ISYE 6740 – HW #6

Jing Ma

2025-03-28

Problem 1:

1.1

Below are the main differences between boosting and bagging ([1]):

1. Boosting is an ensemble framework that trains weak learners using weighted sample points. Based on the performance of the previous weak learner, the framework assigns more weights to the misclassified data points and less weights to the correctly classified samples. In comparison, bagging is an ensemble strategy that trains weak learners using bootstrap datasets (typically the same dataset size as the original one) with replacements. Each bootstrap dataset produces a different decision tree.
2. After training all weak learners, boosting again assigns weight to each weak learner and combines them linearly. In contrast, in the bagging approach, for regression trees, we take an average of all trees' predictions as the final output, whereas for classification trees, the majority votes determine the final classification.

Given that random forests are constructed using bootstrap datasets with replacement rather than the original dataset, random forest is a bagged ensemble of decision trees.

1.2

Below are ways to prevent overfitting in CART:

1. Require that each leaf contains a minimum of x amount of data points;
2. Control the depth of the tree so that it doesn't grow too large;
3. Use cost-complexity pruning to prune a large regression tree defined as ([2]):

$$C_\alpha(T) = \sum_{m=1}^{|T|} N_m Q_m(T) + \alpha \, |T|$$

where α controls the tradeoff between tree size and goodness of fit. We can find a good α value using cross-validation;

4. Use Gini index or cross-entropy to grow classification trees as splitting criteria and apply max depth or min samples per leaf as constraints.

1.3

According to the lecture ([1]), we know that the data-fit complexity is defined as:

$$C_\alpha(S) = \sum_{j=1}^{|J|} \sum_{x_i \in R_j} \left(y_i - \hat{c}_j\right)^2 + \alpha \, |J|$$

As briefly mentioned in question 2, we can control it via tuning α as large values of α result in smaller trees and smaller values of α lead to better fit to the training data. To find a good estimation of α, we can use five-fold cross-validation for instance.

1.4

To construct OOB error, we first obtain the predictions for the ith observation using each of the trees in which that observation was OOB. For regression problem, we average the predicted responses. For classification problem, we take the majority votes.

To compute the OOB error for regression, we perform the above process for each of the n observations and we can compute the OOB MSE. Similarly, we can compute the classification error if for classification problem.

Based on the OOB error plot that we have seen in the lecture, the curve of OOB error plateaus at some point and stops showing meaningful improvements with hundreds of trees. Therefore, based on the plot, we can judgmentally pick the number of trees where the curve flattens. Note that James et al. ([3]) mentioned that using a very large number of trees will not lead to overfitting but in practice we select the number of trees sufficiently large that the error has settled down.

OOB is a valid estimate of test error since the trees are tested using the observations that they were not fit with. James et al. ([3]) pointed out that if the number of trees is sufficiently large, OOB error is virtually equivalent to leave-one-out cross-validation error.

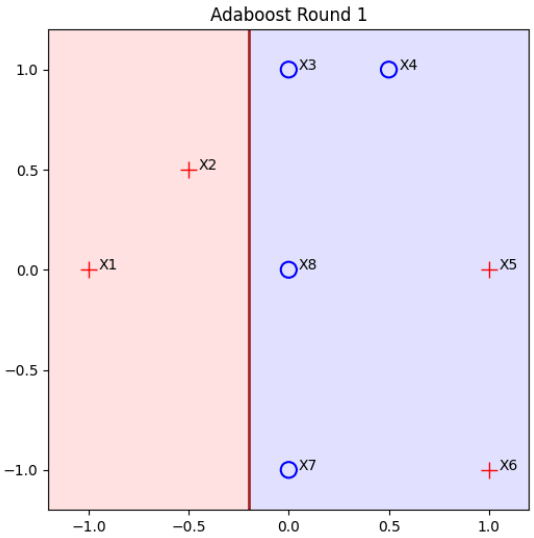
1.5

Based on the lecture ([4]), we know that the expected loss consists of squared bias, variance and irreducible noise. In the linear regression setting, high bias and low variance generally suggests underfitting, or as Prof. David Goldsman puts it, we are very confident that the predictions are wrong. Conversely, low bias and high variance typically suggests overfitting, and hence the model won't generalize well as it fits the training data too well, including the noise.

