Machine Learning & Data Mining
Set 3

Caltech CS/CNS/EE 155
January 19th, 2024

Policies

- Due 11:59 PM, January 26th, via Gradescope.
- 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.
- You are allowed to use up to 48 late hours across the entire term. Late hours must be used in units of whole hours. Specify the total number of hours you have used when submitting the assignment.
- **No use of large language models is allowed.** Students are expected to complete homework assignments based on their understanding of the course material.

Submission Instructions

- Submit your report as a single .pdf file to Gradescope (entry code 2P8P28), under "Problem Set 3".
- 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_prob1.ipynb"

1 Decision Trees [30 Points]

Relevant materials: Lecture 5

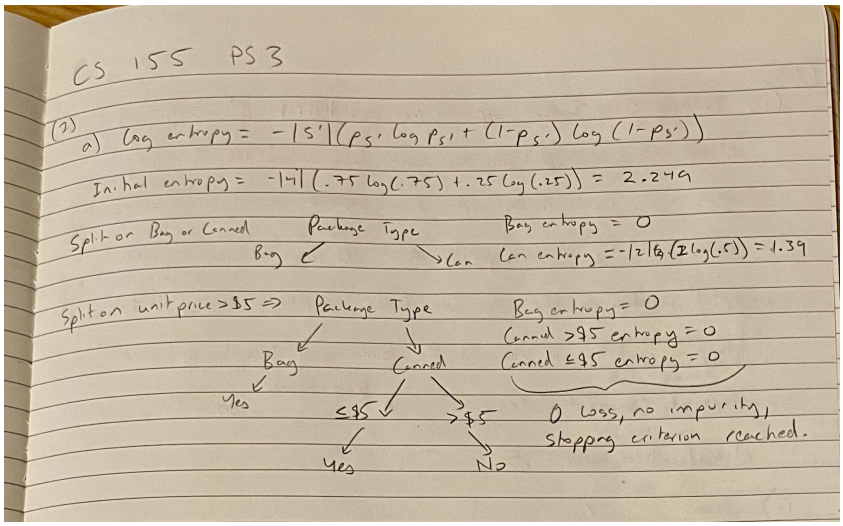
Problem A [7 points]: Consider the following data, where given information about some food you must predict whether it is healthy:

No.	Package Type	Unit Price $>$ \$5	Contains $>$ 5 grams of fat	Healthy?
1	Canned	Yes	Yes	No
2	Bagged	Yes	No	Yes
3	Bagged	No	Yes	Yes
4	Canned	No	No	Yes

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.

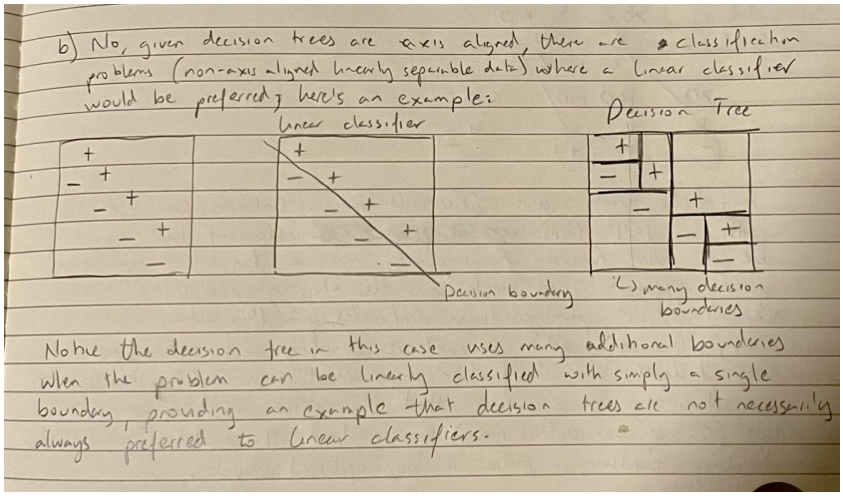
Submit a drawing of your tree showing the impurity reduction yielded by each split (including root) in your decision tree.

