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

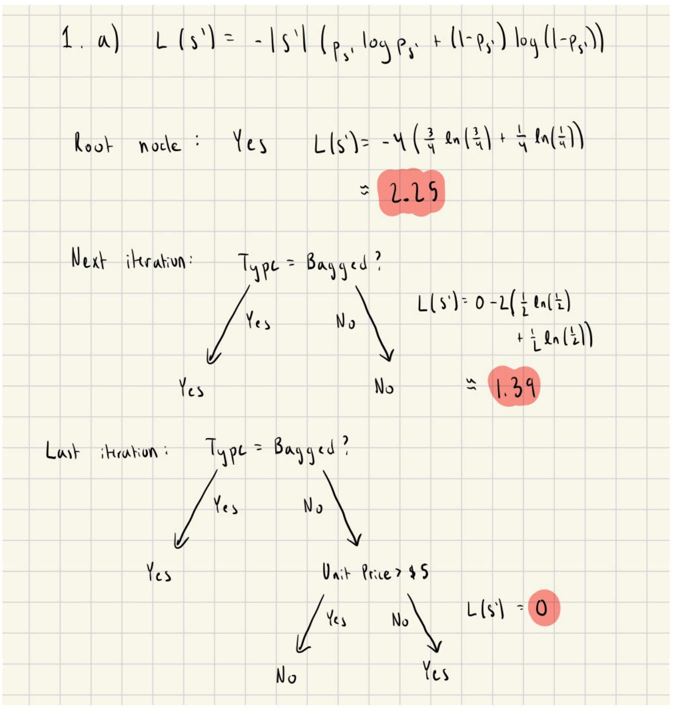
Problem A [7 points]: Consider the following data, where given information about some food you must predict whether it is healthy:

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:



1. a) $L(s') = -|s'|\left(p_{s'}\log p_{s'} + (1-p_{s'})\log(1-p_{s'})\right)$

Root node : Yes $L(s') = -4\left(\frac{3}{4}\ln\left(\frac{3}{4}\right) + \frac{1}{4}\ln\left(\frac{1}{4}\right)\right)$

$\approx \boxed{2.25}$

Next iteration:

$$\begin{tikzcd} & \text{Type = Bagged?} \arrow[dl, "\text{Yes}"'] \arrow[dr, "\text{No}"] & \\ \text{Yes} & & \text{No} \end{tikzcd}$$

$L(s') = 0 - 2\left(\frac{1}{2}\ln\left(\frac{1}{2}\right) + \frac{1}{2}\ln\left(\frac{1}{2}\right)\right)$

$\approx \boxed{1.39}$

Last iteration:

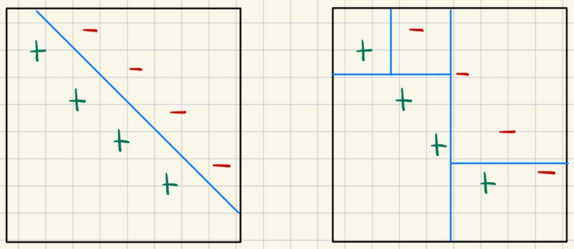
$$\begin{tikzcd} & \text{Type = Bagged?} \arrow[dl, "\text{Yes}"'] \arrow[dr, "\text{No}"] & & \\ \text{Yes} & & \text{Unit Price} > \$5 \arrow[dl, "\text{Yes}"'] \arrow[dr, "\text{No}"] & \\ & \text{No} & & \text{Yes} \end{tikzcd}$$

$L(s') = \boxed{0}$

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.

