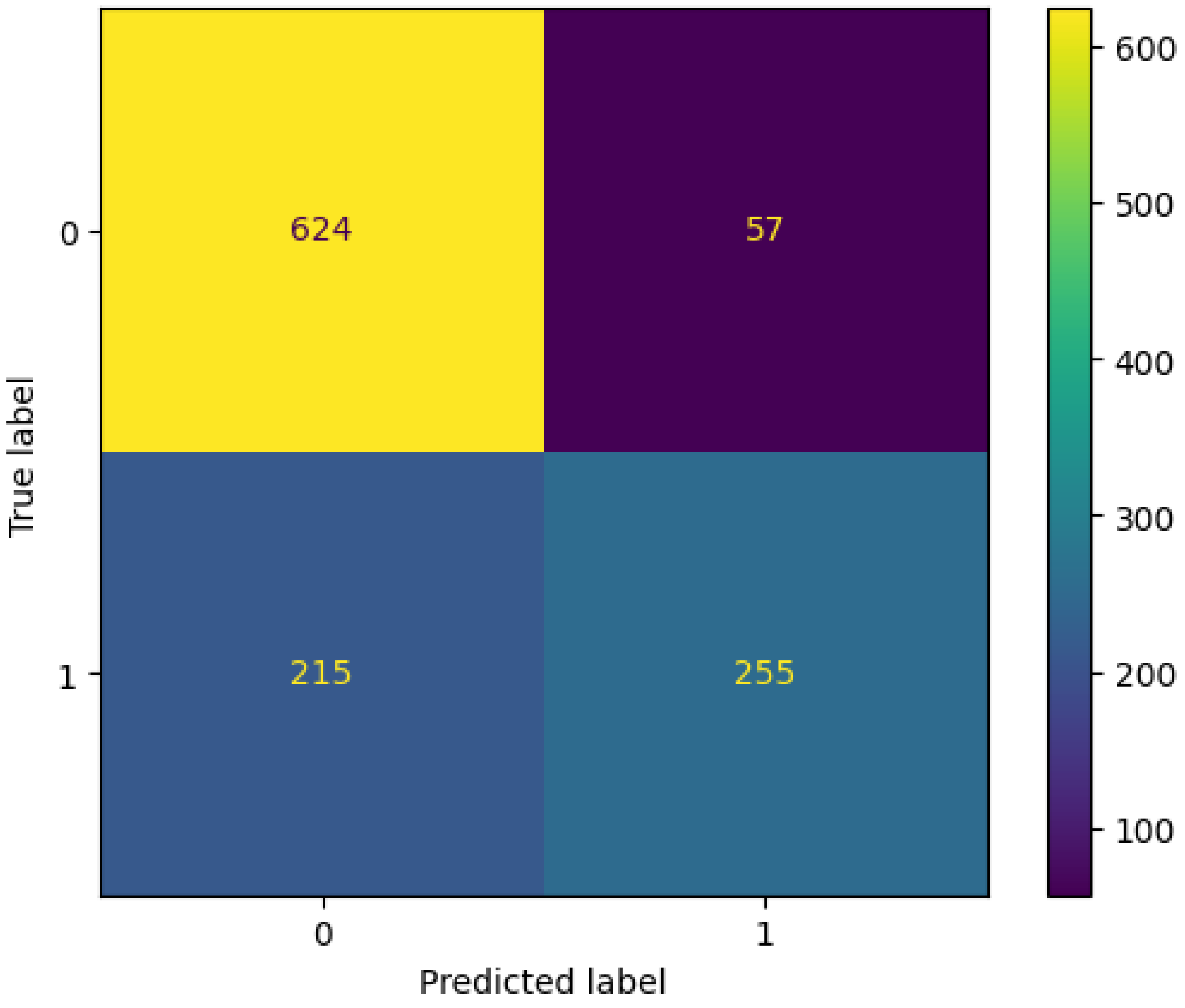
4. Now, we will use a one-class SVM approach for spam filtering. Randomly shuffle the data and partition to use 75% for training and the remaining 25% for testing. Extract all non-spam emails from the training block (75% of data you have selected) to build the one-class kernel SVM using RBF kernel. Then apply it to the 25% of data reserved for testing (thus, this is a novelty detection situation), and report the total misclassification error rate on these testing data. Tune your models appropriately to achieve good performance, i.e. by tuning the kernal bandwidth or other parameters. Give a short explanation on how you reached your final error rate and whether you feel this is a good model.