Solution A:

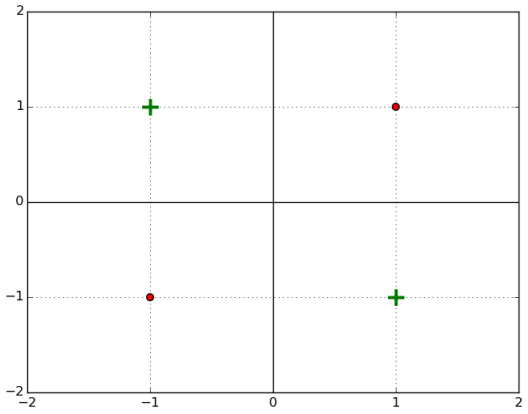


Problem B [4 points]: Compared to a linear classifier, is a decision tree always preferred for classification problems? If not, draw a simple 2-D dataset that can be perfectly classified by a simple linear classifier but which requires an overly complex decision tree to perfectly classify.

Solution B:



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))?

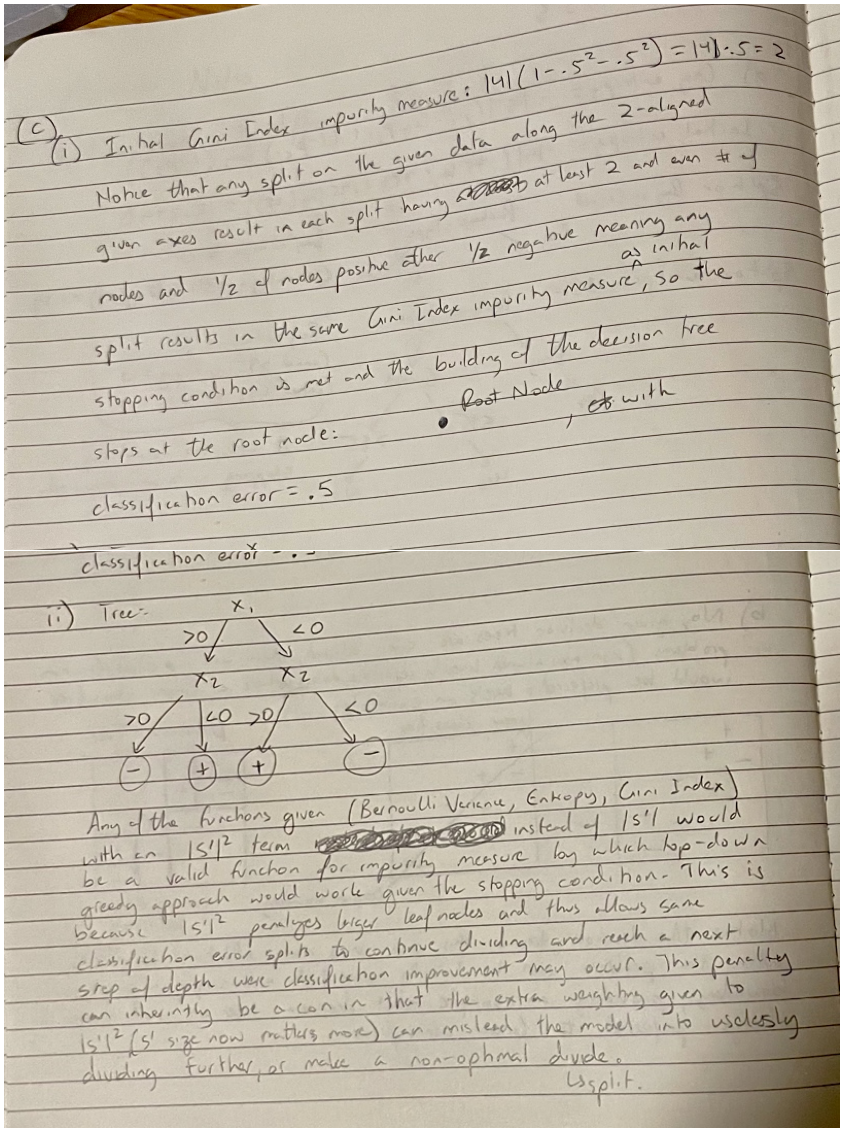
ii. [5 points]: Submit a drawing of a two-level decision tree that classifies the above dataset with zero classification error. (You don't need to use any particular training algorithm to produce the tree.)

