Caltech CS/CNS/EE 155
Jayden Nyamiaka

Policies

• Due 9 PM PST, January 31th on Gradescope (using 4 late hours)

• You are free to collaborate on all of the problems, subject to the collaboration policy stated in the syllabus.

• In this course, we will be using Google Colab for code submissions. You will need a Google account.

Submission Instructions

• Submit your report as a single .pdf file to Gradescope, under ”Set 3 Report”.

• In the report, **include any images generated by your code** along with your answers to the questions.

• Submit your code by **sharing a link in your report** to your Google Colab notebook for each problem (see naming instructions below). Make sure to set sharing permissions to at least ”Anyone with the link can view”. **Links that can not be run by TAs will not be counted as turned in.** Check your links in an incognito window before submitting to be sure.

• For instructions specifically pertaining to the Gradescope submission process, see https://www.gradescope.com/get_started#student-submission.

Google Colab Instructions

For each notebook, you need to save a copy to your drive.

1. Open the github preview of the notebook, and click the icon to open the colab preview.

2. On the colab preview, go to File → Save a copy in Drive.

3. Edit your file name to “lastname firstname set problem”, e.g.”yue_yisong_set3_prob2.ipynb” 1

1 Decision Trees [30 Points]

Relevant materials: Lecture 5

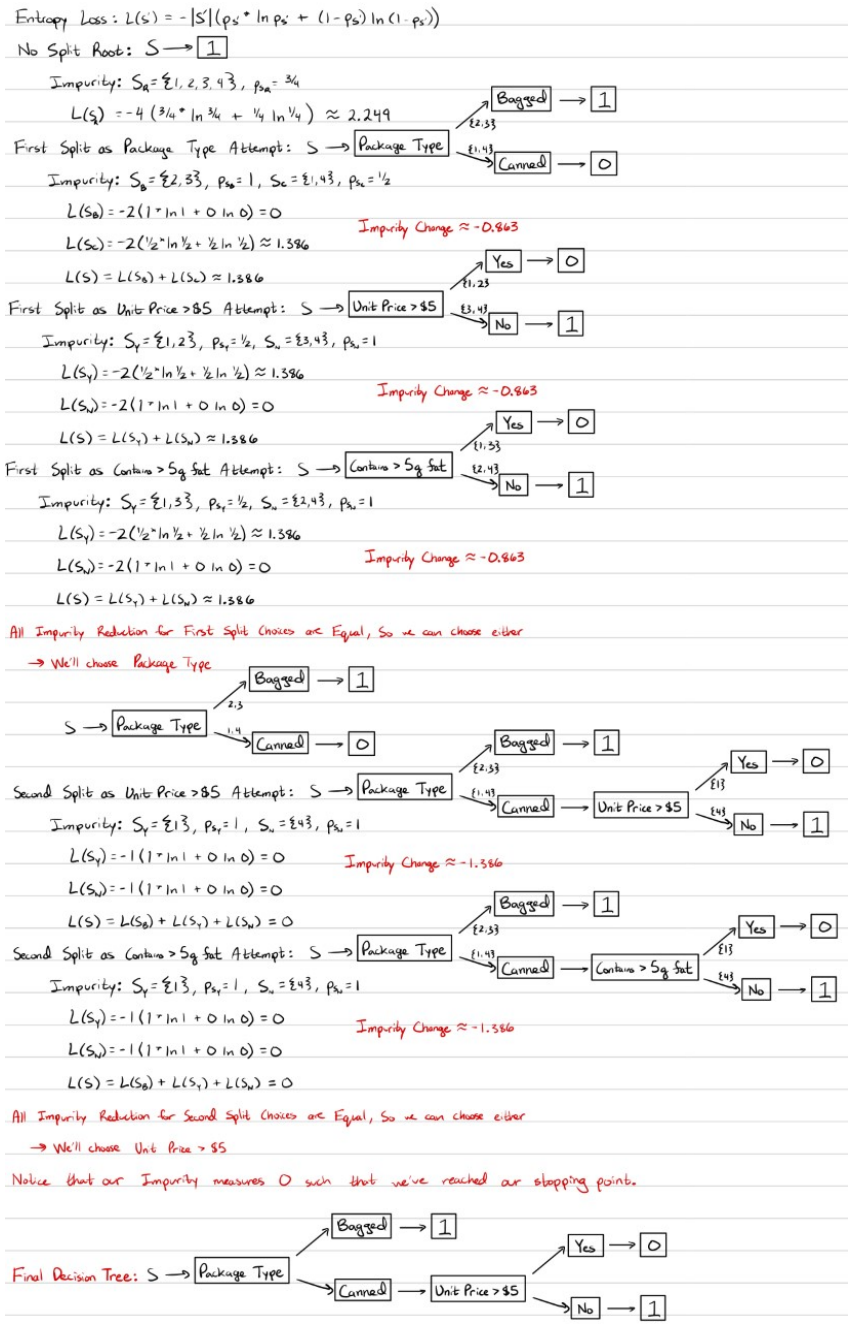
Problem A [7 points]: Consider the following data, where given information about some food you must predict whether it is healthy:

See GitHub

Train a decision tree by hand using top-down greedy induction. Use *entropy* (with natural log) as the impurity measure. Since the data can be classified without error, the stopping criterion will be no impurity in the leaves.

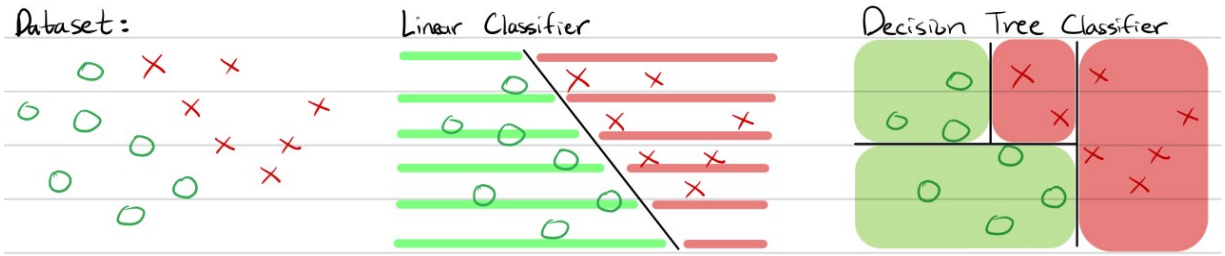
Submit a drawing of your tree showing the impurity reduction yielded by each split (including root) in your decision tree.

Solution A:

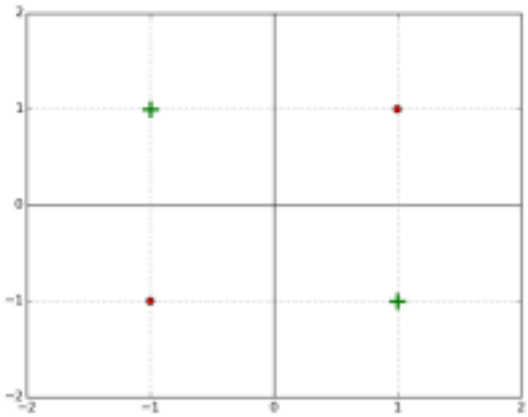


Problem B [4 points]: Compared to a linear classifier, is a decision tree always preferred for classification problems? Briefly explain why or why not. If not, draw a simple 2-D dataset that can be perfectly classified by a simple linear classifier but which requires an overly complex decision tree to perfectly classify.

