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.

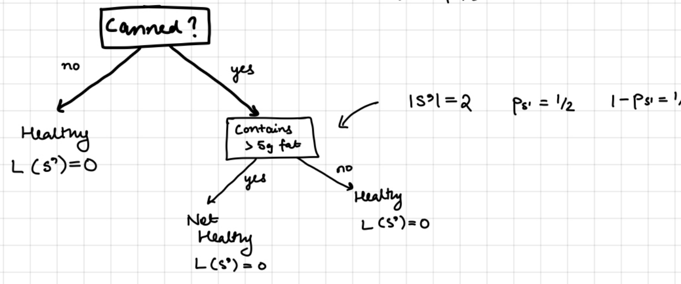
Solution A:

Q1
(a) Top-down greedy induction
– using entropy with natural log.

$$L(S') = \text{Entropy} = -|S'|(p_{S'}\log p_{S'} + (1-p_{S'})\log(1-p_{S'}))$$

To start with, $|S'| = 4$, $p_{S'} = \frac{3}{4}$ $\quad 1-p_{S'} = 1/4$

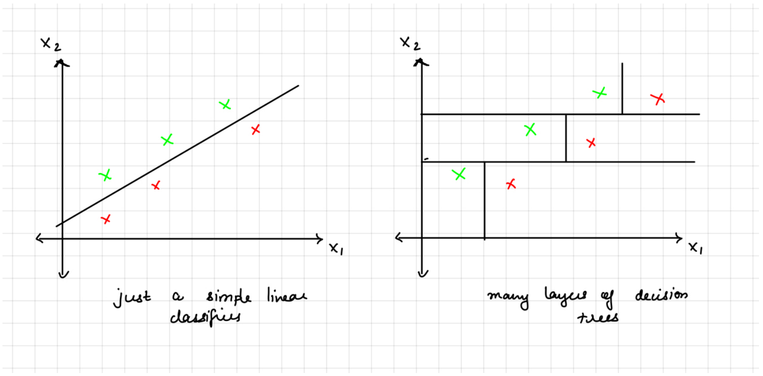
$$L(S') = -4\left(\frac{3}{4}\log\frac{3}{4} + \frac{1}{4}\log\left(\frac{1}{4}\right)\right)$$
$$= 2.24934$$



$|S'|=2$ $\quad p_{S'} = 1/2$ $\quad 1-p_{S'} = 1/2$ $\quad L(S') = -2(\log 1/2)$
$= 1.3863$

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:


x_2
x_1
just a simple linear classifier
x_2
x_1
many layers of decision trees

This dataset can be classified perfectly with a linear classifier (as shown in the image on the left) but, a decision tree would have multiple layers and be overly-complicated.

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

plots/3C.png

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:

Part i)

C(i) Gini loss with no split:

$|S'| = 4$

$p_{s'} = 1/2$

$$L(S') = |S'|\left(1 - p_{S'}^2 - (1-p_{S'})^2\right)$$
$$= 4\left(1 - \frac{1}{4} - \left(1-\frac{1}{2}\right)^2\right)$$
$$= 4\left(1 - \frac{1}{4} - \frac{1}{4}\right) = \frac{4}{2} = 2$$

Splitting either horizontally or vertically results in two pairs of $|S'| = 2$ and $p_s = 1/2$.