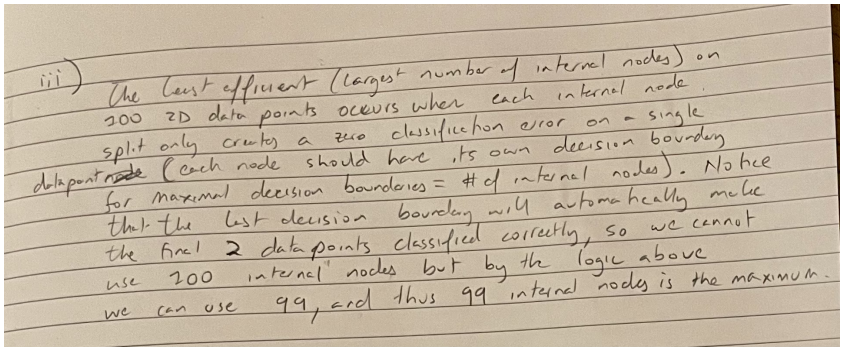
Is there any impurity measure (i.e. any function that maps the data points under a particular node in a

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:

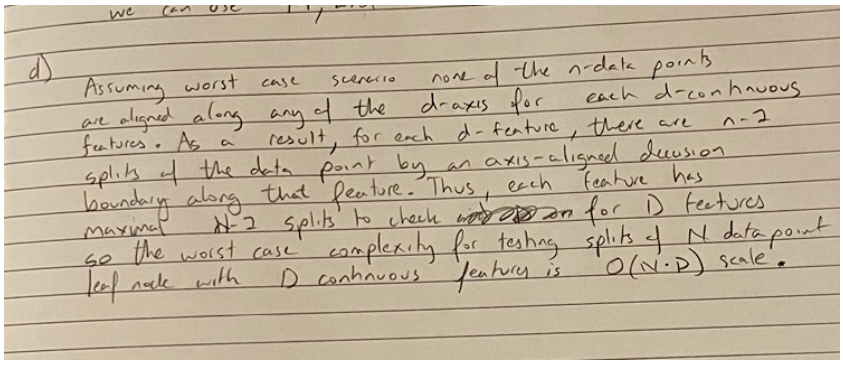




Problem D [4 points]: Suppose in top-down greedy induction we want to split a leaf node that contains N data points composed of D continuous features. What is the worst-case complexity (big-O in terms of N and D) of the number of possible splits we must consider in order to find the one that most reduces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



2 Overfitting Decision Trees [30 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:

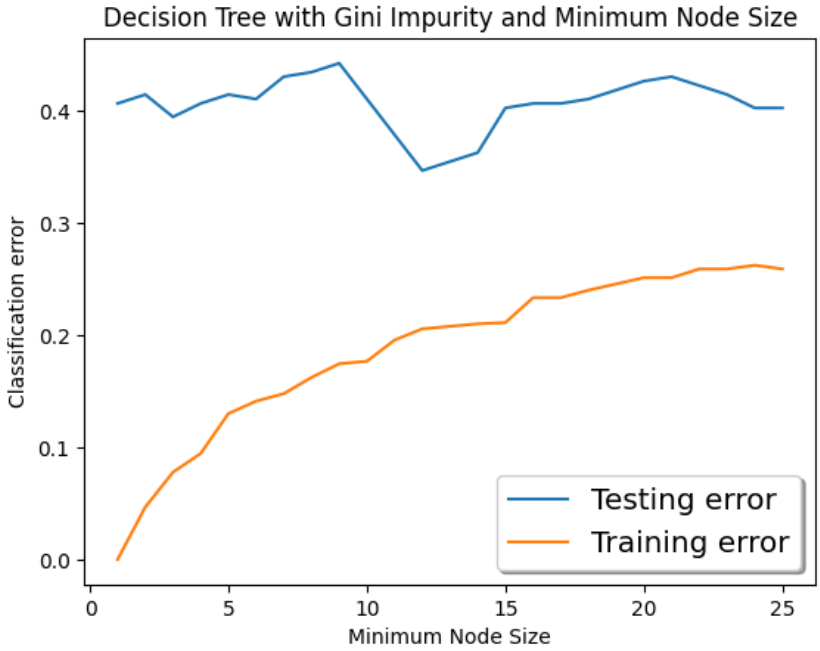
https://archive.ics.uci.edu/ml/datasets/Diabetic+Retinopathy+Debrecen+Data+Set

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 [7 points]: 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.