Solution B: A decision tree is not always preferred over a linear classifier. The decision trees we work with must be axis-aligned; this means they model diagonal boundaries poorly whereas a linear classifier can split data at any angle. Because of this, any linearly separable dataset that has a clear cut diagonal boundary can be modeled much better and more concisely with a linear classifier. A decision tree is still able to perfectly model the data with no impurity, but it must do so using many splits along its axes yielding an overly complex tree, wasted boundaries, and probably overfitting. It is worth noting though that some decision trees outside the scope of this class do not need to remain axis-aligned; for these more advanced decision trees, this diagonal drawback must be reconsidered.



Problem C [15 points]: Consider the following 2D data set:



i. [5 points]: Suppose we train a decision tree on this dataset using top-down greedy induction, with the Gini index as the impurity measure. We define our stopping condition to be if no split of a node results in any reduction in impurity. Submit a drawing of the resulting tree. What is its classification error ((number of misclassified points) / (number of total points))?

ii. [5 points]: Submit a drawing of a two-level decision tree that classifies the above dataset with zero classification error. (You don't need to use any particular training algorithm to produce the tree.)

Is there any impurity measure (i.e. any function that maps the data points under a particular node in a tree to a real number) that would have led top-down greedy induction with the same stopping condition to produce the tree you drew? If so, give an example of one, and briefly describe its pros and cons as an impurity measure for training decision trees in general (on arbitrary datasets).

iii. [5 points]: Suppose there are 100 data points in some 2-D dataset. What is the largest number of unique thresholds (i.e., internal nodes) you might need in order to achieve zero classification training error (on the training set)? Please justify your answer.

Solution C:

i.

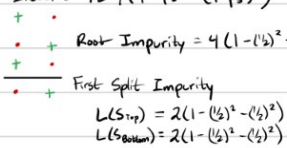
Two Options for the Decision Tree

1. S ⟶ [1]
2. S ⟶ [0]

Both have Classification Error = $\frac{2 \text{ misclassified points}}{4 \text{ total points}} = 1/2 = 50\%$

Work:

Gini Index: $|S'|\left(1-p_{S'}^2-(1-p_{S'})^2\right)$



Root Impurity $= 4\left(1-(1/2)^2-(1/2)^2\right) = 4(1-1/2) = 2$

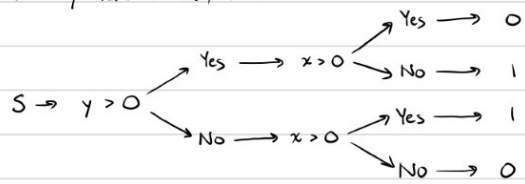
First Split Impurity

$L(S_{Top}) = 2\left(1-(1/2)^2-(1/2)^2\right) = 2(1/2) = 1$

$L(S_{Bottom}) = 2\left(1-(1/2)^2-(1/2)^2\right) = 2(1/2) = 1$

$1 + 1 = 2 = L(S')$ ⟶ No Reduction So Stop

ii. With y = vertical axis / x = horizontal axis:



The impurity measure $L(S') = |S'|^2\left(1-p_{S'}^2-(1-p_{S'})^2\right)$ would produce the above tree with the same stopping condition. It is similar to the Gini Index except the $|S'|$ is squared. By associating more points per leaf node with more error, this makes our model favor splitting more and will yield more splits in our tree compared to most other impurity measures like the Gini Index. It makes our tree continously split as long as the error doesn't increase substantially and there are more points and features to split by. The pros are that our model will be able to classify cases like this, and it will split the dataset into more nodes if that's what's desired. The cons are that our model is more likely to overfit, it will make our model more complex in most cases, it'll take longer to build our tree, and our model is more likely to jump to conclusions in matching features.

iii. The largest number of unique thresholds needed to reach 0 classification error for 100 data points is 99. This is because in the extreme largest case, we need one unique split to separate each data point from the rest of the points. Given 100 points, we'd need one split to separate the first point from the data, then given the other 99 points, we'd need a second unique split to separate the second point from the data, then given the other 98 points .... This would then continue till we've split the 99th point and there's only one point in our "other — points" set. Clearly, this shows we'd need 99 unique splits and that similarly, in general, N-1 unique splits are needed to yield 0 classification error on N data points.

Problem D [4 points]: Suppose in top-down greedy induction we want to split a leaf node that contains N data points composed of D continuous features. What is the worst-case complexity (big-O in terms of N and D) of the number of possible splits we must consider in order to find the one that most reduces impurity? Please justify your answer.

Note: Recall that at each node-splitting step in training a DT, you must consider all possible splits that you can make. While there are an infinite number of possible decision boundaries since we are using continuous features, there are not an infinite number of boundaries that result in unique child sets (which is what we mean by "split").

Solution D: The worst case complexity is O(D * N). We learned in lecture that training takes this long because we can make a split for every feature D which occurs for every data point N. This yields D * N possible splits, and in the worst case, we'd need to go through all of them. Finally, evaluating each split, i.e calculating and comparing impurity and making the split, can be considered a constant time operation, yielding our O(D * N) complexity.

2 Overfitting Decision Trees [30 Points, EC 7 Points]

Relevant materials: Lecture 5

In this problem, you will use the Diabetic Retinopathy Debrecen Data Set, which contains features extracted from images to determine whether or not the images contain signs of diabetic retinopathy. Additional information about this dataset can be found at the link below:

https://archive.ics.uci.edu/ml/datasets/Diabetic+Retinopathy+Debrecen+Data+Set

In the following question, your goal is to predict the diagnosis of diabetic retinopathy, which is the final column in the data matrix. Use the first 900 rows as training data, and the last 251 rows as validation data. Please feel free to use additional packages such as Scikit-Learn. Include your code in your submission.

Problem A [10 points]: Choose one of the following from i or ii:

i. Train a decision tree classifier using Gini as the impurity measure and minimal leaf node size as early stopping criterion. Try different minimal leaf node sizes from 1 to 25 in increments of 1. Then, on a single plot, plot both training and test classification error versus leaf node size. To do this, fill in the classification_err and eval_tree_based_model_min_samples functions in the code template for this problem.

ii. Train a decision tree classifier using Gini as the impurity measure and maximal tree depth as early stopping criterion. Try different tree depths from 2 to 20 in increments of 1. Then, on a single plot, plot both training and test classification error versus tree depth. To do this, fill in the eval_tree_based_model_-max_depth function in the code template for this problem.

Solution A: I chose to do 2A(i). My completion of 2A(ii) is for extra credit problems 2F and 2G.
https://colab.research.google.com/drive/1FMTX3NJgIgDjkgmeG9-SU4D_Fp1zy7L7?usp=sharing
Graphs below

Problem B [6 points]: For either the minimal leaf node size or maximum depth parameters in the previous problem, which parameter value minimizes the test error? What effects does early stopping have on the performance of a decision tree model? Please justify your answer based on the plot you derived.

Solution B: For the decision tree in 2A(i), the decision tree test error minimized when min_samples_leaf was set to 12. We can see this in our plot as our testing classification error dips and reaches its lowest point of approximately 0.34 at min_samples_leaf = 12. Stopping early effects a better decision tree model by limiting overfitting. As we increase the minimum node size up to 25, our training error increases, yet our test error does best at 12. This is because before a minimum node size of 12, our model is way overfitting our data. With lower to no minimum node size, our model becomes overly complex and poorly generalizes new data, so although our training error is low, our testing error shows much error. By stopping our model's training early at minimum node size of 12, we're able to stop the overfitting and yield lower test error. Then, after we reach a minimum node size of 12, any node size after that is now underfitting our data as both test error and training error increase. This shows that stopping earlier than a minimum node size of 12 would underfit our data, so while we do want to stop early to prevent overfitting, we must be careful not to stop too early and underfit.