Problem 2:

2.1

In the first round of Adaboost, we placed the decision stump as displayed in the graph below:



And we set the initial weight of eight data points as

$$D_1(i) = \frac{1}{m} = \frac{1}{8}$$

Therefore, our table is initialized as below, which will be updated as we compute each values:

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				1/8	1/8	1/8	1/8	1/8	1/8	1/8	1/8
2											
3											

Based on the plot of the first round Adaboost, we observed that two red points, X_5 and X_6, were misclassified. We can first compute the error rate ε_1

$$\varepsilon_1 = \sum_{i=1}^{m} D_1(i)\mathbb{I}\{y^i \neq h_1(x^i)\} = \frac{2}{8} = 0.25$$

We can then compute α_1 and Z_1

$$\alpha_1 = \frac{1}{2}\ln\left(\frac{1-\varepsilon_1}{\varepsilon_1}\right) = \frac{1}{2}\ln\left(\frac{0.75}{0.25}\right) \approx 0.55$$

$$\begin{aligned} Z_1 = \sum_{i=1}^{m} D_1(i)\exp\{-\alpha_1 y^i h_1(x^i)\} &= \sum_{i=1}^{m} D_1(i) \times \begin{cases} e^{-\alpha_1} \text{ if } y^i = h_1(x^i) \\ e^{\alpha_1} \text{ otherwise} \end{cases} \\ &= 6 \times \frac{1}{8} \times e^{-0.55} + 2 \times \frac{1}{8} \times e^{0.55} \\ &\approx 0.87 \end{aligned}$$

Hence, the weights for X_1, X_2, X_3, X_4, X_7, and X_8 will be updated as following

$$D_2(i) = \frac{D_1(i)}{Z_1} \times e^{-\alpha_1} = \frac{1}{8} \div 0.87 \times e^{-0.55} \approx 0.083, \quad i \in \{1,2,3,4,7,8\}$$

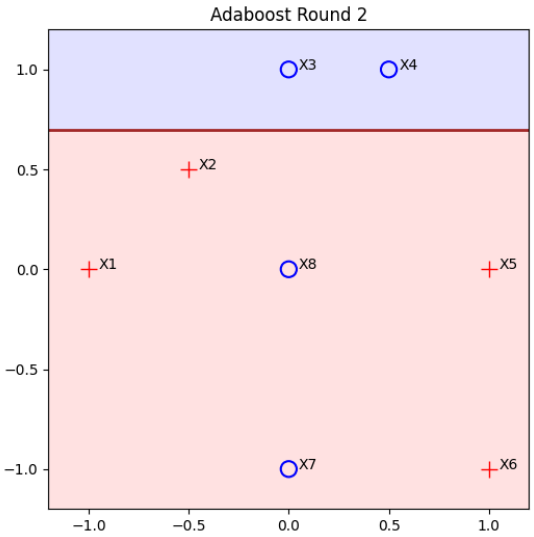
For X_5 and X_6, their weights will be updated as

$$D_2(i) = \frac{D_1(i)}{Z_1} \times e^{\alpha_1} = \frac{1}{8} \div 0.87 \times e^{0.55} \approx 0.25, \quad i \in \{5,6\}$$

Based on the results, we can update the table below

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	0.55	0.87	1/8	1/8	1/8	1/8	1/8	1/8	1/8	1/8
2				0.083	0.083	0.083	0.083	0.25	0.25	0.083	0.083
3											

Next in the round 2 of Adaboost, we placed the decision stump as shown in the plot



We can see that this time, two blue points (X_7, X_8) were misclassified.