Solution A:

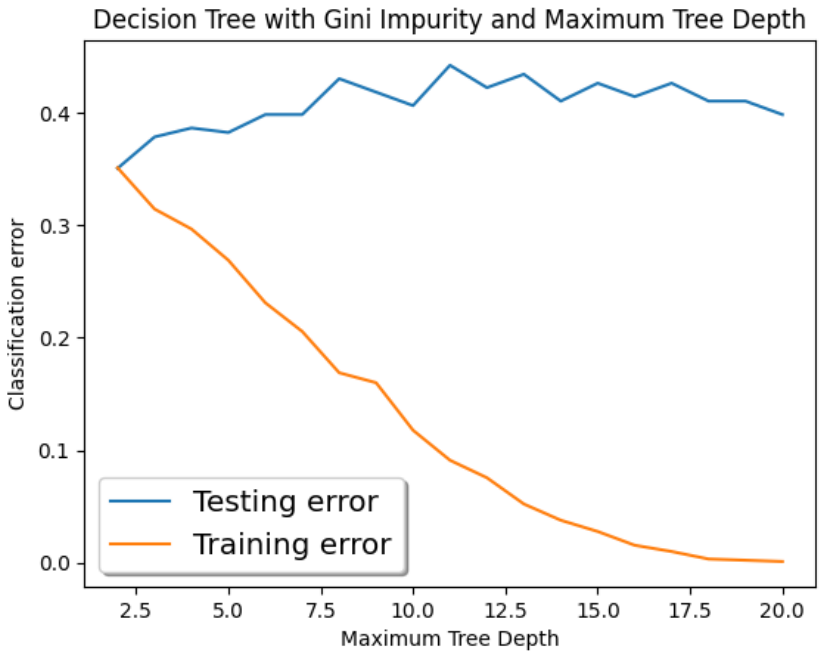
solution code



Problem B [7 points]: 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 B:

solution code



Problem C [4 points]: For both the minimal leaf node size and maximum depth parameters tested in the last two questions, 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 two plots you derived.

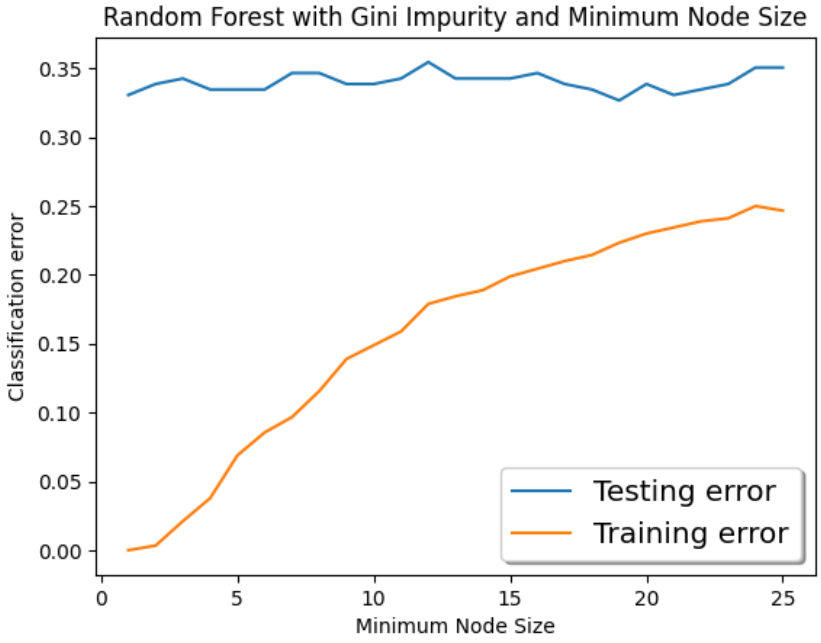
Solution C:

For minimum leaf node size the test error is minimized at minimum size = 12, and for maximum tree depth, the minimum test error is reached at max tree depth = 2. As noticed by the minimum test errors being reached in the earlier end of the range for maximum tree depth and later in the range for minimum leaf node size (larger minimum means less small/individual boundaries), it suggests that early stopping allows for improved test error.

Problem D [2 points]: 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.