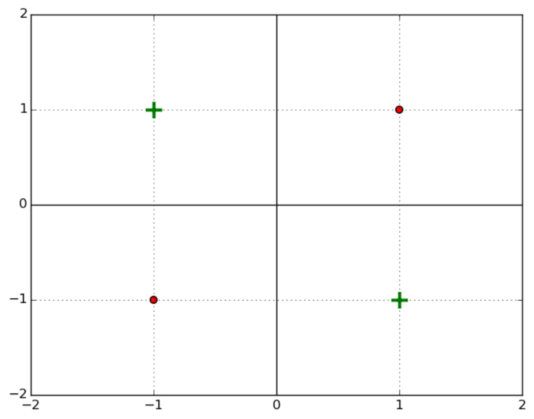
Solution B:

b) No, a decision tree is not always better. As discussed in lecture 5, DT cannot easily model diagonal boundaries b/c they are axis aligned.



Easy SVM solution   Complex DT solution

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))?

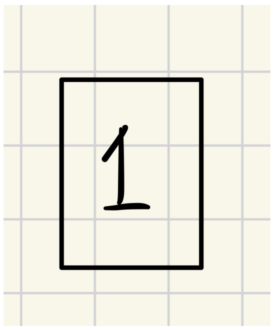
Solution C: *The resulting tree is just a root node (see drawing below). This occurs due to the definition of the stopping condition. Consider the Gini index at the root node.*

$$L(S) = |4| * (1 - 0.5^2 - (1 - 0.5)^2) = 2$$

Then, using the greedy top-down approach, splitting the data in any fashion will result in the same Gini index.

$$L(S) = |2| * (1 - 0.5^2 - (1 - 0.5)^2) + |2| * (1 - 0.5^2 - (1 - 0.5)^2) = 2$$

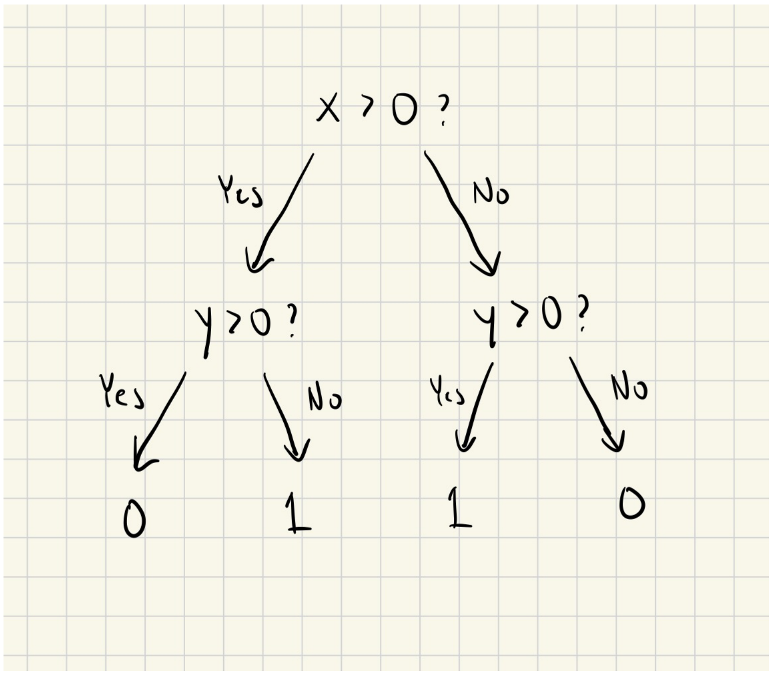
Therefore, since no split of a node results in any reduction in impurity here, we stop our decision tree at the root node (1). The classification error at the root node is $\frac{2}{4} = \frac{1}{2}$.



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).

Solution C:



In order to allow the top-down greedy induction to progress past the simple root-node decision tree, we have to penalize larger leaf sizes. The modified Gini index looks as such

$$L(S') = |S'|^2(1 - p_{s'}^2 - (1 - p_{s'})^2)$$

Then, at the root node, we have an $|S'|^2$ factor of 16 instead of 4 in the iteration after that.

Pros: We can add levels to our decision tree that ultimately result in better classification errors even if not immediately apparent after 1 level.

Cons: This modified impurity measure is prone to overfitting/over-complication of our model. This is because the model seeks to lower leaf sizes and may sacrifice out-of-sample generality while in this pursuit. It is also possible that we add levels to the model but do not lower the classification error, which overcomplicated the model for no reason.