We will repeat the same computation steps above. First, we calculate the error rate ε_2

$$\varepsilon_2 = 0.083 \times 2 = 0.166$$

Then we compute α_2 and Z_2

$$\alpha_2 = \frac{1}{2}\ln\left(\frac{1-0.166}{0.166}\right) \approx 0.81$$

$$Z_2 = 4 \times 0.083 \times e^{-0.81} + 2 \times 0.25 \times e^{-0.81} + 2 \times 0.083 \times e^{0.81} \approx 0.74$$

Next, we update the weights for X_1 through X_4

$$D_3(i) = 0.083 \div 0.74 \times e^{-0.81} \approx 0.05, \quad i \in \{1, 2, 3, 4\}$$

Similarly, the weights for X_5 and X_6 are

$$D_3(i) = 0.25 \div 0.74 \times e^{-0.81} \approx 0.15, \quad i \in \{5, 6\}$$

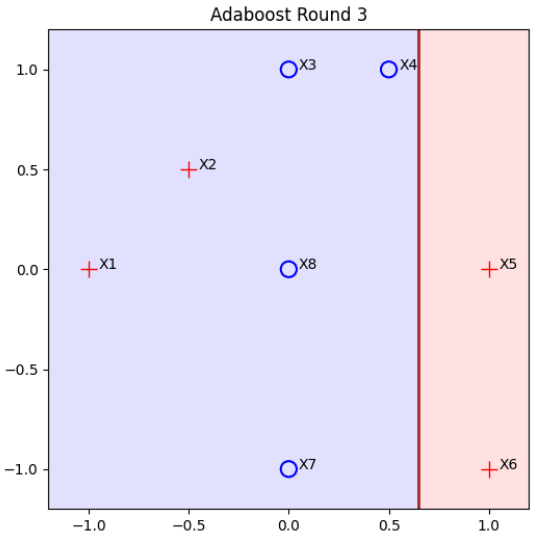
Lastly, the weights for X_7 and X_8 are

$$D_3(i) = 0.083 \div 0.74 \times e^{0.81} \approx 0.25, \quad i \in \{7, 8\}$$

Therefore, we can again update the table as following

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	0.55	0.87	1/8	1/8	1/8	1/8	1/8	1/8	1/8	1/8
2	0.166	0.81	0.74	0.083	0.083	0.083	0.083	0.25	0.25	0.083	0.083
3				0.05	0.05	0.05	0.05	0.15	0.15	0.25	0.25

In our last iteration, we have the decision stump as following



This time, we observed that two red points, X_1 and X_2, were misclassified. Following the same steps as above, we have

$$\varepsilon_3 = 0.05 \times 2 = 0.1$$

$$\alpha_3 = \frac{1}{2}\ln\left(\frac{0.9}{0.1}\right) \approx 1.1$$

$$\begin{aligned} Z_3 &= 2 \times 0.05 \times e^{1.1} + 2 \times 0.05 \times e^{-1.1} \\ &\quad + 2 \times 0.15 \times e^{-1.1} + 2 \times 0.25 \times e^{-1.1} \\ &\approx 0.6 \end{aligned}$$

Therefore, the table is updated as below

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	0.55	0.87	1/8	1/8	1/8	1/8	1/8	1/8	1/8	1/8
2	0.166	0.81	0.74	0.083	0.083	0.083	0.083	0.25	0.25	0.083	0.083
3	0.1	1.1	0.6	0.05	0.05	0.05	0.05	0.15	0.15	0.25	0.25

2.2

We know that the prediction depends on the sign of the outcome

$$H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_{t(x)}\right)$$

Further, we have three weak learners as following

$$h_1(x) = \begin{cases} +1 \text{ if } x \leq -0.2 \\ -1 \text{ otherwise} \end{cases}$$

$$h_2(x) = \begin{cases} +1 \text{ if } y \leq 0.7 \\ -1 \text{ otherwise} \end{cases}$$

$$h_3(x) = \begin{cases} +1 \text{ if } x \geq 0.65 \\ -1 \text{ otherwise} \end{cases}$$

