Machine Learning & Data Mining
Set 3

Caltech CS/CNS/EE 155
January 26th, 2022

Policies

- Due 9 PM PST, January 26th on 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.

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 Decision Trees [30 Points]

Relevant materials: Lecture 5

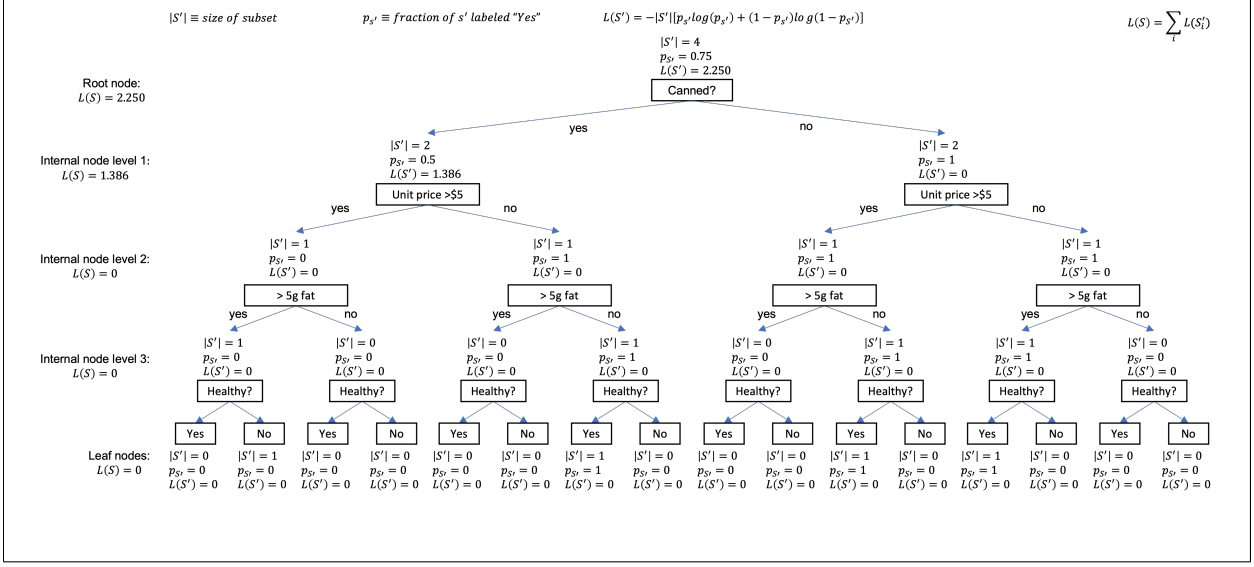
Problem A [7 points]: Consider the following data, where given information about some food you must predict whether it is healthy:

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.

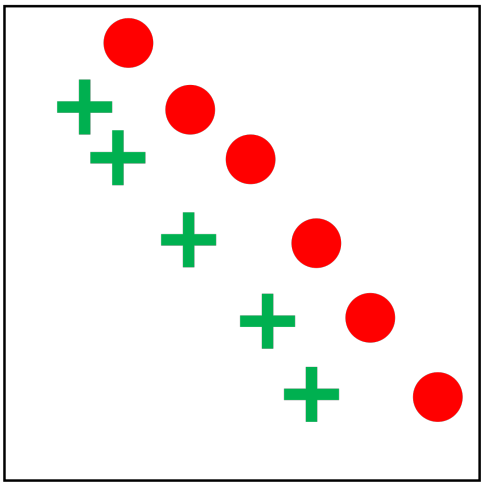
Submit a drawing of your tree showing the impurity reduction yielded by each split (including root) in your decision tree.

Solution A:



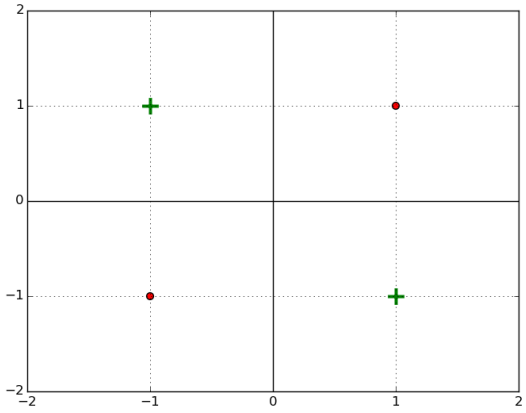
Problem B [4 points]: Compared to a linear classifier, is a decision tree always preferred for classification problems? If not, draw a simple 2-D dataset that can be perfectly classified by a simple linear classifier but which requires an overly complex decision tree to perfectly classify.