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: *The largest number of unique thresholds we need to achieve zero classification training error is 99. Consider all 100 points falling in one line in R^2, with classifications alternating with each point on the line. In that case, we would need 99 unique internal nodes to properly classify the data. In general. if we have N data points in a 2-D dataset, we need N - 1 unique internal nodes.*

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 of the number of possible splits we must consider to find the one that most reduces impurity is O(DN). We have to consider each feature that we can split on, and for each of those features, there are N - 1 unique child sets. This is discussed in Lecture 5.*

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:

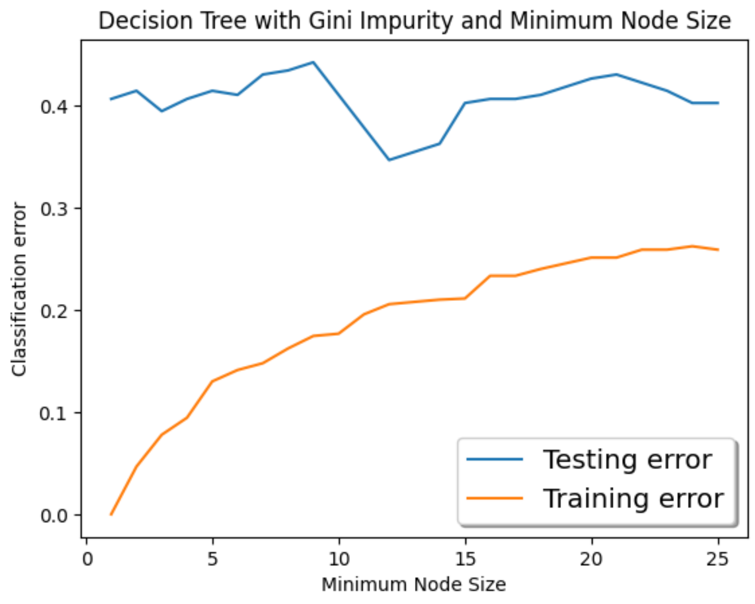
https://archive.ics.uci.edu/ml/datasets/Diabetic+Retinopathy+Debrecen+Data+Set

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.

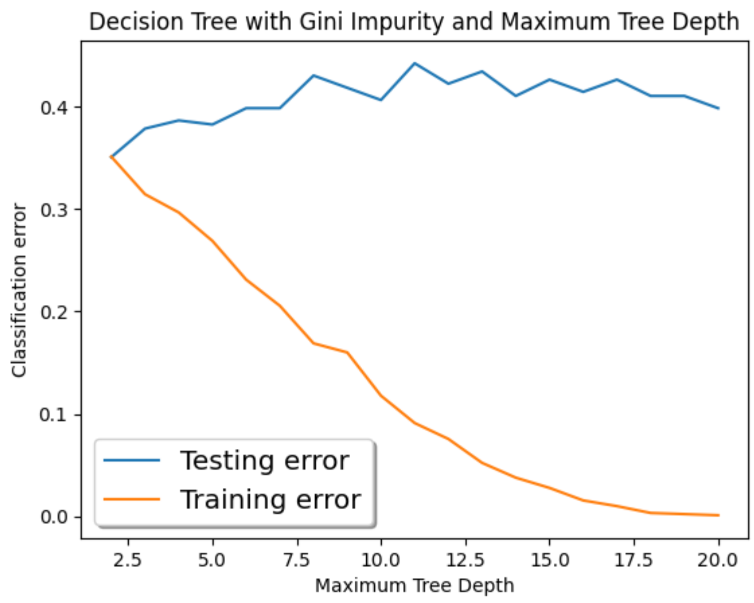
Solution A:

https://colab.research.google.com/drive/1dfAsRUlri5XLjBDC_8R-MalVdADNypfJ?usp=sharing



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:

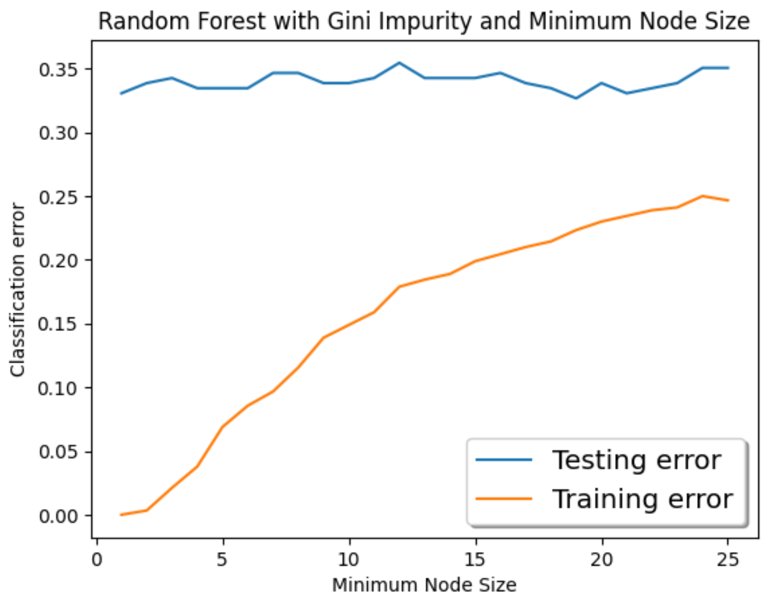


Problem C [4 points]: For both the minimal leaf node size and maximum depth parameters tested in the last two questions, which parameter value minimizes the test error? What effects does early stopping have on the performance of a decision tree model? Please justify your answer based on the two plots you derived.

Solution C: *The minimal leaf node size of 12 minimized the test error while the max depth of 2 minimized the test error. Early stopping is beneficial for model performance because we see that as the minimal leaf node size decreases past 12 and the max depth increases past 2, test error increases. Thus, early stopping helps us prevent over-fitting which decision trees are especially prone to. Note that a smaller minimal leaf node size leads to a more complicated model while a larger max depth size leads to a more complicated model.*

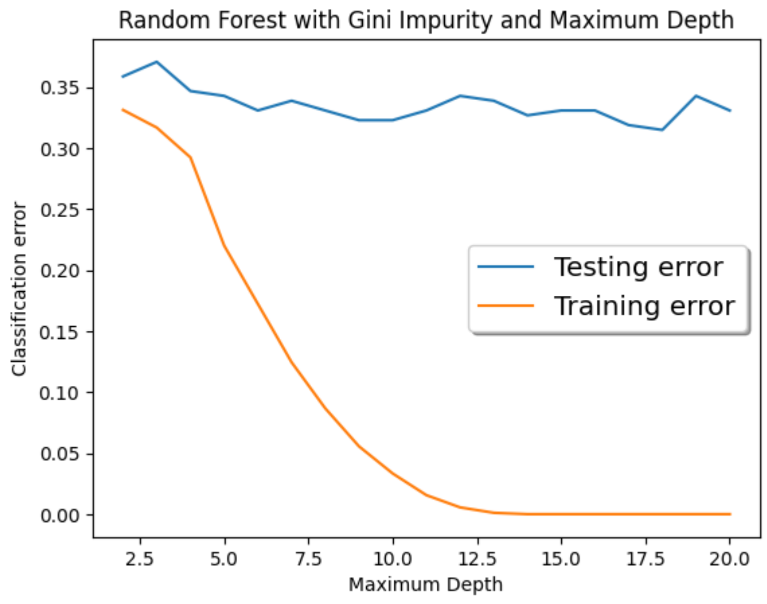
Problem D [2 points]: Train a random forest classifier using Gini as the impurity measure, minimal leaf node size as early stopping criterion, and 1,000 trees in the forest. Try different node sizes from 1 to 25 in increments of 1. Then, on a single plot, plot both training and test classification error versus leaf node size.