Problem C [4 points]: Choose one of the following from i or ii:

i. Train a random forest classifier using Gini as the impurity measure, minimal leaf node size as early stopping criterion, and 1,000 trees in the forest. Try different node sizes from 1 to 25 in increments of 1. Then, on a single plot, plot both training and test classification error versus leaf node size.

ii. Train a random forest classifier using Gini as the impurity measure, maximal tree depth as early stopping criterion, and 1,000 trees in the forest. Try different tree depths from 2 to 20 in increments of 1. Then, on a single plot, plot both training and test classification error versus tree depth.

Solution C: Same Colab link as above / Did 2C(i) for this problem; 2C(ii) is extra credit.
Graphs below

Problem D [6 points]: For either the minimal leaf node size or maximum depth parameters tested, which parameter value minimizes the random forest test error? What effects does early stopping have on the performance of a random forest model? Please justify your answer based on the plot you derived.

Solution D: For the random forest in 2C(i), the random forest test error minimized when min_samples_leaf was set to 19. We can see this in our plot as our testing classification error dips and reaches its lowest point of approximately 0.32 at min_samples_leaf = 19. Stopping early effects a better random forest model by limiting overfitting. As we increase the minimum node size up to 25, our training error increases, yet our test error does best at 19. Before 19, our training error is low, yet our test error is high. This shows that overfitting is occurring in this region. By stopping early, we can limit this overfitting. In contrast, after 19, our test error gets worse and so does our training error. As both training and test error do poorly in this region, we can see that underfitting is occurring. Thus, a minimum node size of 19 produces the best performance random forest model, showing that stopping early is beneficial to limit overfitting but at the same time we can't stop too early as that would underfit our data.

Problem E [4 points]: Do you observe any differences between the curves for the random forest and decision tree plots? If so, explain what could account for these differences.

Solution E: The training error curve is flatter for the random forest compared to the decision tree training curve, and the testing curve for the random forest is flatter and a lot less variant than that of the decision tree. This can probably be explained due to the averaging and randomness of the random forest model. Because the random forest ensembles multiple decision trees, it has an averaging property that mitigates variance as a model. We can see this in how the testing error curve for the random forest is flatter and less variant that the testing error curve for the simple decision tree model; since the random forest is less variant compared to the decision tree, the decision tree model stands to benefit more from the restriction of stopping early. This is likely why the curve is more variant: because choosing the optimal point to stop early has a bigger effect on the decision tree model since it has more variance and thus is more likely to overfit the data. This is why it dips more at its optimal parameter at 12 minimum node size whereas the dip for the optimal parameter for the random forest is not as much. For the same reason, this is likely why the training error curve is flatter for the random forest model. In general, the random forest is also more accurate than the decision tree in both testing error and training error. This is probably because it performs better as an ensemble method due to its averaging and mitigated variance.

Extra Credit [7 points total] :

Problem F: [5 points, Extra Credit] Complete the other option for **Problem A** and **Problem C**.

Solution F: I chose to do 2A(ii) and 2C(ii) for extra credit.
https://colab.research.google.com/drive/1FMTX3NJgIgDjkgmeG9-SU4D_Fp1zy7L7?usp=sharing
Graphs below

Problem G: [2 points, Extra Credit] For the stopping criterion tested in **Problem F**, which parameter value minimizes the decision tree and random forest test error respectively?

Solution G:
For the decision tree in 2A(ii), the decision tree test error minimized at max_depth was set to 2. For the random forest in 2C(ii), the random forest test error minimized at max_depth was set to 18. We can see this on our plots as our test error is the lowest at these points. Similar to the reasoning for minimum node size, stopping early via maximum depth also limits overfitting. When our max_depth is before these optimal points of 2 and 18 respectively, our models underfit our data as we see low training and testing error. Then, setting our max_depth above these optimal points produces models that do a poor job of generalizing yielding overfitting as we can see low training error but high test error. This shows that stopping early increases the performance our our models by limiting overfitting but we need to be careful not to stop too early and cause underfitting.

3 The AdaBoost Algorithm [40 points]

Relevant materials: Lecture 6

In this problem, you will show that the choice of the α_t parameter in the AdaBoost algorithm corresponds to greedily minimizing an exponential upper bound on the loss term at each iteration.

Problem A [3 points]: Let $h_t : \mathbb{R}^m \to \{-1, 1\}$ be the weak classifier obtained at step t, and let α_t be its weight. Recall that the final classifier is *See GitHub*

Suppose $\{(x_1, y_1), \ldots, (x_N, y_N)\} \subset \mathbb{R}^m \times \{-1, 1\}$ is our training dataset. Show that the training set error of the final classifier can be bounded from above if an an exponential loss function is used: *See GitHub*

Solution A:

We can prove this relatively easy since we informally proved this in lecture.

Notice that $\frac{1}{N}\sum_{i=1}^{N}(\ldots i\text{ term}\ldots)$ is present on both sides of the inequality we want to prove, so we can prove the inequality by showing each i term of $\frac{1}{N}\sum_{i=1}^{N}\underbrace{e^{-y_i f(x_i)}}$ is less than or equal to each i term of $\frac{1}{N}\sum_{i=1}^{N}\underbrace{\mathbb{1}(H(x_i)\neq y_i)}$, ie we must show

$$e^{-y_i f(x_i)} \geq \mathbb{1}(H(x_i)\neq y_i)$$

Now, observe that we have two cases based on the signs of y_i and $f(x_i)$.

Case 1: y_i and $f(x_i)$ have matching signs (i.e both pos or neg / Neither can be 0 bc y_i must be 1 or -1)

Observe right side: $H(x_i) = y_i \to \mathbb{1}(H(x_i)\neq y_i) = 0$

Observe left side: $y_i f(x_i) > 0 \to -y_i f(x_i) < 0 \to e^{-y_i f(x_i)} \geq 0$

Thus, $e^{-y_i f(x_i)} \geq \mathbb{1}(H(x_i)\neq y_i) = 0$.

Case 2: y_i and $f(x_i)$ have opposite signs (i.e $f(x_i)$ is pos/neg/0 while y_i is nonmatching pos/neg)

Observe right side: $H(x_i) \neq y_i \to \mathbb{1}(H(x_i)\neq y_i) = 1$

Observe left side: $y_i f(x_i) \leq 0 \to -y_i f(x_i) \geq 0 \to e^{-y_i f(x_i)} \geq 1$

Thus, $e^{-y_i f(x_i)} \geq \mathbb{1}(H(x_i)\neq y_i) = 1$.

Therefore, we've proven $e^{-y_i f(x_i)} \geq \mathbb{1}(H(x_i)\neq y_i)$ for both cases, showing $E = \frac{1}{N}\sum_{i=1}^{N} e^{-y_i f(x_i)} \geq \frac{1}{N}\sum_{i=1}^{N}\mathbb{1}(H(x_i)\neq y_i)$.

□

Problem B [3 points]: Find $D_{T+1}(i)$ in terms of Z_t, α_t, x_i, y_i, and the classifier h_t, where T is the last timestep and $t \in \{1, \ldots, T\}$. Recall that Z_t is the normalization factor for distribution D_{t+1}: *See GitHub*

Solution B:

In lecture, we're provided the recursive definition $D_{t+1}(i) = \frac{D_t(i) e^{-\alpha_t y_i h_t(x_i)}}{Z_t}$ where $Z_t = \sum_{i=1}^{N} D_t(i) e^{-\alpha_t y_i h_t(x_i)}$.

We can use this recursive definition to get a closed form for $D_{T+1}(i)$. Observe the following...