Solution B:



A decision tree is not always preferred for classification, because decision tree partitions can only be axis-aligned. As such, decision trees are not ideal for data sets that require partitioning along a diagonal.

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))?

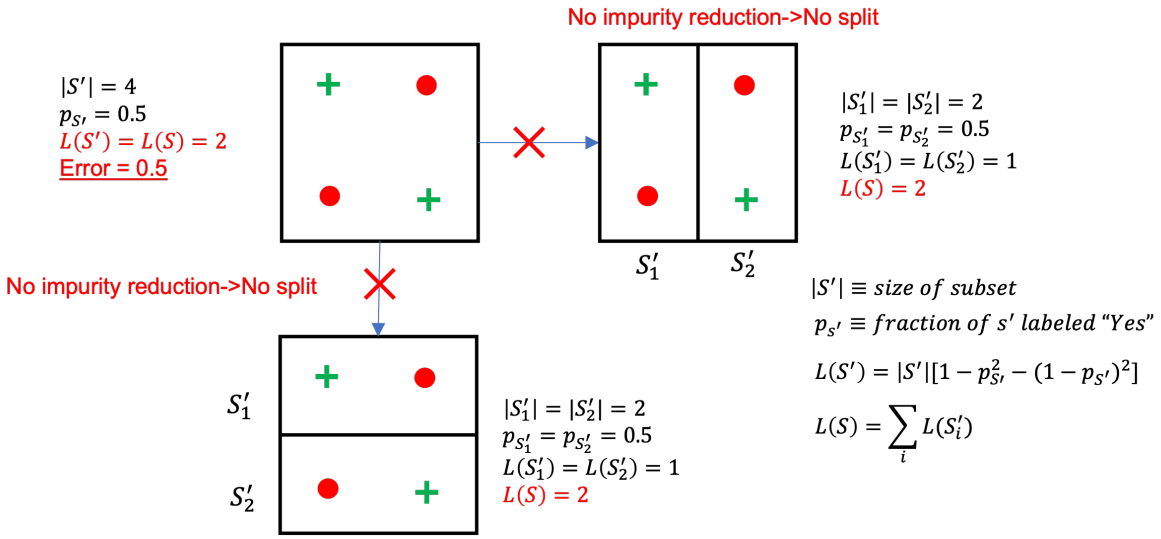
ii. [5 points]: Submit a drawing of a two-level decision tree that classifies the above dataset with zero classification error. (You don't need to use any particular training algorithm to produce the tree.)

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.

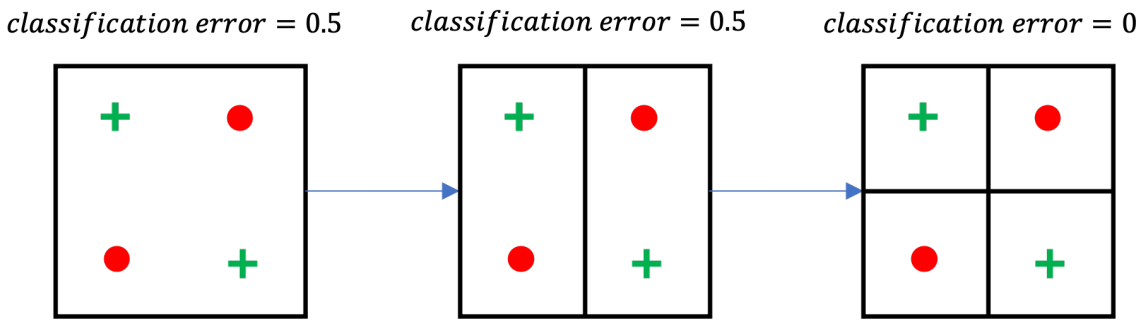
Solution C:

i.



The resulting tree is simply the starting data set with no split, as the root node cannot be split using an axis-aligned partition that will reduce the impurity ($impurity = 2$ for the two possible splits). classification error $=$ 0.5.

ii.