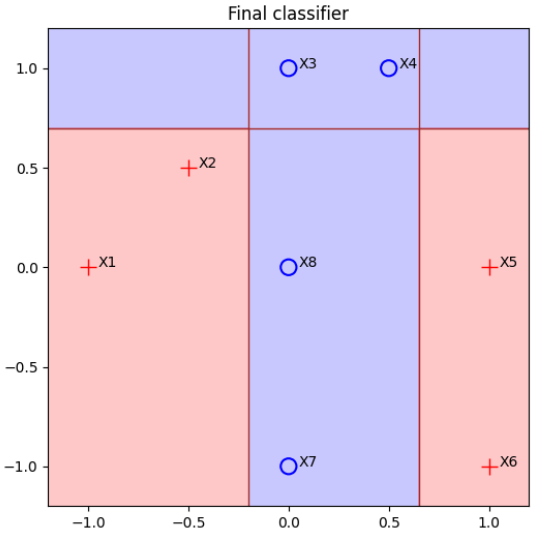
For instance, to compute the prediction for X_1

$$f(X_1) = \text{sign}(0.55 \times (+1) + 0.81 \times (+1) + 1.1 \times (-1)) = \text{sign}(0.26) = +1$$

We repeat the same computation steps for all eight data points and summarize the results in the table below

X_i	Label	h_1	h_2	h_3	$f(x)$	Prediction	Correct?
X_1	+1	+1	+1	−1	0.26	+1	Yes
X_2	+1	+1	+1	−1	0.26	+1	Yes
X_3	−1	−1	−1	−1	−2.46	−1	Yes
X_4	−1	−1	−1	−1	−2.46	−1	Yes
X_5	+1	−1	+1	+1	1.36	+1	Yes
X_6	+1	−1	+1	+1	1.36	+1	Yes
X_7	−1	−1	+1	−1	−0.84	−1	Yes
X_8	−1	−1	+1	−1	−0.84	−1	Yes

As we can see, we obtained 100% accuracy using Adaboost.