Solution D:

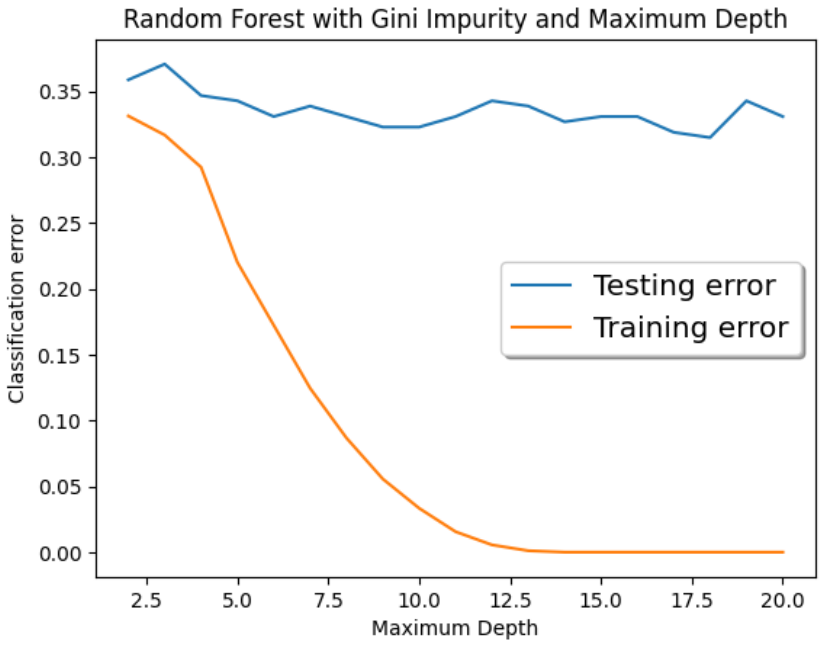
solution code



Problem E [2 points]: 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 E:

solution code



Problem F [4 points]: For both the minimal leaf node size and 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 two plots you derived.

Solution F:

For the minimum leaf node size, the test error is minimized at a minimum leaf node size = 19, and for maximum tree depth the test error is minimized at a max depth = 18. It does not seem like early stopping has much effect on the random forest classifier's performance as the test error seems relatively stable/ insignificantly different across all stopping limits and the max depth size is also optimized at a much later stopping point of depth 18, suggesting that early stopping is not effective for improving random forest classifier performance according to the test results.

Problem G [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 G:

In general for the plots, it is noticeable that the decision tree plots display improvement with early stopping while random forest classifiers do not. Additionally overall, random forest classifier tend to output lower test errors. These trends can be understood by the fact that decision trees can suffer from over fitting without early stopping as

higher depths/ smaller leaf node sizes leave the model vulnerable to overfitting on the training data and lower test performance, which is why early stopping leads to improvement. On the other hand, random forest classifiers take basically a general consensus of multiple decision trees so while each decision tree may experience some overfitting without early stopping the 'averaged' consensus of all trees into a final decision by the classifier results in the overfitting already being minimized as the overfitting effects of each tree are minimized by evaluating on many different trees. Thus, the random forest classifier sees little improvement with early stopping and in general does not overfit and thus produces seemingly consistant better performance versus a single decision tree.

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

$$H(x) = \text{sign}(f(x)) = \text{sign}\left(\sum_{i=1}^{T} \alpha_t h_t(x)\right).$$

Suppose $\{(x_1, y_1), ..., (x_N, y_N)\} \subset \mathbb{R}^m \times \{-1, 1\}$ 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:

$$E = \frac{1}{N}\sum_{i=1}^{N} \exp(-y_i f(x_i)) \geq \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}(H(x_i) \neq y_i),$$

where $\mathbb{1}$ is the indicator function.

Solution A:

We will show that the training set error $\frac{1}{N}\sum_{i=1}^{N} \mathbb{1}(H(x_i) \neq y_i)$ *is bounded from above by exponential loss function* $E = \frac{1}{N}\sum_{i=1}^{N} \exp(-y_i f(x_i))$*, by showing that for each iteration of* $i \in \sum_{i=1}^{N}$*, that* $\exp(-y_i f(x_i)) \geq \mathbb{1}(H(x_i) \neq y_i)$*. For when* $(H(x_i)y_i)$ *this is trivial since* $\mathbb{1}(H(x_i) \neq y_i) = 0$ *and since* $-y_i f(x_i) \neq 0$*,* $\exp(-y_i f(x_i)) > 0$ *and thus for every* $i \in \sum_{i=1}^{N}$ *such that* $H(x_i) = y_i$*,* $\exp(-y_i f(x_i)) \geq \mathbb{1}(H(x_i) \neq y_i)$*. For* $i \in \sum_{i=1}^{N}$*, where* $H(x_i) \neq y_i$*, we have that* $\mathbb{1}(H(x_i) \neq y_i) = 1$*. However, we also have that* $-y_i f(x_i) > 0$ *since* $y_i, f(x_i)$ *must have opposite signs for* $H(x_i) \neq y_i$*. As a result, since* $-y_i f(x_i) > 0$*,* $\exp(-y_i f(x_i)) > 1$ *meaning that for all* $i \in \sum_{i=1}^{N}$ *such that* $H(x_i) \neq y_i$*,* $\mathbb{1}(H(x_i) \neq y_i) = 1 < \exp(-y_i f(x_i))$*. We have thus show* $\forall i \in \sum_{i=1}^{N}$*, that* $\exp(-y_i f(x_i)) \geq \mathbb{1}(H(x_i) \neq y_i)$ *and therefore* $E = \frac{1}{N}\sum_{i=1}^{N} \exp(-y_i f(x_i)) \geq \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}(H(x_i) \neq y_i)$ *must be true.*

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}:

$$Z_t = \sum_{i=1}^{N} D_t(i) \exp(-\alpha_t y_i h_t(x_i)).$$

Solution B:

We know that the update weighting for D_{t+1} is recursive and given by $D_{t+1} = \frac{D_t \exp(-a_t y_i h_t(x_i))}{Z_t}$. Notice by the recursive pattern that any D_t can be solved explicitly with D_1 as $D_2 = \frac{D_1 \exp(-a_1 y_i h_1(x_i))}{Z_1}$ where $D_1 = \frac{1}{N}$ and for each increasing i, D_i is simply some function of D_1 multiplied by $\frac{\exp(-a_{i-1} y_t h_{i-1}(x_t))}{Z_{i-1}}$. This is specifically defined as $D_{T+1} = D_1 * \Pi_{n=1}^{T} \frac{\exp(-a_n y_i h_n(x_i))}{Z_n} = \frac{1}{N} * \Pi_{n=1}^{T} \frac{\exp(-a_n y_i h_n(x_i))}{Z_n}$

Problem C [2 points]: Show that $E = \sum_{i=1}^{N} \frac{1}{N} e^{\sum_{t=1}^{T} -\alpha_t y_i h_t(x_i)}$.

Solution C:

Given exponential loss function $E = \frac{1}{N} \sum_{i=1}^{N} \exp(-y_i f(x_i))$ notice that $f(x_i) = \sum_{i=t}^{T} \alpha_t h_t(x)$ since sign$(f(x)) =$ sign $\left(\sum_{i=1}^{T} \alpha_t h_t(x) \right)$. Substituting we get $E = \frac{1}{N} \sum_{i=1}^{N} \exp(-y_i f(x_i)) = \frac{1}{N} \sum_{i=1}^{N} \exp(-y_i \sum_{t=1}^{T} \alpha_t h_t(x))$. Since $\frac{1}{N}$ is constant for all $i \in \sum_{i=1}^{N}$ we can maintain equivalence when moving that term inside the summation and conclude that $E = \frac{1}{N} \sum_{i=1}^{N} \exp(-y_i f(x_i)) = \sum_{i=1}^{N} \frac{1}{N} \exp(-y_i \sum_{t=1}^{T} \alpha_t h_t(x))$, and thus since y_i is constant for $\sum_{i=1}^{T}$ we can move it into the second summation and conclude that $E = \sum_{i=1}^{N} \frac{1}{N} e^{-\sum_{t=1}^{T} \alpha_t y_i h_t(x)}$

Problem D [5 points]: Show that

$$E = \prod_{t=1}^{T} Z_t.$$

Hint: *Recall that $\sum_{i=1}^{N} D_t(i) = 1$ because D is a distribution.*

Solution D:

From (B) we have:
$D_{T+1} = \frac{1}{N} * \Pi_{t=1}^{T} \frac{\exp(-a_t y_i h_t(x_i))}{Z_t}$

$= \frac{1}{N} * \Pi_{t=1}^{T} \exp(-a_t y_i h_t(x_i)) \Pi_{t=1}^{T} \frac{1}{Z_t}$

$D_{T+1} * \Pi_{t=1}^{T} Z_t = \frac{1}{N} * \Pi_{t=1}^{T} \exp(-a_t y_i h_t(x_i))$