If we only consider the Bernoulli variance, entropy, and Gini index as the impurity measure, we will not be able to produce the above decision tree with the given stopping condition and data set. This is because every root node and any resulting internal node from the root node will always have a $p_{S'} = 0.5$ while equally splitting the size of the subset. Practically, this means that the impurity will not change between the root node and the first level of internal nodes, which satisfies the criteria of stopping at the level of the root node. Technically, if you used $L(S') = |S'| - p_{S'}$, this would lead to top-down greedy induction using the above stopping condition (impurity would be 3.5, 3, and 2 for each level). This impurity measure would be good at decreasing the impurity given this specific situation, but this produces a linear loss function, which is difficult to minimize. The pros are that it will always be able to minimize your data set, but it may be computationally difficult to do, especially in higher dimensions.

iii. Each internal node represents a query or splitting (e.g. a line separating a partition in two dimensions), and in a worst-case scenario (i.e. largest number of unique thresholds) to achieve zero classification error, you will need to split the data such that each partition only has one data point. As such, to make each partition contain only have one data point each, you need to do N-1 splittings (assuming only in 2D), or in the case of 100 data points, you need 99 unique thresholds.

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 worse case complexity is $\mathcal{O}(ND)$. Even if the feature space is continuous, the data set will have a maximum number of N different values (assuming each is unique). As such, the maximum number of discrete queries that you can do for a continuous feature is $\mathcal{O}(N)$. Therefore, the maximum number of queries is simply the number of points multiplied by the number of continuous features.*

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`

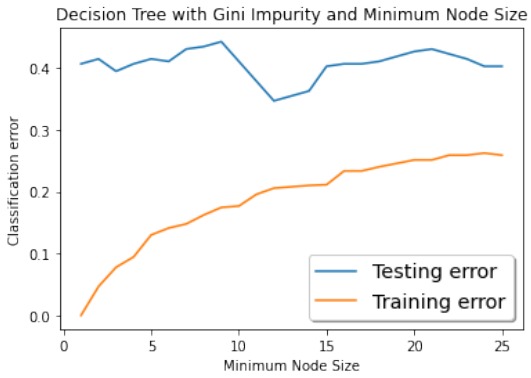
In the following question, your goal is to predict the diagnosis of diabetic retinopathy, which is the final column in the data matrix. Use the first 900 rows as training data, and the last 251 rows as validation data. Please feel free to use additional packages such as Scikit-Learn. Include your code 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: *Link to Google Colab notebook for problem 2.*
Solution A,i:



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.

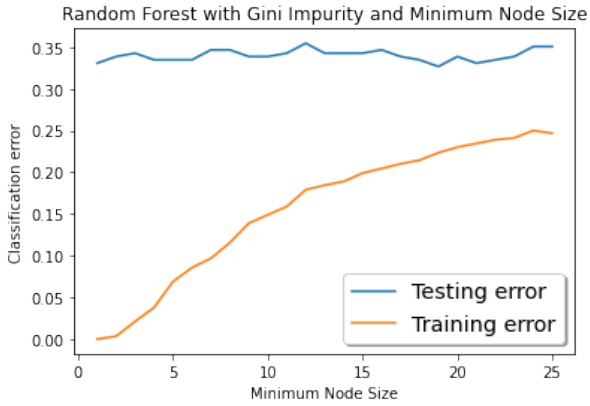
Solution B: *For solution A, I provided a plot that optimizes the minimum leaf node size parameter,* `min_samples_leaf`*. The test error is minimized when* `min_samples_leaf` $= 12$*. Using a stopping criteria for a decision tree classifier is clearly important, as different minimum node sizes produce different test errors. In fact, there is an optimal minimum node size that minimizes the test error, and this may vary depending on your data. Namely, the models with* `min_samples_leaf` < 10 *and* `min_samples_leaf` > 15 *perform about on par with one another. As such, stopping early can either hurt or help your model, and how it reacts to this stopping criteria most likely depends on the model you use and the data set that you have.*

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:
Solution C,i:



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 solution C, I provided a plot that optimizes the minimum leaf node size parameter,* `min_samples_leaf`*. The test error is minimized when* `min_samples_leaf` $= 19$*. Using a stopping criteria for a random forest model does not seem to be as important, as the test error is about the same no matter the value of* `min_samples_leaf`*. As you can see from the plots, the test error is approximately the same using the various values of* `min_samples_leaf`*. As such, stopping early most likely has little effect on the performance of a random forest model.*