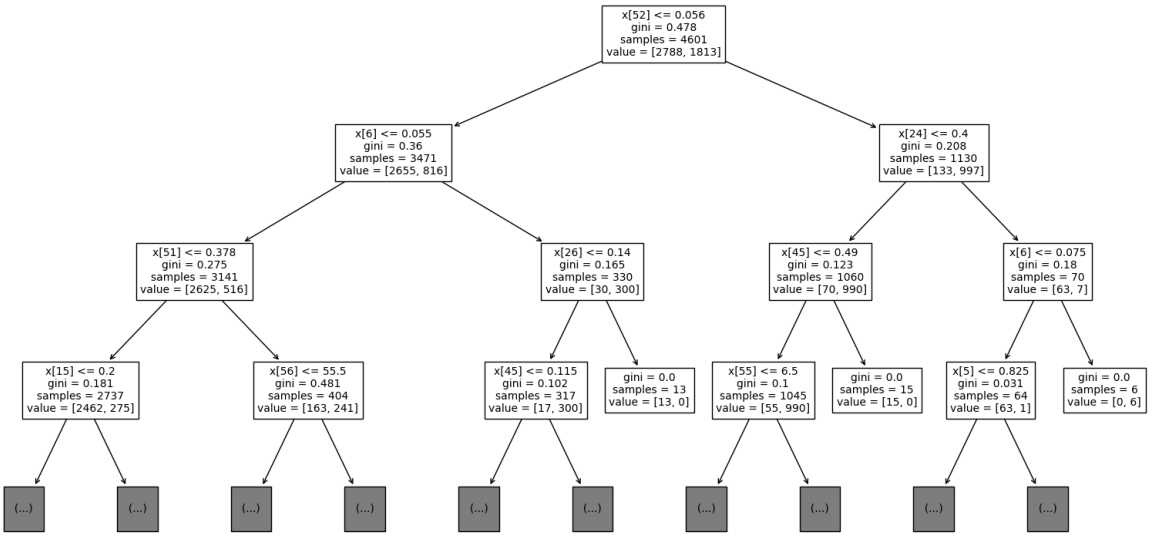


Adaboost outperforms a single decision stump because each new decision stump is trained to fix the errors made by the previous ones. From part 1, we can see that the misclassified data points are assigned more weights. So in the next round, it forces the stump to focus more on these samples. At the end, the final classifier combines all the weak learners, giving more weights to the better performing ones.

Problem 3:

3.1

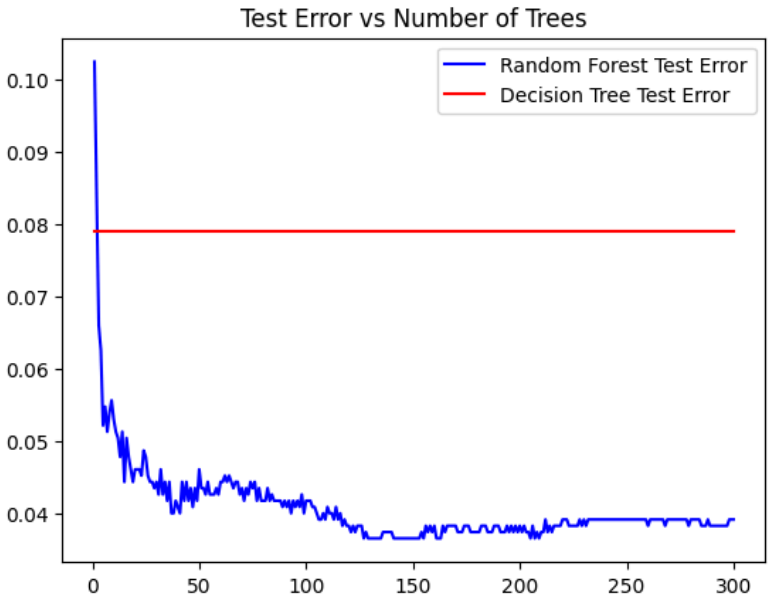
We are given the spambase dataset, which contains spam emails (1) or non-spam emails (0). In this question, we built a CART model using the entire dataset and showed the partial tree structure as following



In this part of the problem, no pruning has been performed and the CART model was built using the default settings of the DecisionTreeClassifier() from sklearn.

3.2

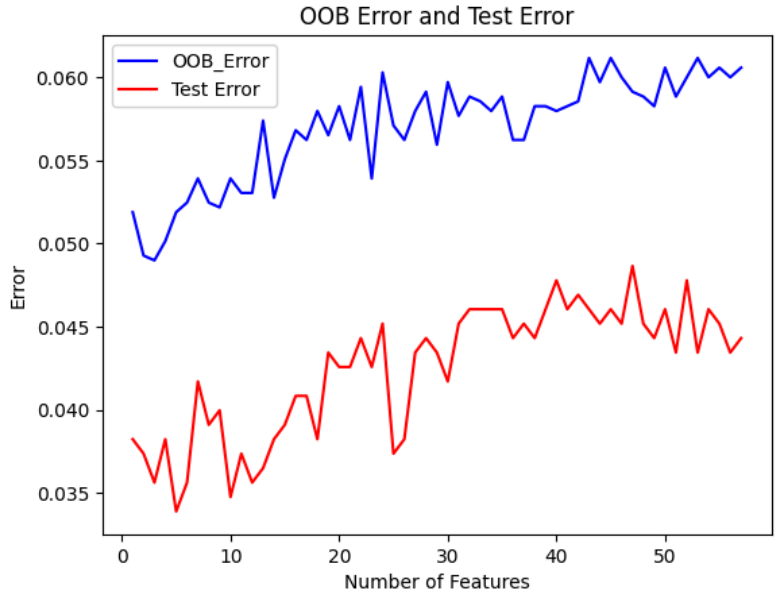
In this section, we first randomly shuffled the data and then set 75% for training and the remaining 25% for testing. We then built a random forest model using different number of tree values in the range of [1, 300]. Below is the curve of test error versus the number of trees for the random forest. We also plot the test error for the CART model but it's just a constant value as CART model is only a single tree.