$$L(S') = 2\left(1 - \frac{1}{4} - \left(1-\frac{1}{2}\right)^2\right) = 1$$

$$\Sigma L = 1 \times 2 = 2$$

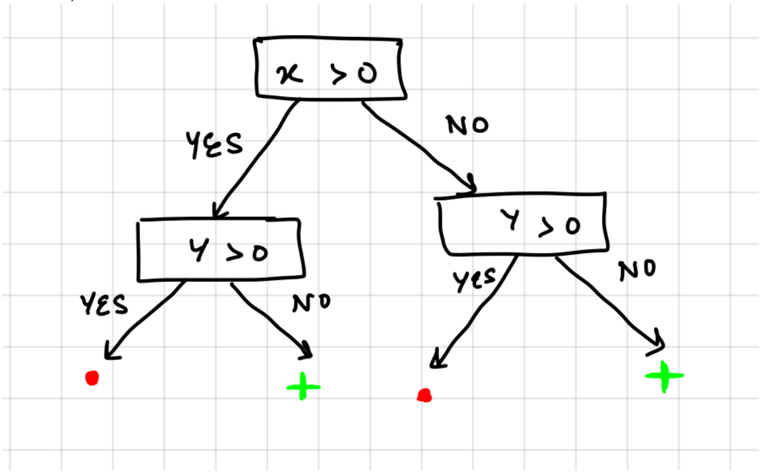
So, there is no way to split such that the loss is reduced.

Resulting tree is just a root node:

$$\boxed{\text{root}}$$

Classification error $= \frac{2}{4} = \frac{1}{2} = 50\%$.

Part ii)



An impurity measure that would produce this tree is one that strongly deters leaves with a large $|S'|$. For example, if we square the $|S'|$ term in the gini impurity, we will have $L(S') = |S'|^2(1 - p_{S'}^2 - (1 - p_{S'})^2)$. So the gini impurity at the root node will be $L(S') = 8$ and after the split, the loss from S1 and S2 would be $2xL(S1) = 2x(2) = 4$. So, the first layer will be made. The second layer will also be made because gini impurity will be 0 after the second layers are made. Resulting in a tree like the one drawn.

The biggest problem with such an index is that it favours very small-sized subsets to be leaves. This could cause overfitting (because there will be too many layers) and bad generalization.

Part iii) The largest number of unique thresholds or internal nodes will be 99. In the worst case, every point has to be separated by a different decision boundary. Each decision boundary separates the area into two parts, so for 100 points, 99 decision boundaries will make a 100 partitions.

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 $O(ND)$. Since there are N-1 data points, for each feature D, there are a maximum of N-1 possible splits. Giving us a worst case of D(N-1) possible splits.*

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.

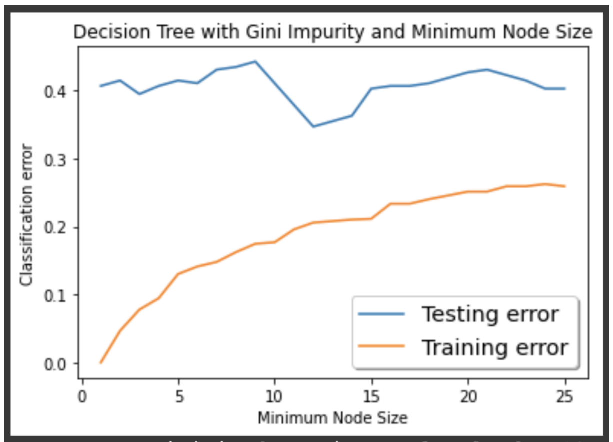
Problem A [10 points]: Choose one of the following from i or ii:

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:

CODE: *https://colab.research.google.com/drive/1k5lP-MjI_WBXHP0d-z10bTLGZebahHvq?usp=sharing*



```
Test error minimized at min_samples_leaf = 12
```

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.