Solution D:



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:



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: *The minimal leaf node size of 19 minimized the test error while the max depth of 18 minimized the test error. For random forest models, early stopping does not seem to have much effect. We see this because test error largely stays the same as we vary our minimal leaf node sizes and our max depth.*

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: *We see that the test error for the decision tree plots are generally larger than the test error for the random forest plots. This makes sense because random forests are an application of bagging to normal decision trees. The random forests help account for overfitting in decision trees by generating several distinct "forests" using our sample data and random feature sampling and aggregating the model predictions together. This also explains why test error is relatively constant for random forests but we see more fluctuation in decision trees. The prevention of overfitting in random forests means that it is less reliant on regularization parameters such as minimal leaf node size and maximum depth.*

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:

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

Multiply through by N + remove series notation

$\ast$ Prove $\exp(-y_i f(x)) \geq \mathbb{1}(H(x_i) \neq y_i) \;\; \forall \; i$

Case 1 : $\text{sign}(y_i) = \text{sign}(f(x_i))$

- $-y_i f(x) < 0 \Rightarrow \exp(-y_i f(x)) \geq 0$
- $\mathbb{1}(H(x_i) \neq y_i) = 0 \leftarrow$ because $H(x_i) = y_i$
- Therefore, $\exp(-y_i f(x)) \geq \mathbb{1}(H(x_i) \neq y_i)$

Case 1 : $\text{sign}(y_i) \neq \text{sign}(f(x_i))$

b/c $\exp(0) = 1$ + inc after that

- $-y_i f(x) > 0 \Rightarrow \exp(-y_i f(x)) \geq 1$
- $\mathbb{1}(H(x_i) \neq y_i) = 1 \leftarrow$ because $H(x_i) \neq y_i$
- Therefore, $\exp(-y_i f(x)) \geq \mathbb{1}(H(x_i) \neq y_i)$

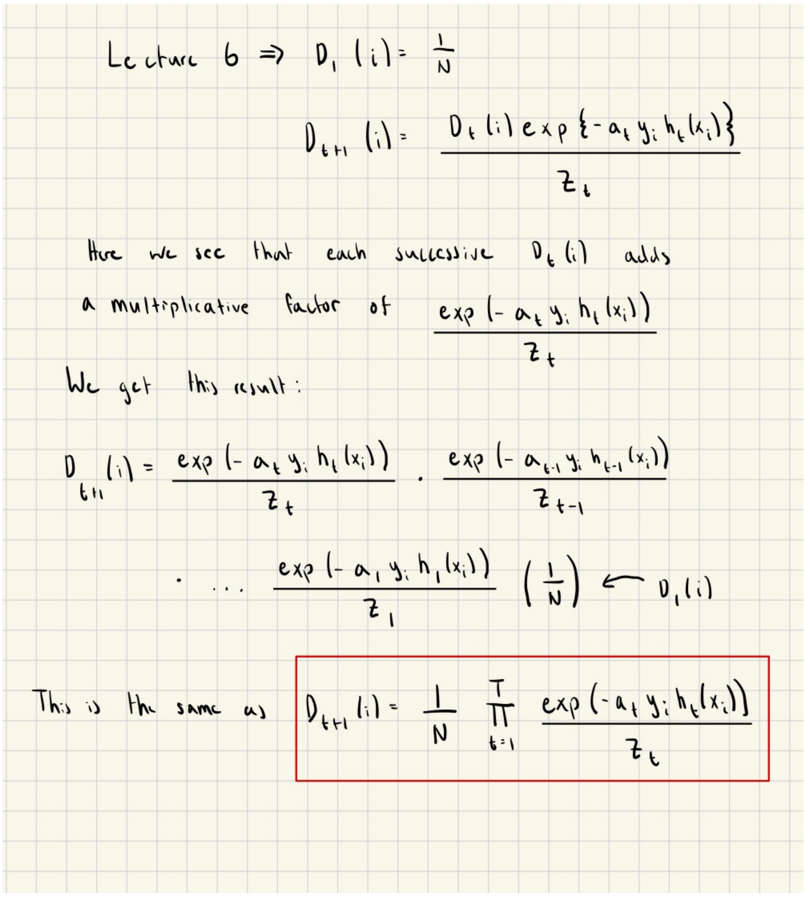
Since we've shown $\exp(-y_i f(x)) \geq \mathbb{1}(H(x_i) \neq y_i) \;\; \forall \; i$

we've also shown $\boxed{E = \frac{1}{N}\sum_{i=1}^{N} \exp(-y_i f(x)) \geq \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}(H(x_i) \neq y_i)}$