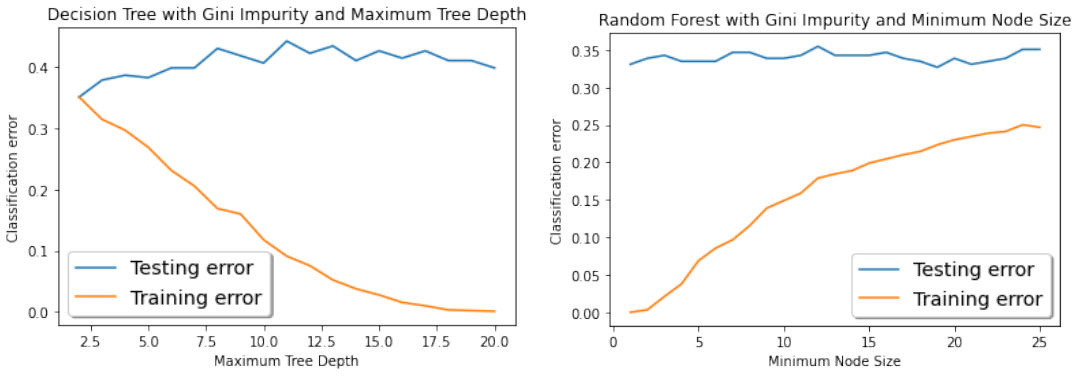
Problem E [4 points]: Do you observe any differences between the curves for the random forest and decision tree plots? If so, explain what could account for these differences.

Solution E: *The random forest seems to be less sensitive to parameters that define the model (e.g.* `min_samples_leaf`*). This is intuitive because we know that a random forest model should reduce the variance, meaning that different random forest models will provide predictions that are relatively speaking about the same (i.e. the expected value of the predictions is about the same) and are less sensitive to parameterization. This is most likely due to the fact that random forests sample both features and data at every node, de-correlating the trees and how they are affected by the parameterization (since it is all random).*

Extra Credit [7 points total] :

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

Solution F:



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: *The maximum tree depth parameter,* `max_depth`*, optimizes the decision tree model when* `max_depth` $= 2$ *and optimizes the random forest model when* `max_depth` $= 18$*. As in the case for the minimum leaf node size, the random forest is not affected by the maximum tree depth (i.e. the test error is about the same for different parameters) while the decision tree model has a defined optimal maximum tree depth that clearly minimizes the test error.*

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: *For the term* $\exp(-y_i f(x_i))$*, we know that* $y_i \subset \{-1, 1\}$*. As such,*

$$-y_i f(x_i) > 0 \text{ for } H(x_i) \neq y_i \text{ (an incorrectly labeled point)}$$
$$-y_i f(x_i) < 0 \text{ for } H(x_i) = y_i \text{ (a correctly labeled point)}$$

This translates to

$$\exp(-y_i f(x_i)) > 1 \text{ for } H(x_i) \neq y_i$$
$$\exp(-y_i f(x_i)) < 1 \text{ for } H(x_i) = y_i$$

Note that this term behaves such that it crosses 1 *when the aggregate scoring function,* $f(x)$*, (and thus the classifier* $H(x)$*, changes sign, thereby either moving from an incorrect label to correct label or vice versa. Additionally we know that the exponential function is always greater than 0.*

Now considering the term $\mathbb{1}(H(x_i) \neq y_i)$*, by definition*

$$\mathbb{1}(H(x_i) \neq y_i) = 1 \text{ for } H(x_i) \neq y_i$$
$$\mathbb{1}(H(x_i) \neq y_i) = 0 \text{ for } H(x_i) = y_i$$

As such, by comparing the two terms for the two conditions (correctly labeled and incorrectly labeled points), we can see that the exponential loss function is always greater than the indicator or 0/1 loss function, and always serves as an upper bound. The one caveat is when $-y_i f(x_i) = 0$*. As we can see,* $\exp(0) = 1$ *in this case, and*

the 0/1 loss function is discontinuous. As such, while the exponential function practically always serves as an upper bound, the discontinuity requires us to say that $\exp(-y_i f(x_i)) \geq \mathbb{1}(H(x_i) \neq y_i)$

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: *Given that* $D_1 = 1/N$ *and the equation for* $D_{t+1} = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$*, we know that*

$$\begin{aligned} D_2(i) &= \frac{\exp\left(-\alpha_1 y_i h_1(x_i)\right)}{N Z_1} \\ D_3(i) &= \frac{D_2(i) \exp\left(-\alpha_2 y_i h_2(x_i)\right)}{Z_2} \\ &= \frac{\exp\left(-\alpha_1 y_i h_1(x_i)\right)}{N Z_1} \frac{\exp\left(-\alpha_2 y_i h_2(x_i)\right)}{Z_2} \end{aligned}$$

Generalizing the solution to the $T + 1$ *case:*

$$D_{T+1}(i) = \frac{\prod_{t=1}^{T} \exp\left(-\alpha_t y_i h_t(x_i)\right)}{N \prod_{t=1}^{T} Z_1}$$

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: *From 3A, we know that* $E = \frac{1}{N} \sum_{i=1}^{N} \exp(-y_i f(x_i))$ *and* $f(x_i) = \sum_{t=1}^{T} \alpha_t h_t(x_i)$*. Substituting in, we get:* $E = \frac{1}{N} \sum_{i=1}^{N} \exp\left(\sum_{t=1}^{T} -\alpha_t y_i h_t(x_i)\right)$.