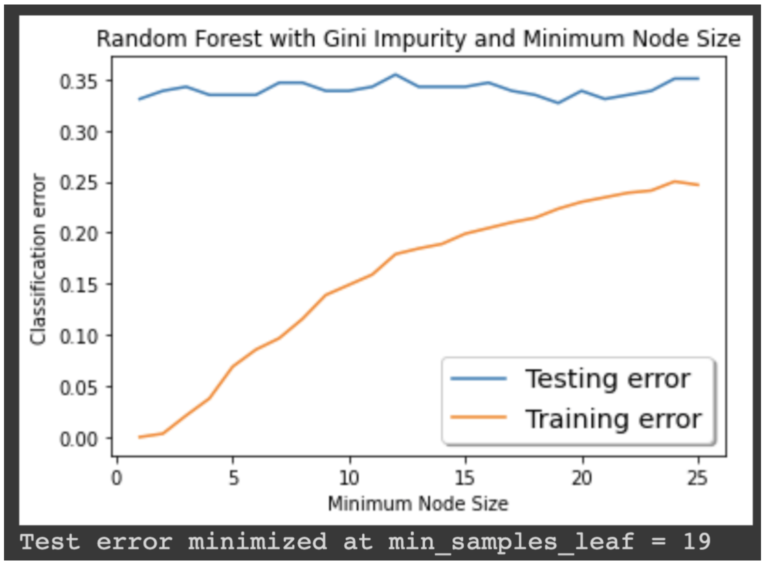
Solution B: *The test error is minimised at min_samples_leaf size 12. Early stopping reduces overfitting. We can see this because as minimum node size increases from 1 to 25, the train error increases steadily but the test error initially reduces (up till 12). This means, from 0 to 12 the model was still overfitting to the training set. After that minimum, we see the test error increase again, this is because the model has began underfitting as a minimum node size* > 12 *is too large to classify properly.*

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.

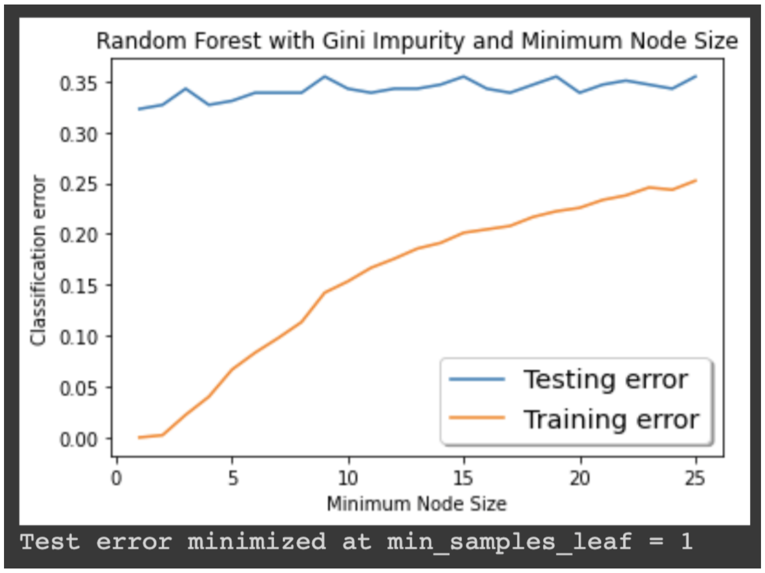
ii. Train a random forest classifier using Gini as the impurity measure, maximal tree depth as early stopping criterion, and 1,000 trees in the forest. Try different tree depths from 2 to 20 in increments of 1. Then, on a single plot, plot both training and test classification error versus tree depth.

Solution C:



```
Test error minimized at min_samples_leaf = 19
```

On changing the random seed, the minimum node size with best test-error performance went as low as 1.



```
Test error minimized at min_samples_leaf = 1
```

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: *The test error is minized when minimum samples in a leaf node is set to 19 with one seed, with another seed the test error is minimized at min_samples_leaf=1. A random forest classifier is much less prone to* *overfitting. The test error seems to almost stay constant as the parameter value changes.*