$= \frac{1}{N} * e^{\sum_{t=1}^{T} (-a_t y_i h_t(x_i))}$

Now if we sum both sides from $1 \rightarrow N$ *we get:*

$\sum_{i=1}^{N} D_{T+1} * \Pi_{t=1}^{T} Z_t = \sum_{i=1}^{N} \frac{1}{N} * e^{\sum_{t=1}^{T}(-a_t y_i h_t(x_i))}$

Since $\Pi_{t=1}^{T} Z_t$ *is constant over all* $i \in \sum_{i=1}^{N}$ *and* D_{T+1} *is a distribution over all* N *we can then conclude the equivalent that:* $\Pi_{t=1}^{T} Z_t = \sum_{i=1}^{N} \frac{1}{N} * e^{\sum_{t=1}^{T}(-a_t y_i h_t(x_i))} = E$

Problem E [5 points]: Show that the normalizer Z_t can be written as

$$Z_t = (1 - \epsilon_t)\exp(-\alpha_t) + \epsilon_t \exp(\alpha_t)$$

where ϵ_t is the training set error of weak classifier h_t for the weighted dataset:

$$\epsilon_t = \sum_{i=1}^{N} D_t(i)\mathbb{1}(h_t(x_i) \neq y_i).$$

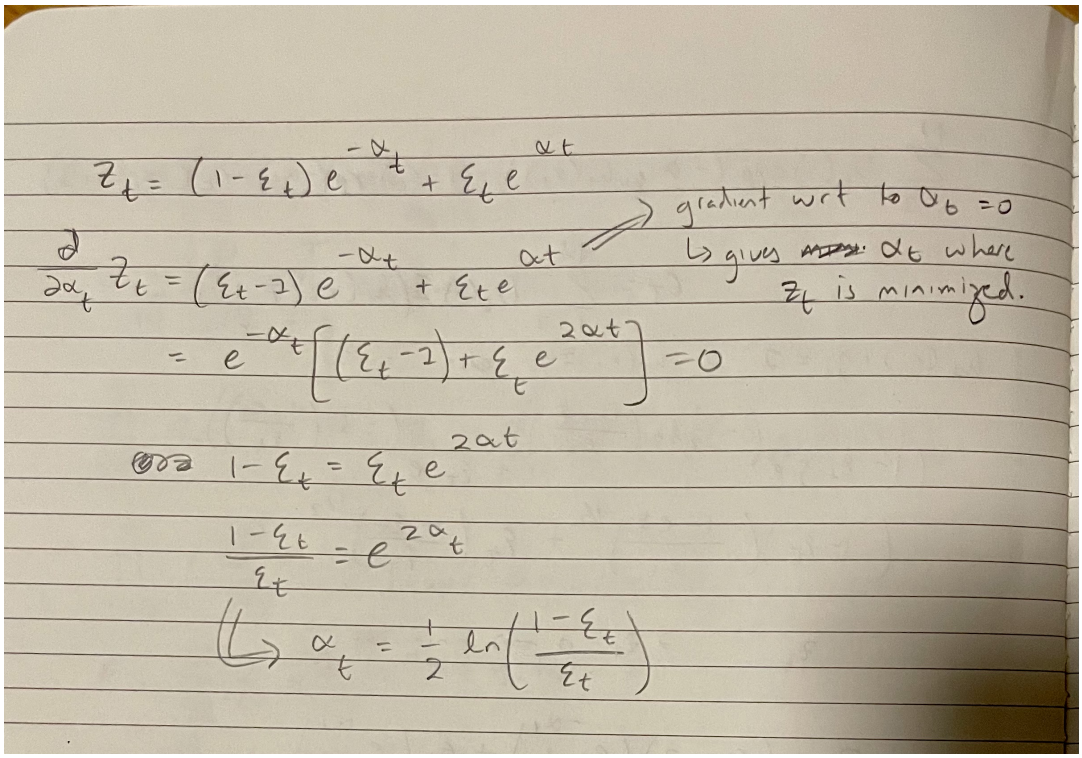
Solution E:

Given the definition of $Z_t = \sum_{i=1}^{N} D_t(i)\exp(-\alpha_t y_i h_t(x_i))$. *Notice that* $y_i h_t(x_i) = 1$ *if* $y_i = h_t(x_i)$ *and -1 otherwise. Thus,* $\alpha_t y_i h_t(x_i)) = -\alpha$ *if* $y_i = h_t(x_i)$ *and* α *otherwise. This results in us being able to redefine* Z_t *as* $Z_t = \sum_{i=1}^{N} D_t(i)(e^{-\alpha} * (\mathbb{1}(y_i = h_t(x_i))) + e^{\alpha}(\mathbb{1}(y_i \neq h_t(x_i))))$.. *This term is equivalent to the sums of* $Z_t = e^{-\alpha}\sum_{i=1}^{N} D_t(i)(\mathbb{1}(y_i = h_t(x_i))) + e^{\alpha}\sum_{i=1}^{N} D_t(i)(\mathbb{1}(y_i \neq h_t(x_i)))$. *Notice now that* $\sum_{i=1}^{N} D_t(i)(\mathbb{1}(y_i \neq h_t(x_i))) = \epsilon_t$ *and* $\sum_{i=1}^{N} D_t(i)(\mathbb{1}(y_i = h_t(x_i))) = 1 - \epsilon_t$, *and thus we get that* $Z_t = (1-\epsilon_t)\exp(-\alpha_t) + \epsilon_t \exp(\alpha_t)$

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:

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

Solution F:



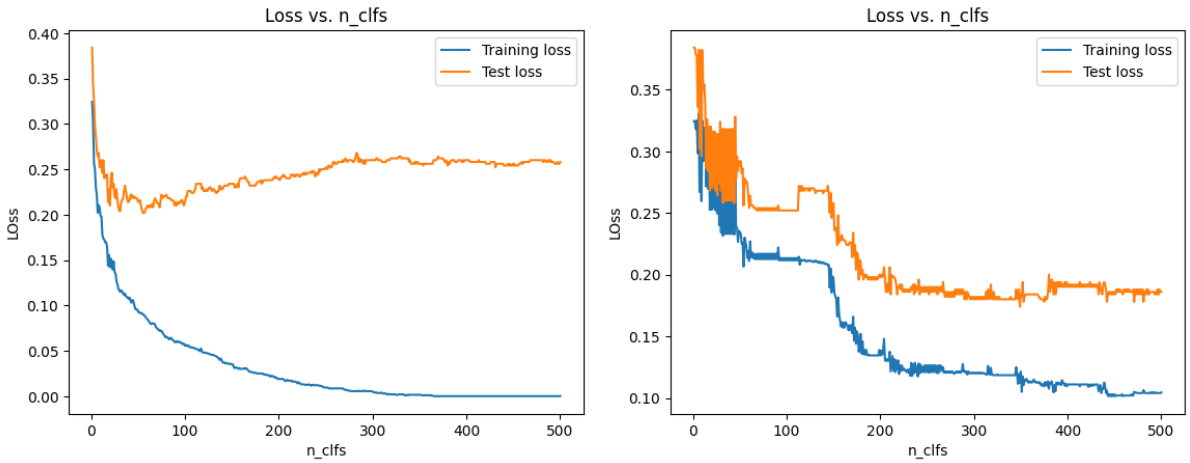
Problem G [14 points]: Implement the `GradientBoosting.fit()` and `AdaBoost.fit()` methods in the notebook provided for you. Some important notes and guidelines follow:

- 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.

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:

Link to gradient boost and AdaBoost fit as well as visuals



The loss curves for gradient boost (left) and AdaBoost (right) appear to both be convergent with gradient boost descending to 0 training loss and AdaBoost descending to .1 training loss. Additionally, the training loss descent of gradient boost is much smoother than AdaBoost. However the test loss curve for AdaBoost appears to have better convergence than gradient boost as n_clfs increases because the gradient boost test loss optimizes at around .2 for ¡ 100 clfs while the AdaBoost test loss continues to decrease beyond that point reaching ¡.2 final test loss on n_clfs ¿ 300.

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

Solution I:

Considering the test error and the difference between test and training error it appears that the AdaBoost model performed better on the classification dataset compared to gradient boosting in that the test error was lower and the difference between test and training error was lower signifying better generalization.

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:

As the the dataset weights change across the animation we see that the dataset weights change a lot around the true decision boundaries, and change less on the interior of these boundaries suggesting that the largest dataset weights are at points where the model is not very certain about it's prediction, (low/ incorrect $f(x)$ value at that point).