We know $D_1(i) = 1/N$.

According to recursive definition,

$$D_2(i) = \frac{D_1(i) e^{-\alpha_1 y_i h_1(x_i)}}{Z_1} = \frac{1/N e^{-\alpha_1 y_i h_1(x_i)}}{Z_1}$$

$$D_3(i) = \frac{D_2(i) e^{-\alpha_2 y_i h_2(x_i)}}{Z_2} = \frac{1/N e^{-\alpha_1 y_i h_1(x_i)} e^{-\alpha_2 y_i h_2(x_i)}}{Z_1 Z_2} = \frac{e^{-y_i(\alpha_1 h_1(x_i) + \alpha_2 h_2(x_i))}}{N Z_1 Z_2}$$

$$D_4(i) = \frac{D_3(i) e^{-\alpha_3 y_i h_3(x_i)}}{Z_3} = \frac{e^{-y_i(\alpha_1 h_1(x_i) + \alpha_2 h_2(x_i))} e^{-\alpha_3 y_i h_3(x_i)}}{N Z_1 Z_2 Z_3} = \frac{e^{-y_i(\alpha_1 h_1(x_i) + \alpha_2 h_2(x_i) + \alpha_3 h_3(x_i))}}{N Z_1 Z_2 Z_3}$$

$$D_{T+1}(i) = \frac{\exp\left(-y_i \sum_{j=1}^{t} \alpha_j h_j(x_i)\right)}{N \prod_{j=1}^{t} Z_j} = \frac{\prod_{j=1}^{t} \exp(-y_i \alpha_j h_j(x_i))}{N \prod_{j=1}^{t} Z_j} = \frac{1}{N} \prod_{j=1}^{t} \frac{\exp(-y_i \alpha_j h_j(x_i))}{Z_j} = \frac{1}{N} \prod_{t=1}^{T} \frac{\exp(-y_i \alpha_t h_t(x_i))}{Z_t}$$

Therefore, $D_{T+1}(i) = \frac{1}{N} \prod_{t=1}^{T} \frac{\exp(-y_i \alpha_t h_t(x_i))}{Z_t}$ ← ANSWER FOR GRADING

Please, still consider my further analysis below.

Technically, we could end here, but let's see if we can go further by eliminating and further simplifying out the Z terms.

To do so, let's further analyze the denominator $N \prod_{j=1}^{T} Z_j$. So, let's take a closer look at Z_t...

$$Z_t = \sum_{i=1}^{N} \left(D_t(i) e^{-\alpha_t y_i h_t(x_i)} \right) = \sum_{i=1}^{N} \left(\frac{D_{t-1}(i) e^{-\alpha_{t-1} y_i h_{t-1}(x_i)}}{Z_{t-1}} e^{-\alpha_t y_i h_t(x_i)} \right) = \frac{1}{Z_{t-1}} \sum_{i=1}^{N} D_{t-1}(i) e^{-y_i(\alpha_{t-1} h_{t-1}(x_i) + \alpha_t h_t(x_i))}$$

$$D_t(i) = \frac{D_{t-1}(i) e^{-\alpha_{t-1} y_i h_{t-1}(x_i)}}{Z_{t-1}}$$

$$Z_t = \frac{1}{Z_{t-1}} \sum_{i=1}^{N} D_{t-1}(i) e^{-y_i(\alpha_{t-1} h_{t-1}(x_i) + \alpha_t h_t(x_i))} = \frac{1}{Z_{t-1}} \sum_{i=1}^{N} \left(\frac{D_{t-2}(i) e^{-\alpha_{t-2} y_i h_{t-2}(x_i)}}{Z_{t-2}} e^{-y_i(\alpha_{t-1} h_{t-1}(x_i) + \alpha_t h_t(x_i))} \right) = \frac{1}{Z_{t-1} Z_{t-2}} \sum_{i=1}^{N} \left(D_{t-2}(i) e^{-y_i(\alpha_{t-2} h_{t-2}(x_i) + \alpha_{t-1} h_{t-1}(x_i) + \alpha_t h_t(x_i))} \right)$$

$$D_{t-1}(i) = \frac{D_{t-2}(i) e^{-\alpha_{t-2} y_i h_{t-2}(x_i)}}{Z_{t-2}}$$

$$Z_t = \frac{1}{Z_{t-1} Z_{t-2}} \sum_{i=1}^{N} \left(D_{t-2}(i) e^{-y_i(\alpha_{t-2} h_{t-2}(x_i) + \alpha_{t-1} h_{t-1}(x_i) + \alpha_t h_t(x_i))} \right)$$

$$Z_t = \frac{1}{\prod_{j=1}^{t-1} Z_j} \sum_{i=1}^{N} D_i(i) e^{-y_i\left(\sum_{j=1}^{t} \alpha_j h_j(x_i)\right)} = \frac{1}{\prod_{j=1}^{t-1} Z_j} \sum_{i=1}^{N} \frac{1}{N} e^{-y_i\left(\sum_{j=1}^{t} \alpha_j h_j(x_i)\right)} = \frac{1}{N \prod_{j=1}^{t-1} Z_j} \sum_{i=1}^{N} \exp\left(-y_i\left(\sum_{j=1}^{t} \alpha_j h_j(x_i)\right)\right) = \frac{\sum_{i=1}^{N} \prod_{j=1}^{t} \exp(-y_i \alpha_j h_j(x_i))}{N \prod_{j=1}^{t-1} Z_j}$$

So $N \prod_{j=1}^{t} Z_j = N \prod_{j=1}^{t-1} Z_j * Z_T = N \prod_{j=1}^{t-1} Z_j * \frac{1}{N \prod_{j=1}^{t-1} Z_j} \sum_{i=1}^{N} \prod_{j=1}^{t} \exp(-y_i \alpha_j h_j(x_i)) = \sum_{i=1}^{N} \prod_{j=1}^{t} \exp(-y_i \alpha_j h_j(x_i))$

Thus, $\frac{x_i}{\sum_{i=1}^{n} x_i}$

$$D_{T+1}(i) = \frac{\prod_{j=1}^{t} \exp(-y_i \alpha_j h_j(x_i))}{N \prod_{j=1}^{t} Z_j} = \frac{\prod_{j=1}^{t} \exp(-y_i \alpha_j h_j(x_i))}{\sum_{i=1}^{N} \prod_{j=1}^{t} \exp(-y_i \alpha_j h_j(x_i))} = \frac{1}{\sum_{k=1, k \neq i}^{N} \prod_{j=1}^{t} \exp(-y_k \alpha_j h_j(x_k))} = \sum_{k=1, k \neq i}^{N} \prod_{j=1}^{t} \exp(y_k \alpha_j h_j(x_k))$$

Further, $D_{T+1}(i) = \sum_{k=1, k \neq i}^{N} \prod_{t=1}^{T} \exp(y_k \alpha_t h_t(x_k))$

↑ where the summation has k equal every integer in the set $\{1, \ldots, N\} / \{i\}$, thus skipping the value of i.

Please, refer to the above form for grading, but please let me know if my further analysis here is correct.

So, we were able to further simplify $D_{T+1}(i)$'s closed form solution by eliminating the Z terms. □

Problem C [2 points]: Show that *See GitHub*

Solution C:

To prove $E = \sum_{i=1}^{N} \frac{1}{N} \exp\left(\sum_{t=1}^{T} -\alpha_t y_i h_t(x_i)\right)$, we start with $E = \frac{1}{N} \sum_{i=1}^{N} \exp(-y_i f(x_i))$ from part a.