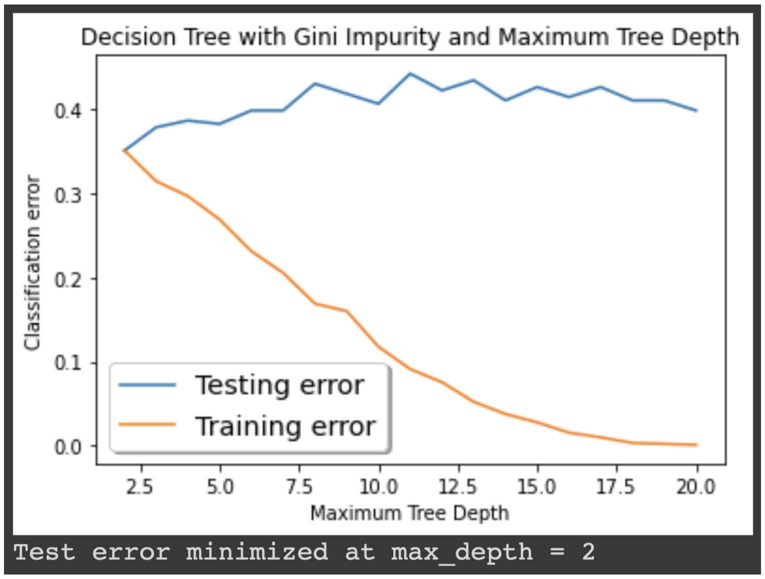
Problem E [4 points]: Do you observe any differences between the curves for the random forest and decision tree plots? If so, explain what could account for these differences.

Solution E: *Yes, the test error curves for random forests and decision trees are very different. The test error for decision trees seems to reduce with an increase in the minimum leaf node size, and benefits greatly from early stopping. In the random forest on the other hand, the train error doesn't seem to change much with early stopping. The decision trees are much more prone to overfitting than the random forest. This is probably because random forest already accounts for variance by random sampling from the feature space and sampling from the dataset and averaging over several decisiont trees. Thus, random forests are less prone to variance and overfitting.*

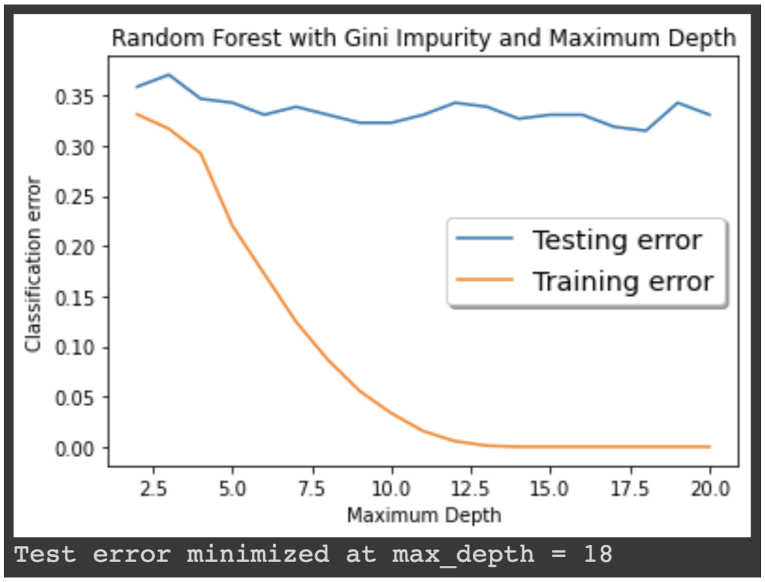
Extra Credit [7 points total] :

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

Solution F:



```
Test error minimized at max_depth = 2
```



```
Test error minimized at max_depth = 18
```

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: *Max_depth 2 minimises test error for the deccision tree while max depth 18 minimizes error for the random forest.*

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: *Let's look at the range of both sides of the equation for a single data point.*
The right side of the equation $H(x_i) \neq y_i$ can be either 0 or 1. Let's look at two cases:
Case 0: $H(x_i = y_i)$, then y_i and $f(x_i)$ have the same sign, so $y_i f(x_i)$ is positive, and the exponent is negative ($exp(-y_i f(x_i))$) and less than 1 but greater than 0. Case 1: $H(x_i \neq y_i)$, then y_i and $f(x_i)$ have the different signs, so $y_i f(x_i)$ is negative, and the exponent is positive ($exp(-y_i f(x_i))$) and greater than 1.