   The two parameters I chose to tune were Gamma (kernel coefficient which controls influence of individual data points on SVM boundary) and nu (a lower bound for number of samples acting as support vectors and an upper bound for number of samples allowed on the wrong side of the hyperplane). I applied two methods. In the first method, I used GridSearchCV and then chose the best set of paramters to use on our testing set. In the second method, I directly appleid each set of parameters, training the models on all of the training data and testing on the test set. The second method provided significantly better results, most likely because of larger datasets to use as a 'validation' to determine parameters on representative set.

   Using the second method, the best nu was determined to be 0.05, the best gamma involved setting the parameter to 'scale', which involves using 1 / (n_features * X.var()) as value of gamma. The associated misclassification rate is 0.2363 or roughly 23.63

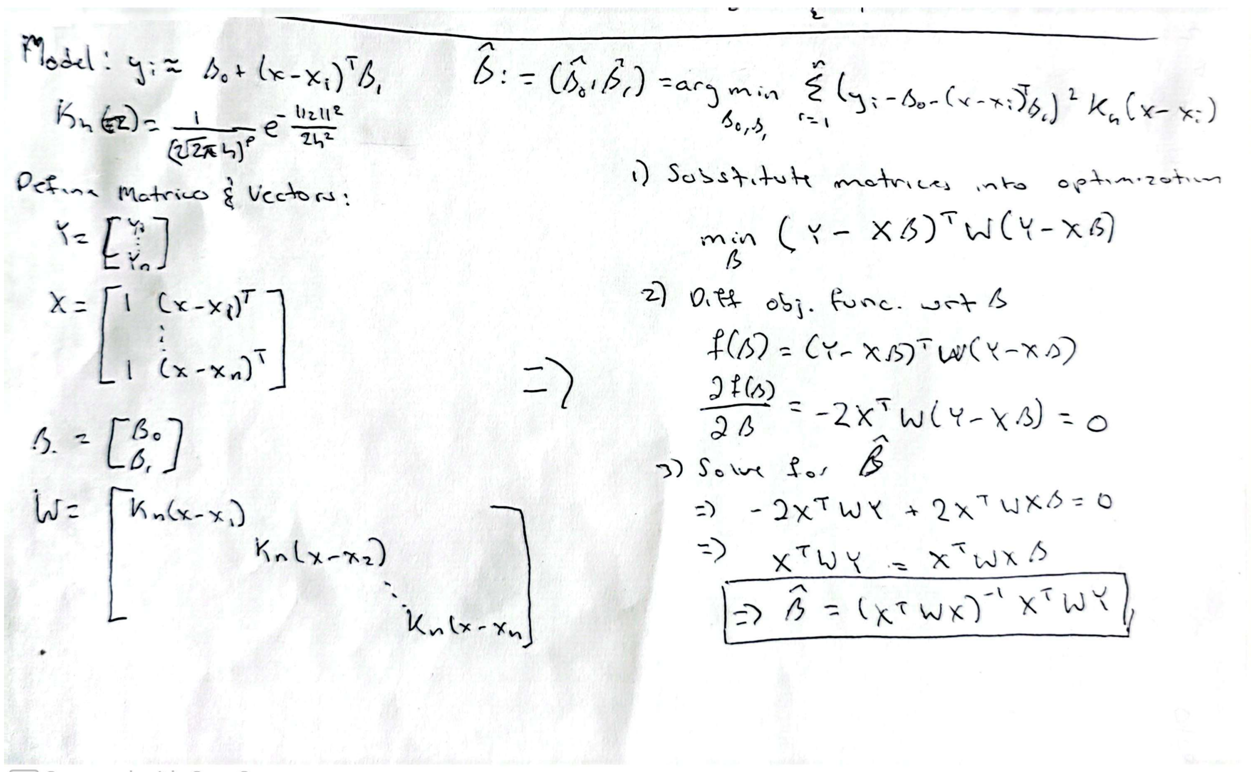
   Whether this is a sufficient model depends on the context of the problem and the business-determined acceptance for error. Is it ok to misclassify roughly 23% of all emails? Additionally, it will be helpful to determine whether more spam or non-spam emails are misclassified.