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 3B we can rearrange terms to find:*

$$D_{T+1}(i) = \frac{\prod_{t=1}^{T} \exp\left(-\alpha_t y_i h_t(x_i)\right)}{N \prod_{t=1}^{T} Z_1}$$

$$\prod_{t=1}^{T} \exp\left(-\alpha_t y_i h_t(x_i)\right) = N D_{T+1}(i) \prod_{t=1}^{T} Z_1$$

$$\exp\left(\sum_{t=1}^{T} -\alpha_t y_i h_t(x_i)\right) = N D_{T+1}(i) \prod_{t=1}^{T} Z_1$$

Substituting this to the term from Solution 3C and using the identity that $\sum_{i=1}^{N} D_t(i) = 1$*:*

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

$$E = \sum_{i=1}^{N} D_{T+1}(i) \prod_{t=1}^{T} Z_1$$

$$E = \prod_{t=1}^{T} Z_1$$

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: *From 3B, we know* $Z_t = \sum_{i=1}^{N} D_t(i)\exp(-\alpha_t y_i h_t(x_i))$. *We also know that* $y \subset \{-1, 1\}$ *and* $h_t(x_i) \subset \{-1, 1\}$, *and as such:*

$$\exp(-\alpha_t y_i h_t(x_i)) = \begin{cases} \exp(-\alpha_t) \text{ if } y_i = h_t(x_i) \\ \exp(\alpha_t) \text{ if } y_i \neq h_t(x_i) \end{cases}$$

Substituting terms and expressing the piece wise using the indicator function:

$$\begin{aligned} Z_t &= \sum_{i=1}^{N} D_t(i)\exp(-\alpha)\mathbb{1}(h(x_i) \neq y_i) + \sum_{i=1}^{N} D_t(i)\exp(\alpha)\mathbb{1}(h(x_i) \neq y_i) \\ &= \left(1 - \sum_{i=1}^{N} D_t(i)\mathbb{1}(h(x_i) = y_i)\right)\exp(-\alpha) + \sum_{i=1}^{N} D_t(i)\exp(\alpha)\mathbb{1}(h(x_i) \neq y_i) \\ Z_t &= (1-\epsilon_t)\exp(-\alpha_t) + \epsilon_t \exp(\alpha_t) \end{aligned}$$

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:

$$\begin{aligned} \frac{\partial Z_t}{\alpha_t} &= -(1-\epsilon_t)\exp(-\alpha_t) + \epsilon_t\exp(\alpha) = 0 \\ \epsilon_t\exp(\alpha) &= (1-\epsilon_t)\exp(-\alpha_t) \\ \exp(2\alpha_t) &= \frac{(1-\epsilon_t)}{\epsilon_t} \\ \alpha_t &= \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon}\right) \end{aligned}$$

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 $t \in \{0, \ldots, \texttt{self.n\_clfs}\}$.
- 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 Google Colab notebook for Problem 3.*
The loss curves for the gradient boost is more smooth compared to the AdaBoost loss curves. The training loss curve for gradient boost monotonically decreases as it approaches 0, while the AdaBoost training loss roughly monotonically decreases (it is much noisier) until approaches approximately 0.1. The gradient boost test loss first decreases, reaching a minimum loss of approximately 0.2, before monotonically increasing as it approaches its final value of approximately 0.25. The AdaBoost test loss also first decreases, and then increases slightly, before (roughly) monotonically decreasing until its final value around 0.2.

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 set loss in the gradient boost model is 0 and the final test loss is 0.264. In the AdaBoost model, the final training set loss is 0.1045 and the final test loss is 0.186. While the gradient boost model had a lower training loss, the AdaBoost performed better with the test set. As such, the AdaBoost model performed better at classifying the data set as the validation/test loss was lower.*

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 data set weights are represented by the size of the dots in the animation, which change as we add weak classifiers to our model. Observing this, we see that the size of the dots are smallest (i.e. smallest weights) in areas that are relatively "far" from the blue/red boundary, but largest (i.e. largest weights) close to the boundary. This makes sense given our understanding of the AdaBoost algorithm, which should weigh points that are close*

to the decision boundary higher than points that are far away to generate a better classification of the data points.