Both cases are true for any datapoint. The average is therefore greater for the left side of the equation than the right and thus the left side forms an upper bound.

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 $t \in \{1, \dots, T\}$. 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:
From lecture we have: $D_{t+1}(i) = \frac{D_t(i) exp(-a_i y_i h_t(x_i))}{Z_t}$
$D_2(i) = \frac{D_1(i) exp(-a_i y_i h_1(x_i))}{Z_1}$ *We know* $D_1(i) = \frac{1}{N}$
So, $D_2(i) = \frac{exp(-a_i y_i h_1(x_i))}{N * Z_1}$
$D_3(i) = \frac{D_1(i) D_2(i) exp(-a_i y_i h_2(x_i))}{Z_2} = \frac{exp(-a_i y_i h_1(x_i)) exp(-a_i y_i h_2(x_i))}{N * Z_2}$

Generalizing this to any T, we have: $D_{T+1} = \Pi_{t=1}^{T} \frac{exp(-a_i y_i h_t(x_i))}{N * Z_t}$

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:

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

and

$$f(x_i) = \Sigma_{t=1}^{T} \alpha_t h_t(x)$$

.

Replacing the value of $f(x_i)$ *and moving* $\frac{1}{N}$ *into the first sum (since it is a constant for any N), we get the desired outcome:*

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

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:

From part B, we have $D_{T+1} = \Pi_{t=1}^{T} \frac{exp(-a_i y_i h_t(x_i))}{N * Z_t} \implies D_{T+1} \Pi_{t=1}^{T} Z_t = \frac{1}{N} \Pi_{t=1}^{T} exp(-a_i y_i h_t(x_i)) = \frac{1}{N} e^{\Sigma_{t=1}^{T} a_i y_i h_t(x_i)}$

From part C, we have $E = \sum_{i=1}^{N} \frac{1}{N} e^{\sum_{t=1}^{T} -\alpha_t y_i h_t(x_i)}$.

So,

$$E = \Sigma_{i=1}^{N} D_{T+1} \Pi_{t=1}^{T} Z_t$$

Since D is a distribution, and $\sum_{i=1}^{N} D_t(i) = 1$, *we have*

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

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:
We know that

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

Let's write this exponential expression using $\mathbb{1}(h_t(x_i) \neq y_i)$

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

So,

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

Since D is a distribution and replacing the definition of ϵ_t *into the equation above, we have our solution 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: *We show this is the optimal choice for* α_t *by taking the derivative of* Z_t *with respect to* α_t.

$$\frac{\delta Z_t}{\delta \alpha_t} = \frac{\delta((1 - \epsilon_t)\exp(-\alpha_t) + \epsilon_t \exp(\alpha_t))}{\delta \alpha_t}$$

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

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

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.

CODE: `https://colab.research.google.com/drive/1z94SvMdbhkp_7wZ7340QcoeDdLtd2MHL?usp=sharing`

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 curve created by gradient boosting is smoother than the curve created by AdaBoost (there's less fluctuation in the curve).*

But, the test error approaches a lower value in AdaBoost than gradient boost.

Gradient boosting is smoother because it uses a regressor and is able to account for smaller values changes every iteration. AdaBoost on the other hand uses a classifier, so the errors fluctuate a lot initially but eventually become smooth.

The final test error in gradient boost is higher than the final test error in AdaBoost because while gradient boost is able to generate several weak regressor, AdaBoost is able to change the weights and prioritize certain datapoints over others, thus making the most of the available data and getting a low test error.

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

Solution I: *The AdaBoost clf has a lower final loss value (0.186) than the GradientBoost clf (0.264), and so it performed better on this dataset.*

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 dataset weights are the largest in magnitude for misclassified points and smallest at the correctly classified points.*