By creating a confusion matrix of the svm with the best parameters, we can see that far more spam emails were predicted as non-spam than non-spam emails predicted as spam, by a factor of 4 almost. If we're ok with allowing more spam to filter through as long as fewer true emails are missed, this model's performance may be acceptable.



4 Locally weighted linear regression and bias-variance tradeoff

1. To find the matrix form of the solution for Beta in locally weighted linear regression, we first create our intial matrix representations for each relevant element in our optimization problem, then substitute and solve.



Model: $y_i \approx \beta_0 + (x-x_i)^T\beta_1$

$K_h(z) = \frac{1}{(\sqrt{2\pi}h)^p} e^{-\frac{\|z\|^2}{2h^2}}$

Define Matrices & Vectors:

$$Y = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$$

$$X = \begin{bmatrix} 1 & (x-x_1)^T \\ & \vdots \\ 1 & (x-x_n)^T \end{bmatrix}$$

$$\beta = \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix}$$

$$W = \begin{bmatrix} K_h(x-x_1) & & \\ & K_h(x-x_2) & \\ & & \ddots \\ & & & K_h(x-x_n) \end{bmatrix}$$

$\Rightarrow$

$$\hat{\beta} := (\hat{\beta}_0, \hat{\beta}_1) = \arg\min_{\beta_0,\beta_1} \sum_{i=1}^{n} (y_i - \beta_0 - (x-x_i)^T\beta_1)^2 K_h(x-x_i)$$