Now, we know from lecture, $f(x_i) = \sum_{t=1}^{T} \alpha_t h_t(x_i)$, so we can plug this in as follows

$$E = \frac{1}{N} \sum_{i=1}^{N} \exp\left(-y_i \sum_{t=1}^{T} \alpha_t h_t(x_i)\right)$$

Now, with a little reorganizing obeying the rules of summations, we can move $\frac{1}{N}$ into summation $\sum_{i=1}^{N}$ and $-y_i$ into summation $\sum_{t=1}^{T}$ to obtain

$$E = \sum_{i=1}^{N} \frac{1}{N} \exp\left(\sum_{t=1}^{T} -\alpha_t y_i h_t(x_i)\right).$$

□

Problem D [5 points]: Show that *See GitHub*

Solution D:

To do this, we can combine our results from parts b and c and what we know from lecture.

Observe $D_{T+1}(i) = \frac{1}{N} \prod_{t=1}^{T} \frac{\exp(-y_i \alpha_t h_t(x_i))}{Z_t} = \frac{\frac{1}{N} \prod_{t=1}^{T} \exp(-y_i \alpha_t h_t(x_i))}{\prod_{t=1}^{T} Z_t} = \frac{\frac{1}{N} \exp\left(\sum_{t=1}^{T} -\alpha_t y_i h_t(x_i)\right)}{\prod_{t=1}^{T} Z_t}$

Note that $D_{T+1}(i)$ has E part of E inside it, so we can solve for that via the following

$$D_{T+1}(i) = \frac{\frac{1}{N} \exp\left(\sum_{t=1}^{T} -\alpha_t y_i h_t(x_i)\right)}{\prod_{t=1}^{T} Z_t} \longrightarrow \frac{1}{N} \exp\left(\sum_{t=1}^{T} -\alpha_t y_i h_t(x_i)\right) = \prod_{t=1}^{T} Z_t \, D_{T+1}(i)$$

Now, we can substitute this into $E = \sum_{i=1}^{N} \frac{1}{N} \exp\left(\sum_{t=1}^{T} -\alpha_t y_i h_t(x_i)\right)$...

$$E = \sum_{i=1}^{N} \left(\prod_{t=1}^{T} Z_t \, D_{T+1}(i)\right)$$

And, we can pull $\prod_{t=1}^{T} Z_t$ out of the summation since is independent of i.

$$E = \prod_{t=1}^{T} Z_t * \sum_{i=1}^{N} D_{T+1}(i)$$

Then, we know any distribution $D(i)$ summed from 1 to N is 1, so $\sum_{i=1}^{N} D_{T+1}(i) = 1$.

Therefore, we have $E = \prod_{t=1}^{T} Z_t$.

□

Problem E [5 points]: Show that the normalizer Z_t can be written as *See GitHub*
where $_t$ is the training set error of weak classifier h_t for the weighted dataset: *See GitHub*

Solution E:

To do this, we can analyze the values of y_i and $h_t(x_i)$ starting from our result from b, $Z_t = \sum_{i=1}^{N} D_t(i)\exp(-\alpha_t y_i h_t(x_i))$.

Because y_i is our classification label and $h_t(x_i)$ is a prediction for our classification label, they are both either 1 or -1.

Thus, similar to part a, we can analyze the two cases.

If y_i and $h(x_i)$ have matching signs (when $\mathbb{1}(h_t(x_i) \neq y_i) = 0$): $y_i h_t(x_i) = 1 \rightarrow \exp(-\alpha_t y_i h_t(x_i)) = \exp(-\alpha_t)$

If y_i and $h_t(x_i)$ have opposite signs (when $\mathbb{1}(h_t(x_i) \neq y_i) = 1$): $y_i h_t(x_i) = -1 \rightarrow \exp(-\alpha_t y_i h_t(x_i)) = \exp(\alpha_t)$

We can then express $\exp(-\alpha_t y_i h_t(x_i))$ in terms of $\exp(-\alpha_t)$ and $\exp(\alpha_t)$ depending on whether $\mathbb{1}(h_t(x_i) \neq y_i)$ outputs 1 or 0.

$$\exp(-\alpha_t y_i h_t(x_i)) = (1 - \mathbb{1}(h_t(x_i) \neq y_i))\, e^{-\alpha_t} + \mathbb{1}(h_t(x_i) \neq y_i) e^{\alpha_t}$$

Now, we can plug this in and simplify. Observe the following manipulation.

$$Z_t = \sum_{i=1}^{N} \left(D_t(i)(1 - \mathbb{1}(h_t(x_i) \neq y_i))\, e^{-\alpha_t} + \mathbb{1}(h_t(x_i) \neq y_i) e^{\alpha_t} \right)$$

$$Z_t = e^{-\alpha_t} \sum_{i=1}^{N} \left(D_t(i) - D_t(i)\mathbb{1}(h_t(x_i) \neq y_i) \right) + e^{\alpha_t} \sum_{i=1}^{N} D_t(i)\mathbb{1}(h_t(x_i) \neq y_i)$$

$$Z_t = e^{-\alpha_t} \left[\sum_{i=1}^{N} D_t(i) - \sum_{i=1}^{N} D_t(i)\mathbb{1}(h_t(x_i) \neq y_i) \right] + e^{\alpha_t} \sum_{i=1}^{N} D_t(i)\mathbb{1}(h_t(x_i) \neq y_i)$$

Take note that $\sum_{i=1}^{N} D_t(i) = 1$ and $\sum_{i=1}^{N} D_t(i)\mathbb{1}(h_t(x_i) \neq y_i) = \epsilon_t$. Thus, we have our desired result

$$Z_t = (1 - \epsilon_t)e^{-\alpha_t} + \epsilon_t e^{\alpha_t}$$

□

Problem F [2 points]: We derived all of this because it is hard to directly minimize the training set error, but we can greedily minimize the upper bound E on this error. Show that choosing α_t greedily to minimize Z_t at each iteration leads to the choices in AdaBoost: *See GitHub*

Solution F:

To minimize Z_t using α_t, we must find α_t^* that satisfies $\frac{\partial Z_t}{\partial \alpha_t} = 0$. We can do this because we know Z_t has a global minimum where its gradient equals 0 based on lecture.

$$0 = \frac{\partial Z_t}{\partial \alpha_t} = \frac{\partial}{\partial \alpha_t}\left((1 - \epsilon_t)e^{-\alpha_t} + \epsilon_t e^{\alpha_t} \right) = (\epsilon_t - 1)e^{-\alpha_t} + \epsilon_t e^{\alpha_t}$$

$$0 = (\epsilon_t - 1)e^{-\alpha_t} + \epsilon_t e^{\alpha_t} = e^{-\alpha_t}(\epsilon - 1 + \epsilon_t e^{2\alpha_t})$$

$$0 = \epsilon_t - 1 + \epsilon_t e^{2\alpha_t} \rightarrow e^{2\alpha_t} = \frac{1 - \epsilon_t}{\epsilon_t} \rightarrow \alpha_t = \frac{1}{2}\ln\left(\frac{1 - \epsilon_t}{\epsilon_t}\right)$$

$$\alpha_t^* = \frac{1}{2}\ln\left(\frac{1 - \epsilon_t}{\epsilon_t}\right)$$

Problem G [14 points]: Implement the GradientBoosting.fit() and AdaBoost.fit() methods in the notebook provided for you. Some important notes and guidelines follow:

- For both methods, make sure to work with the class attributes provided to you. Namely, after GradientBoosting.fit() is called, self.clfs should be appropriately filled with the self.n_- clfs trained weak hypotheses. Similarly, after AdaBoost.fit() is called, self.clfs and self.coefs should be appropriately filled with the self.n_clfs trained weak hypotheses and their coefficients, respectively.