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}:

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

Solution B:



Lecture 6 $\Rightarrow$ $D_1(i) = \frac{1}{N}$

$$D_{t+1}(i) = \frac{D_t(i) \exp\{-a_t y_i h_t(x_i)\}}{Z_t}$$

Here we see that each successive $D_t(i)$ adds a multiplicative factor of $\frac{\exp(-a_t y_i h_t(x_i))}{Z_t}$

We get this result:

$$D_{t+1}(i) = \frac{\exp(-a_t y_i h_t(x_i))}{Z_t} \cdot \frac{\exp(-a_{t-1} y_i h_{t-1}(x_i))}{Z_{t-1}} \cdot \ldots \frac{\exp(-a_1 y_i h_1(x_i))}{Z_1} \left(\frac{1}{N}\right) \leftarrow D_1(i)$$

This is the same as $\boxed{D_{t+1}(i) = \frac{1}{N} \prod_{t=1}^{T} \frac{\exp(-a_t y_i h_t(x_i))}{Z_t}}$

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:

Start with $E = \frac{1}{N} \sum_{i=1}^{N} \exp(-y_i f(x_i))$

$$f(x) = \sum_{i=1}^{T} a_t h_t(x)$$

$\exp(x) = e^x$

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

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

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:

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

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

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

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

$\uparrow$ product of $e^x e^y = e^{x+y}$

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

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

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

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

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:

Prove $Z_t = (1-\epsilon_t)\exp(-a_t) + \epsilon_t \exp(a_t)$

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

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

Case 1: $h_t(x_i) = y_i$

- $h_t(x_i)\, y_i = 1$ + $\mathbb{1}(h_t(x_i) \neq y_i) = 0$
- $Z_t = \underbrace{\sum_{i=1}^{N} D_t(i)}_{=1} \exp(-a_t) = \exp(-a_t)$
- $\epsilon = 0 \Rightarrow Z_t = (1-0)\exp(-a_t) + (0)\exp(a_t)$

  $= \exp(-a_t)$

equivalent

Case 2: $h_t(x_i) \neq y_i$

- $h_t(x_i)\, y_i = -1$ + $\mathbb{1}(h_t(x_i) \neq y_i) = 1$
- $Z_t = \underbrace{\sum_{i=1}^{N} D_t(i)}_{=1} \exp(a_t) = \exp(a_t)$
- $\epsilon = 1 \Rightarrow Z_t = (1-1)\exp(-a_t) + (1)\exp(a_t)$

  $= \exp(a_t)$

equivalent

$$\boxed{\text{Therefore, the normalizer } Z_t = (1-\epsilon_t)\exp(-a_t) + \epsilon_t \exp(a_t)}$$

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:

Established that $Z_t = (1-\epsilon_t)\exp(-a_t) + \epsilon_t \exp(a_t)$

We want to find when $\frac{d}{da_t}(Z_t) = 0$ to minimize it

$$0 = \frac{d}{da_t}\left[(1-\epsilon_t)\exp(-a_t) + \epsilon_t\exp(a_t)\right]$$

$$0 = -(1-\epsilon_t)\exp(-a_t) + \epsilon_t \exp(a_t)$$

$$\epsilon_t \exp(a_t) = (1-\epsilon_t)\exp(-a_t)$$ ☆ Note $\frac{e^x}{e^y} = e^{x-y}$