1) Substitute matrices into optimization

$$\min_{\beta} (Y - X\beta)^T W (Y - X\beta)$$

2) Diff obj. func. wrt β

$$f(\beta) = (Y - X\beta)^T W (Y - X\beta)$$

$$\frac{\partial f(\beta)}{\partial \beta} = -2X^T W (Y - X\beta) = 0$$

3) Solve for $\hat{\beta}$

$\Rightarrow -2X^TWY + 2X^TWX\beta = 0$

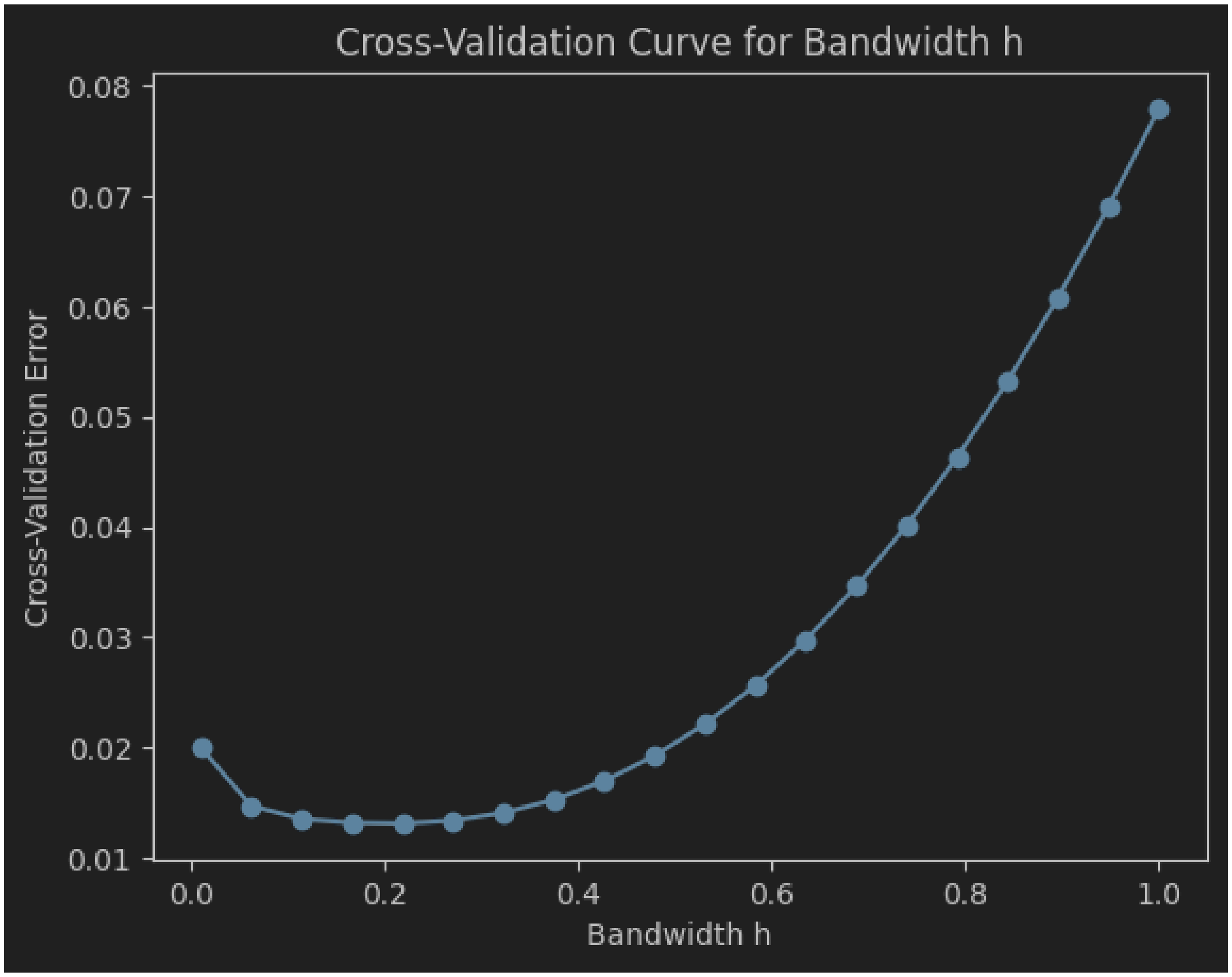
$\Rightarrow X^TWY = X^TWX\beta$

$$\boxed{\Rightarrow \hat{\beta} = (X^TWX)^{-1}X^TWY}$$

2. Use the data.mat file to perform local linear weighted linear regression. Using 5-fold cross validation to tune the bandwidth parameter h, report a plot showing your cross validation curve and provide your optimal bandwidth, h.