- AdaBoost.fit() should additionally return an (N, T) shaped numpy array D such that D[:, t] contains D_{t+1} for each $t \in \{0, \ldots,$ self.n_clfs$\}$.

- For the AdaBoost.fit() method, **use the 0/1 loss** instead of the exponential loss.

- The only Sklearn classes that you may use in implementing your boosting fit functions are the DecisionTreeRegressor and DecisionTreeClassifier, not GradientBoostingRegressor.

Solution G:
https://colab.research.google.com/drive/1jlmBv7gwNEv31c7uMzVzttOJCap7LZXW?usp=sharing
Graphs below

Problem H [2 points]: Describe and explain the behaviour of the loss curves for gradient boosting and for AdaBoost. You should consider two kinds of behaviours: the smoothness of the curves and the final values that the curves approach.

Solution H: The loss curves for Gradient Boosting are much smoother and flatter while the loss curves for AdaBoost are more rigid with noisy fluctuations and dips. The Gradient Boost model performs optimally somewhere around 50 weak regressors and then the test error slowly increases as we add more weak regressors. It seems to approach a value somewhere around 0.3 loss for the test error while its training error approaches 0. This indicates overfitting, which is as expected since Gradient Boosting is not optimized for classification scenarios. AdaBoost, on the other hand, has both its test and training error get lower and lower approaching 0 as we add more weak classifiers. Instead of reaching a point where more weak classifiers cause overfitting, AdaBoost continues to perform better. This is because of the nature of AdaBoost: more classifiers compound to improve predictions due to its weighting of the models since it is optimized for classification whereas in Gradient Boosting each of its weak regressors don't have weights and end up muddling the prediction accuracy. In this way, AdaBoost is able to prevent overfitting and continue to get more accurate approaching both 0 for training and test error. Lastly, the fluctuations for AdaBoost may be explained due to the increased accuracy with each new weak classifier. At certain numbers of weak classifiers, the training and test error for AdaBoost dips since the model benefits a lot from those added models. This may be because the weighting part of the algorithm is able to better balance and weight the different classifiers to yield better performance. Due to the weighting of Adaboost, there may be certain optimal numbers of classifiers that decrease error, giving its error curves the noisy fluctuations we see. In contrast, the Gradient Boosting model may not benefit that much from each added weak regressor since the regressors here do not complement each other as well as the classifiers do in AdaBoost. This makes it so that the curves are much flatter and that the testing error increases after a certain number.

Problem I [2 points]: Compare the final loss values of the two models. Which performed better on the classification dataset?

Solution I: AdaBoost performed better with a testing error of 0.186 despite having a higher training error of 0.1045 whereas the Gradient Boosting model had a testing error of 0.264, but a training error of 0. This is to be expected though as AdaBoost optimizes boosting for classification problems while Gradient Boosting could do better as an algorithm. This shows that overfitting may have occurred during our Gradient Boosting.

Problem J [2 points]: For AdaBoost, where are the dataset weights the largest, and where are they the smallest?

Hint: *Watch how the dataset weights change across time in the animation.*

Solution J: The weights are largest at the misclassified points around the boundary between the positively and negatively labeled points mostly near the center, and the weights are smallest for the correctly classified points away from that boundary and especially towards the far ends where they are farthest from the boundary. We can see this by examining the size and location of the weights on the animation. This makes sense so that we can correctly predict and compensate for predictions that would be otherwise low and easily flipped.

4 Convex Functions [7 points, EC 3 Points]

This problem further develops the ideas of convex functions, and provides intuition for why convex optimization is so important for Machine Learning.

Given a convex set X , a function $f : X \to \mathbb{R}$ is **convex** if for all $\mathbf{x}, \mathbf{y} \in X$ and all $t \in [0, 1] : f(t\mathbf{x} + (1 - t)\mathbf{y}) \leq tf(\mathbf{x}) + (1 - t)f(\mathbf{y})$

Problem A [3 points]: Let X be a convex set. If f is a convex function, show that any local minimum of f in X is also a global minimum.

Solution A:

Let us assume to the contrary that given our convex set $\mathcal{X}$ and convex function f, we have some point m that is a local minimum of f in $\mathcal{X}$ but not a global minimum such that there exists another point h somewhere in $\mathcal{X}$ where $f(h) < f(m)$.

Since both m and h are in $\mathcal{X}$, we have

$$f(tm + (1-t)h) \leq tf(m) + (1-t)f(h) \quad \forall t \in [0,1].$$

Since $f(m) > f(h)$, we can substitute and have our equality be strictly less than as follows

$$f(tm + (1-t)h) < tf(m) + (1-t)f(m) \quad \forall t \in [0,1].$$

Now, observe $tf(m) + (1-t)f(m) = (t+1-t)f(m) = f(m)$, showing that $tf(m) + (1-t)f(m) = f(m)$ no matter what t equals.

Thus, we have with strictly less than

$$f(tm + (1-t)h) < f(m) \quad \forall t \in [0,1].$$

Now, note what happens when we confirm the endpoints of $t \in [0,1]$.

$t=0 \rightarrow f(0*m + 1*h) = f(h) < f(m)$. This is correct and how we arrived at our strict inequality.

$t=1 \rightarrow f(1*m + 0*h) = f(m) < f(m)$. We have arrived at a contradiction!

The endpoint $t=1$ yielding $f(m) < f(m)$ shows that our construction of $f(h) < f(m)$ was incorrect such that m must also be a global minimum. □

Problem B [4 points]: *Using part A*, explain why convex loss functions are desirable when training learning models.

Solution B: Part A proves that any local minimum of f in X is also a global minimum in X. In training learning models, we often construct mathematical functions to model our data for which we will have optimal performance when our loss computed via a loss function is at its global minimum. Thus, our goal is to find the global minimum of our loss function so that our model best predicts our data, but the process of doing so is often time consuming, computationally expensive, and requires a lot of data. For convex functions however, finding a local minimum means we've found the global minimum and thus have achieved our optimal model. This makes the analysis of finding the global minimum easier as we can simply use first-order gradients to follow our convex loss functions to its local minimum which is also its global minimum.
To see why this is so desirable, let's consider non-convex loss functions. If we weren't using a convex loss function such that there are multiple local minima, then using first-order gradient analysis wouldn't be enough; the effort to find a local minima via first-order analysis yields a local minima that

isn't guaranteed to be a global minima, so we still don't know if a better performing model exists. Thus, we'd have to use other, more computationally expensive analysis to find or confirm our global minima since first-order analysis isn't guaranteed to yield our optimal model. This is not ideal and would be troublesome to work with.
Use of first-order analysis to find global minima is most efficient compared to other methods and becomes possible with convex loss functions, which is why they are desired when training learning models. Having only one minima makes it so we can use relatively simple analysis to find our optimal model without having to worry about non global minima like in the case of non convex functions.

Problem C : [3 points, Extra Credit] The Kullback-Leibler (KL) divergence is a measure of statistical distance between two probability distributions (*p, q*), also called the relative entropy. KL divergence can be used to generate optimal parameters for visualization models (which we will also see in set 4).

See GitHub

Show that the KL divergence is a convex loss function.

Hint: *Use the log sum inequality*

Solution C:

We will prove $KL[P||Q] = \sum_{x \in \mathcal{X}} p(x) \cdot \log \frac{p(x)}{q(x)}$ is a convex function. To do so, let's define two points $w = (p_w, q_w)$ and $z = (p_z, q_z)$ to be our two inputs to KL divergence.