As we can see, the test error appears to stabilize around 150 trees in the random forest model. Further, the random forest model shows better performance than the classification tree model given the smaller test error.

3.3

In this question, we experimented with number of features ranging from 1 to 57, and plotted the OOB error and test error for each number of variables selected at random to split. Below is the plot



As we can see, the curves for both OOB error and test error are quite noisy as the more features that we include. From the lecture, we know that the more features that we use to make the split, the higher the correlation among trees. Take an extreme example in which we use all 57 features, all trees start to look very similar as they are likely to make the same decision at each split, which cancels out the effect of variance reduction benefit. As a result, both OOB and test errors go up as the number of features increases.

3.4

Next, we use one-class SVM model to make predictions. Among the 75% training set, we find all the data points that are non-spam and fit the one-class SVM model. To find the best parameters of the model, we used GridSearchCV() and tested 15 'gamma' and 'nu' values. 'gamma' parameter controls the kernel bandwidth, while the 'nu' parameter controls the upper bound on outliers as well as lower bound on support vectors.

Based on the grid search result, we see that the best parameters are 'gamma' = 0.0001 and 'nu' = 0.01.

Since OneClassSVM() labels inliers 1 and outliers −1, in our case, we need to convert labels of 1 to 0 (non-spam), and conversely, labels of −1 to 1 (spam). We then compute the error rate against the true labels, noting error rate of 34%.

We can see that this model has much higher error rate compared to random forest. This might be that OneClassSVM has only "seen" what the normal data points look like, and therefore set up a boundary based on this knowledge. In comparison, in random forest, the model has explicitly learned both spam emails and non-spam emails, and therefore can generalize well.

Problem 4:

4.1

In this question, we are given data points $(x_i, y_i), i = 1, ..., n$. We are also given a Gaussian kernel function

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

We want to solve local linear regression given the bias $\beta_0 \in \mathbb{R}$ and the weights $\beta_1 \in \mathbb{R}^p$

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

To simplify the derivation process, let

$$X = \begin{pmatrix} 1 \\ x_1 \\ ... \\ x_n \end{pmatrix}, \quad X \in \mathbb{R}^{n \times (p+1)}$$

$$\beta = \begin{pmatrix} \beta_0 \\ \beta_1 \end{pmatrix}, \quad \beta \in \mathbb{R}^{(p+1) \times 1}$$

$$Y = \begin{pmatrix} y_1 \\ ... \\ y_n \end{pmatrix}, \quad Y \in \mathbb{R}^{n \times 1}$$

$$W = w_i I, \quad W \in \mathbb{R}^{n \times n}$$

where $w_i = \exp\left(-\frac{\|x - x_i\|^2}{2h^2}\right)$, the exponential term in the Gaussian kernel function. We can drop the $\frac{1}{\left(\sqrt{2\pi}h\right)^p}$ as it's a constant, which does not affect the result that we will derive using derivative.