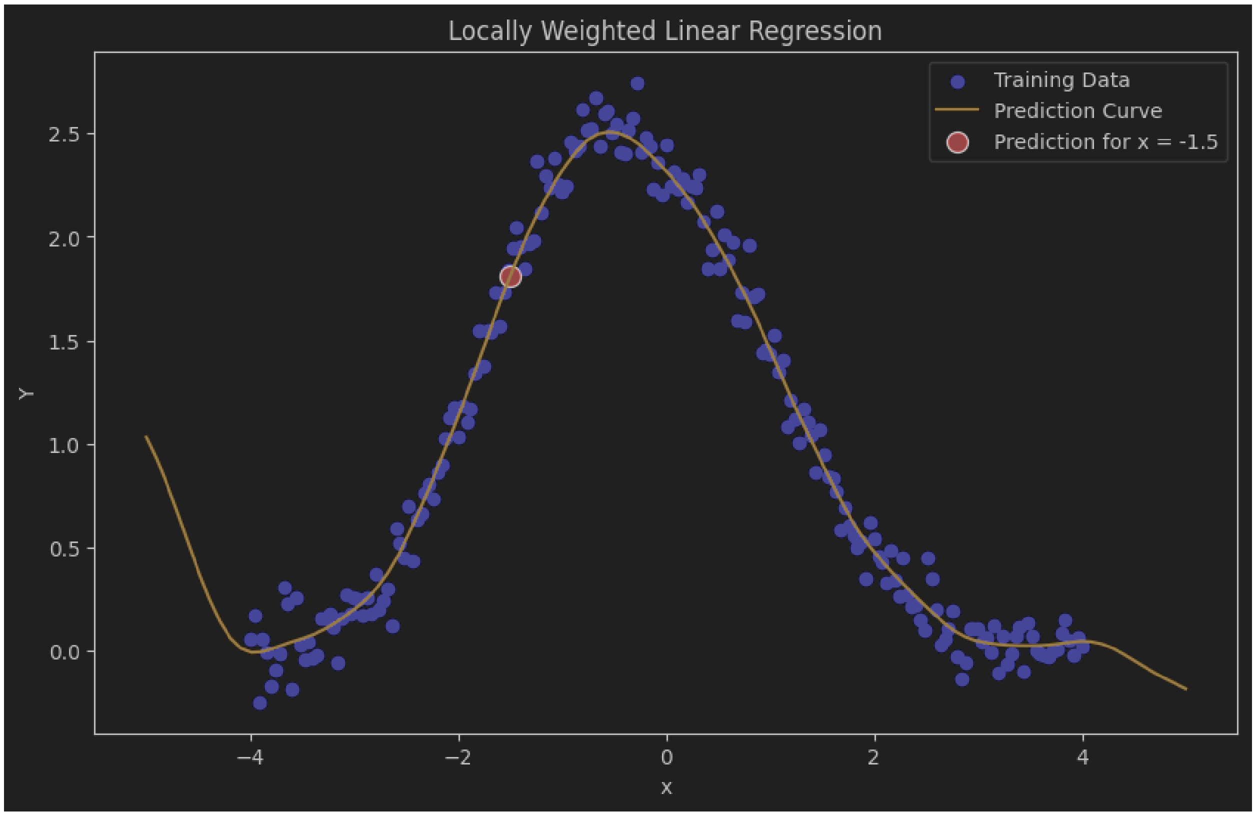
   The first function created is the gaussian kernel function. The second function is the one which implements the locally weighted linear regression in the form we derived in the previous section. Lastly, we apply CV.

   The optimal bandwidth h from the range of values tested was determined to be 0.21842105263157896. The CV error for different values of h is shown below.



3. Using the tuned hyper-parameter h to make a prediction for x = -1.5. Provide the predicted y value, and report a plot showing your training data, prediction curve, and a marker indicating your prediction.

   The plot below shows the visualized data and our Locally Weighted Linear Regression Line. Our model predicts the y value for x = -1.5 to be 1.8105.



Sources

1. https://scikit-learn.org/dev/modules/generated/sklearn.svm.OneClassSVM.html
2. https://scikit-learn.org/1.5/modules/generated/sklearn.ensemble.RandomForestClassifier.html
3. https://scikit-learn.org/dev/modules/generated/sklearn.model_selection.GridSearchCV.html
4. https://scikit-learn.org/dev/modules/generated/sklearn.metrics.ConfusionMatrixDisplay.html
5. https://sefiks.com/2018/11/02/a-step-by-step-adaboost-example/
6. Course Lectures