To prove KL divergence is a convex function, we must show

$$KL[tp_w + (1-t)p_z \,||\, tq_w + (1-t)q_z] \leq tKL[p_w||q_w] + (1-t)KL[p_z||q_z] \quad \forall t \in [0,1].$$

Knowing this, we can start our manipulation here, plugging our inputs into KL divergence.

$$KL[tp_w + (1-t)p_z \,||\, tq_w + (1-t)q_z] = \sum_{x \in \mathcal{X}} \left[(tp_w + (1-t)p_z) \cdot \log \frac{tp_w + (1-t)p_z}{tq_w + (1-t)q_z} \right] \quad \forall t \in [0,1]$$

Now, consider that the log sum inequality states that for $\forall a_i, b_i \geq 0, i \in [1,n]$ with $a = \sum_{i=1}^{n} a_i$ and $b = \sum_{i=1}^{n} b_i$,

$$\sum_{i=1}^{n} a_i \log \frac{a_i}{b_i} \geq a \log \frac{a}{b}.$$

In $(tp_w + (1-t)p_z) \cdot \log \frac{tp_w + (1-t)p_z}{tq_w + (1-t)q_z}$, both $t, (1-t) \geq 0$ such that we can consider $a = tp_w + (1-t)p_z$ and $b = tq_w + (1-t)q_z$ in a simple $n=2$ term case to write

$$(tp_w + (1-t)p_z) \cdot \log \frac{tp_w + (1-t)p_z}{tq_w + (1-t)q_z} \leq tp_w \cdot \log \frac{tp_w}{tq_w} + (1-t)p_z \cdot \log \frac{(1-t)p_z}{(1-t)q_z} \quad \forall t \in [0,1].$$

Thus, we can plug that back into our summation where we have

$$\sum_{x \in \mathcal{X}} \left[(tp_w + (1-t)p_z) \cdot \log \frac{tp_w + (1-t)p_z}{tq_w + (1-t)q_z} \right] \leq \sum_{x \in \mathcal{X}} \left[tp_w \cdot \log \frac{tp_w}{tq_w} + (1-t)p_z \cdot \log \frac{(1-t)p_z}{(1-t)q_z} \right] \quad \forall t \in [0,1].$$

Then, we can factor, simplify, and condense.

$$\sum_{x \in \mathcal{X}} \left[tp_w \cdot \log \frac{tp_w}{tq_w} + (1-t)p_z \cdot \log \frac{(1-t)p_z}{(1-t)q_z} \right] = t \sum_{x \in \mathcal{X}} p_w \cdot \log \frac{p_w}{q_w} + (1-t) \sum_{x \in \mathcal{X}} p_z \cdot \log \frac{p_z}{q_z} = tKL[p_w||q_w] + (1-t)KL[p_z||q_z] \quad \forall t \in [0,1]$$

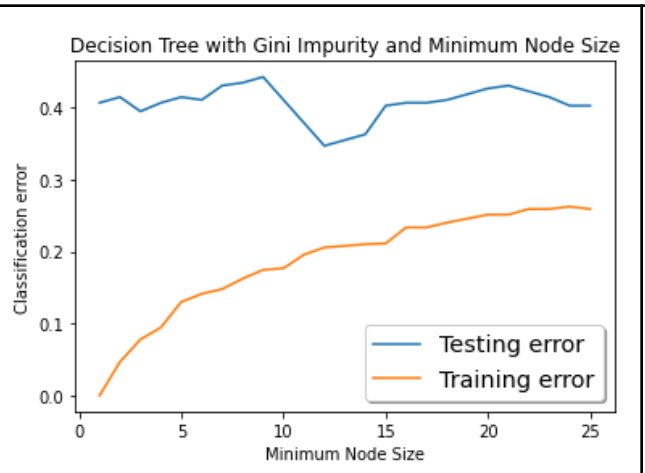
Therefore, we arrive at the following, proving KL divergence is a convex loss function.

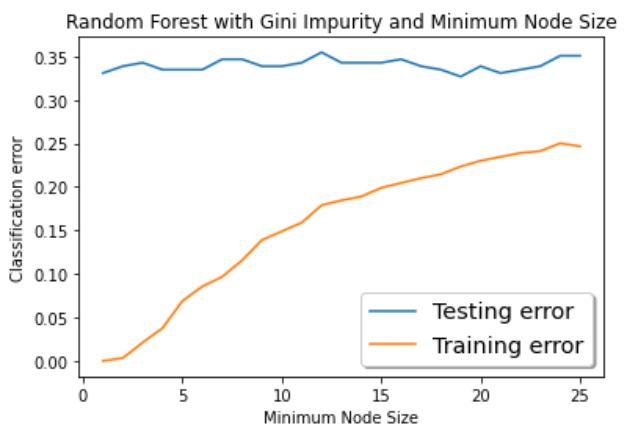
$$KL[tp_w + (1-t)p_z \,||\, tq_w + (1-t)q_z] \leq tKL[p_w||q_w] + (1-t)KL[p_z||q_z] \quad \forall t \in [0,1]. \qquad \square$$