Next, we define the loss function as

$$L(\beta) = \sum_{i=1}^{n} \left(y_i - \beta_0 - (x - x_i)^T \beta_1\right)^2 \cdot w_i = (Y - X\beta)^T W (Y - X\beta)$$

We then set the derivative of $L(\beta)$ to 0

$$\frac{\mathrm{d}L(\beta)}{\mathrm{d}\beta} = -2X^T W(Y - X\beta) = 0 \Longrightarrow X^T WY = X^T WX\beta$$

We obtain

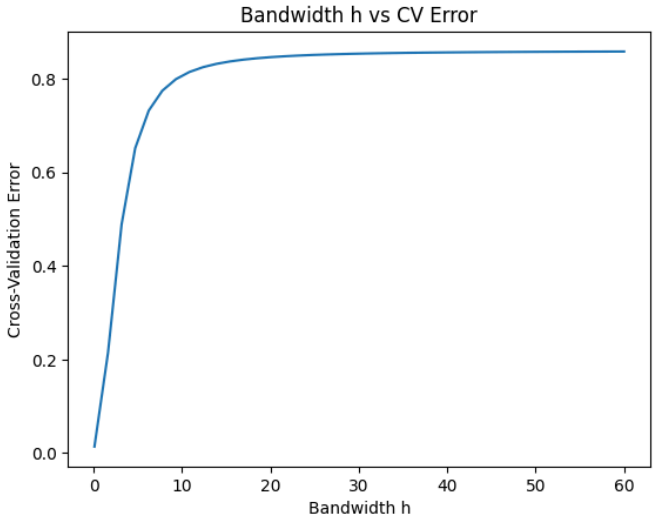
$$\beta = (X^T WX)^{-1} X^T WY$$

4.2

We used 5-fold cross validation to first split the entire dataset into 5 folds.

For each fold, we loop through the data points in the test fold and compute the weight matrix, W, for each test data, and compute the coefficients, β.

Then we use the β to make predictions for the test data, calculate the mean squared error, average the MSE for all 5 folds, which becomes the cross-validation error for each bandwidth h.

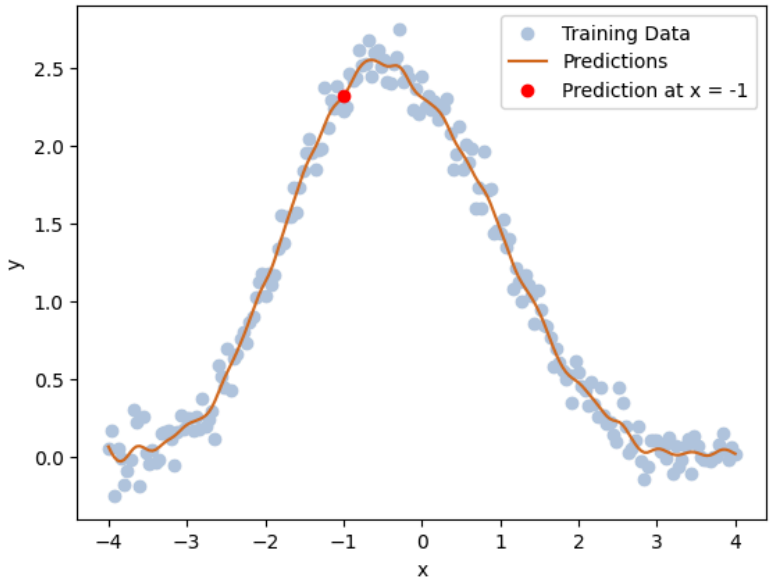


Note that as h increases, the cross-validation error curve goes up until plateaued. So our estimate of h is 0.1, which gives the lowest cross-validation error.

4.3

Using the same computation algorithm and the found optimal h value, we observed that when $x = -1$, our predicted value $y \approx 2.316$.

To derive the prediction curve, we computed the predicted values for all the x values. Below shows the plot of training data, prediction curve, and the red dot is our predicted value when $x = -1$:



As we can see, the prediction curve well captured the intrinsic pattern of the training data.

References:

1. Lecture 15 Decision tree and random forest lecture slides
2. The Elements of Statistical Learning
3. An Introduction to Statistical Learning with Applications in Python
4. Lecture 16 Bias-Variance Tradeoff and Cross-Validation lecture slides