$$\epsilon_t \exp(2a_t) = 1-\epsilon_t$$

$$\exp(2a_t) = \frac{1-\epsilon_t}{\epsilon_t}$$

$$2a_t = \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$$

$$\boxed{a_t = \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)}$$

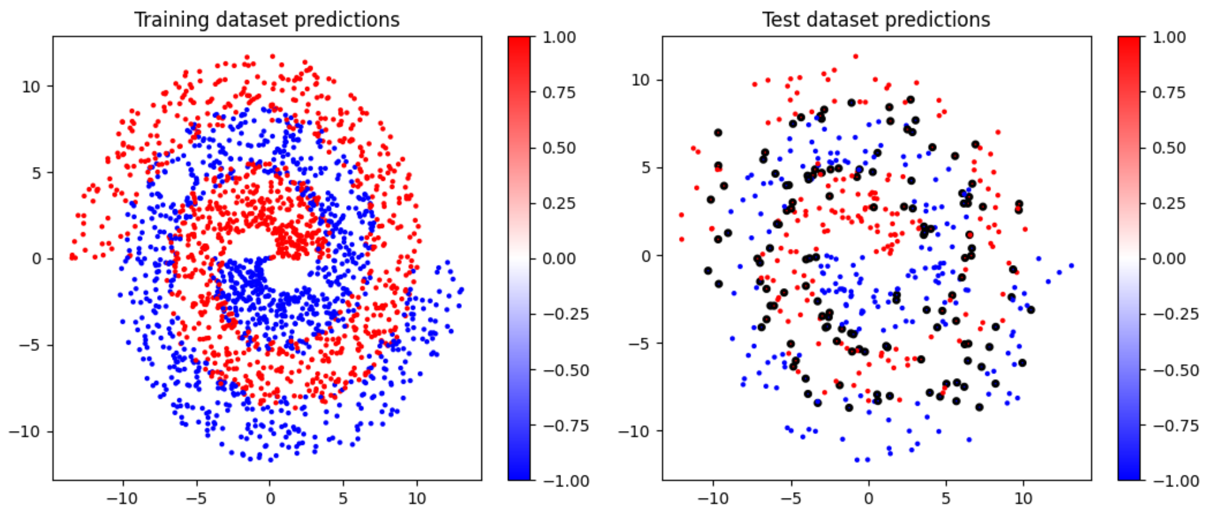
Problem G [14 points]: Implement the `GradientBoosting.fit()` and `AdaBoost.fit()` methods in the notebook provided for you. Some important notes and guidelines follow:

- 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.

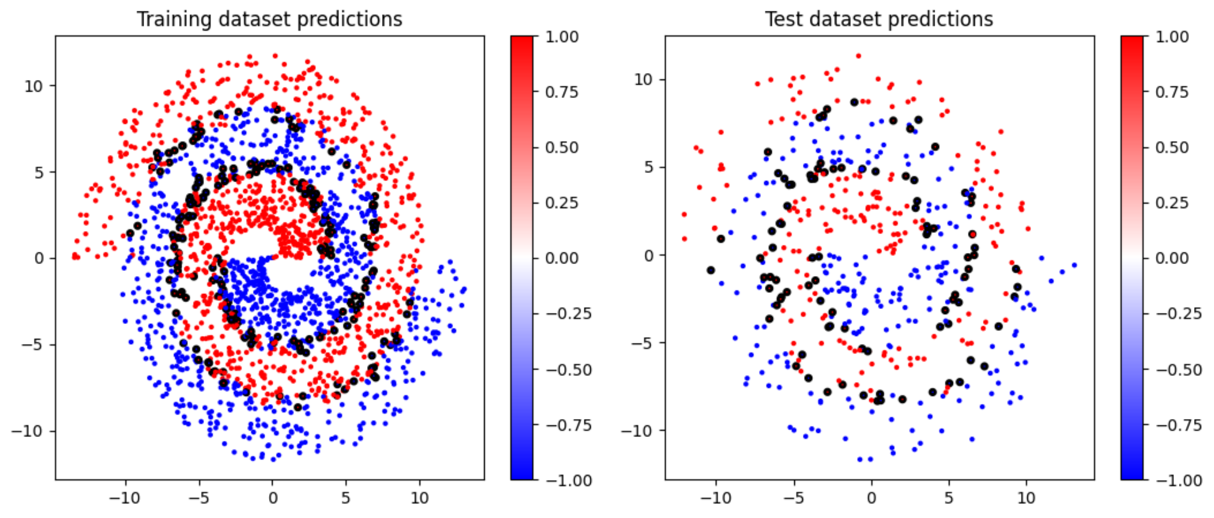
- `AdaBoost.fit()` should additionally return an (N, T) shaped numpy array `D` such that `D[:, t]` contains D_{t+1} for each $t \in \{0, \ldots,$ `self.n_clfs`$\}$.
- For the `AdaBoost.fit()` method, **use the 0/1 loss** instead of the exponential loss.
- The only Sklearn classes that you may use in implementing your boosting fit functions are the DecisionTreeRegressor and DecisionTreeClassifier, not GradientBoostingRegressor.