Generated Images

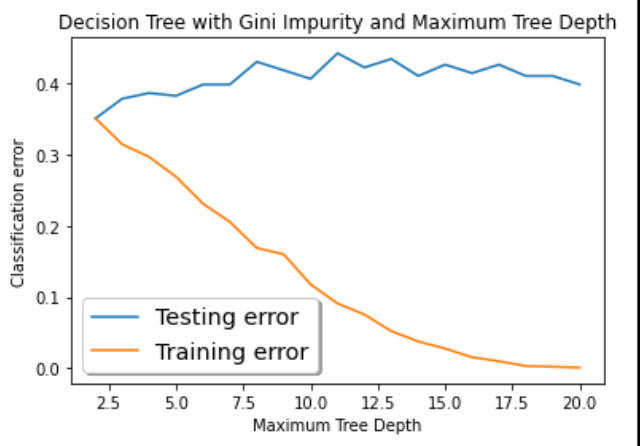
Part 2

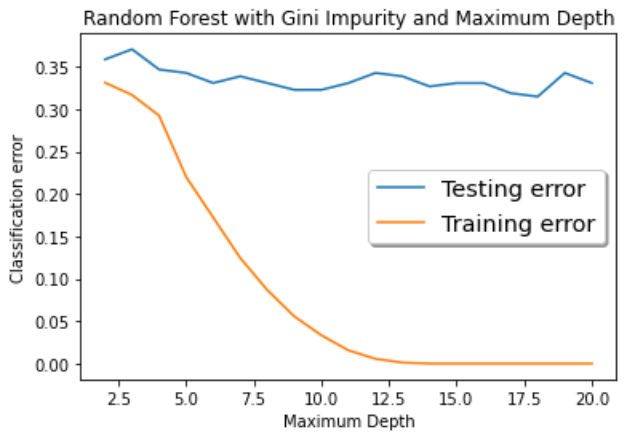
Credit for Problems 2A and 2C





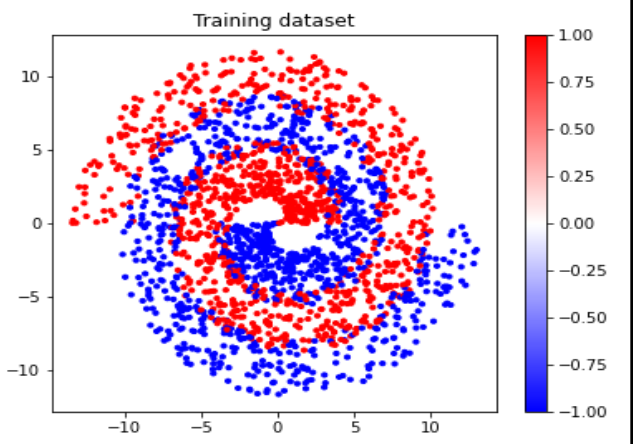
Extra Credit for Problem 2F

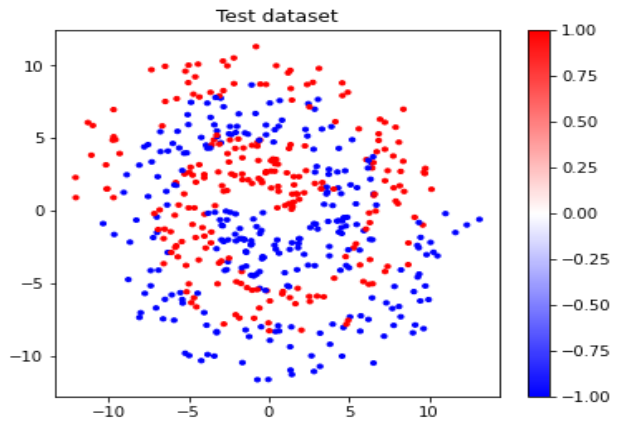




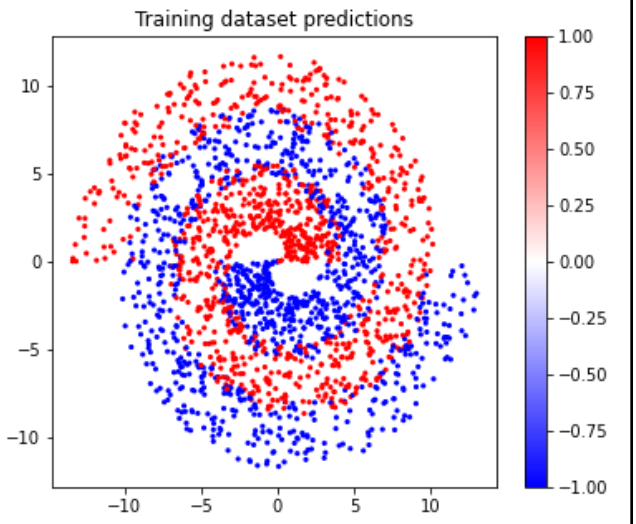
Part 3

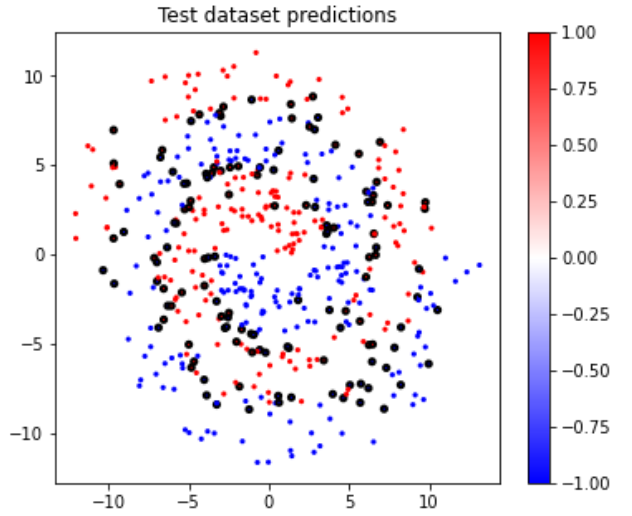
Datasets

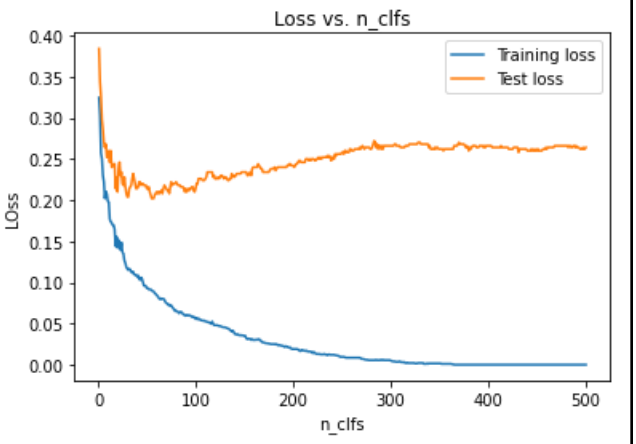




Gradient Boosting

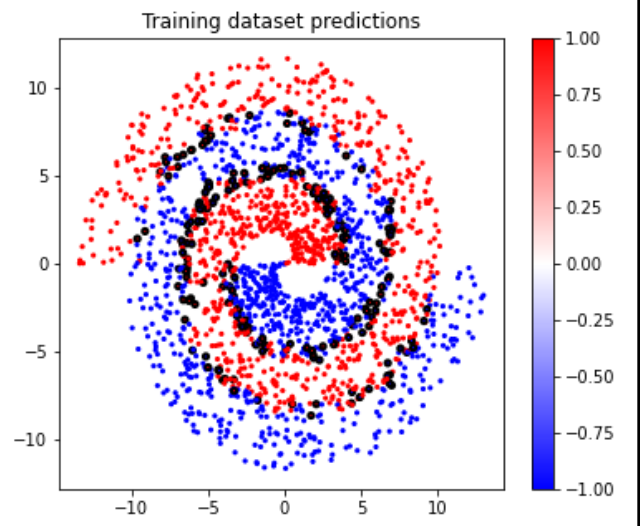


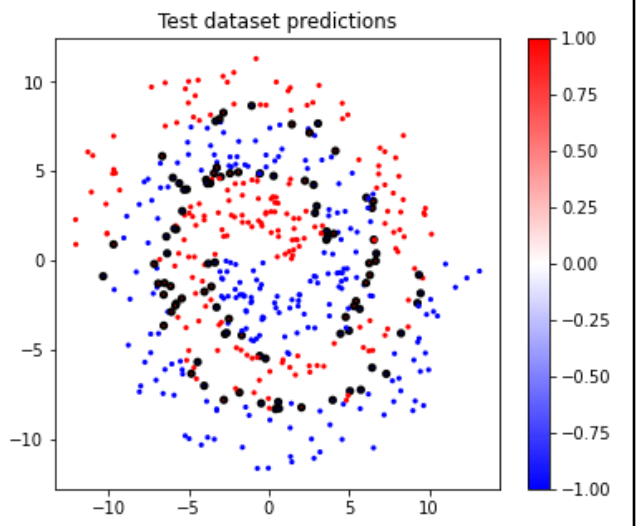


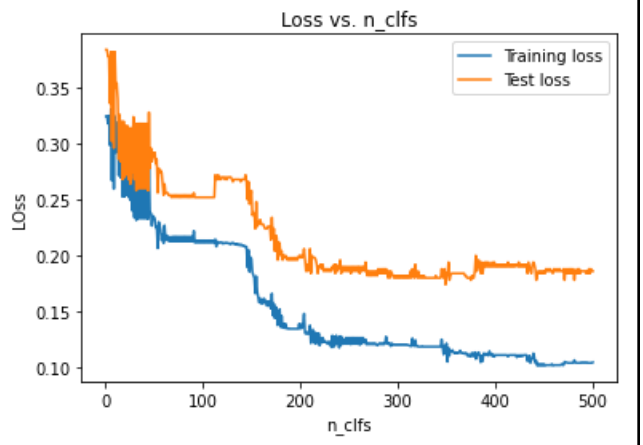


Animation in Notebook

AdaBoost







Animation in Notebook