Solution G:

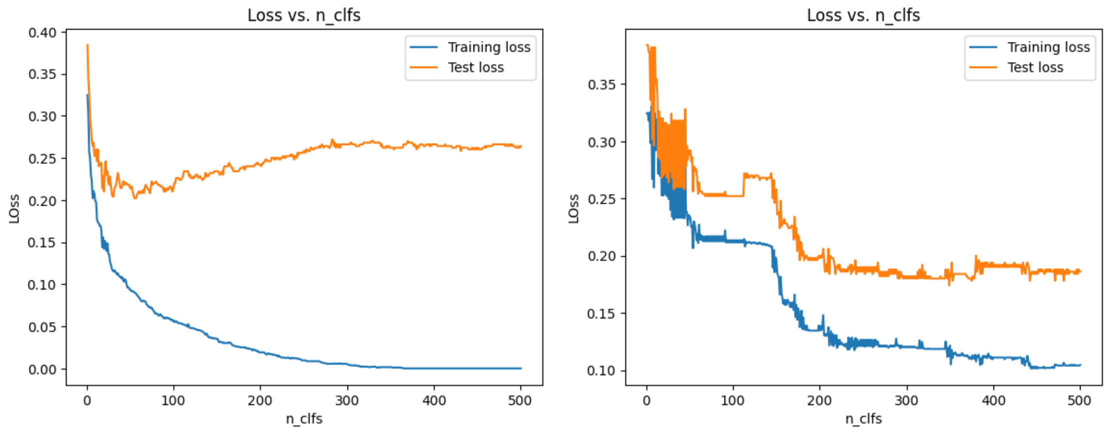
https://colab.research.google.com/drive/1S7fK-Wf4kKwniPvW8LbVjrgH1vtF7S7j?usp=sharing



Gradient Boost Predictions pictured above



Ada Boost Predictions pictured above



Gradient Boost Loss (L) and AdaBoost Loss (R) pictured above

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 Gradient Boost Loss curve is smoother and more differentiable than the AdaBoost Loss curve. This is because it fits on the residuals (making it more smooth over time). However, we see in both curves that as the number of classifiers increases the curve starts to flatten and approach a limiting value. The gradient boost training loss approaches 0 while the AdaBoost training loss hovers a little higher (around 0.10). Howveer, we can still be confident in the AdaBoost model because it outperforms Gradient Boost handily in test loss, indicating it is less prone to overfitting.

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

Solution I:

The final training loss is around 0 for Gradient Boost and around 0.10 for AdaBoost. The final test loss is around 0.26 for Gradient Boost and around 0.19 for AdaBoost. Therefore, although AdaBoost has worse in-sample performance, it is a better overall model because it generalizes out of sample better.

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:

For AdaBoost, the dataset weights are the largest for points that are prone to misclassification/closer to the decision boundary (think about SVMs). They are smallest for the points that are safely away from the decision boundary.