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

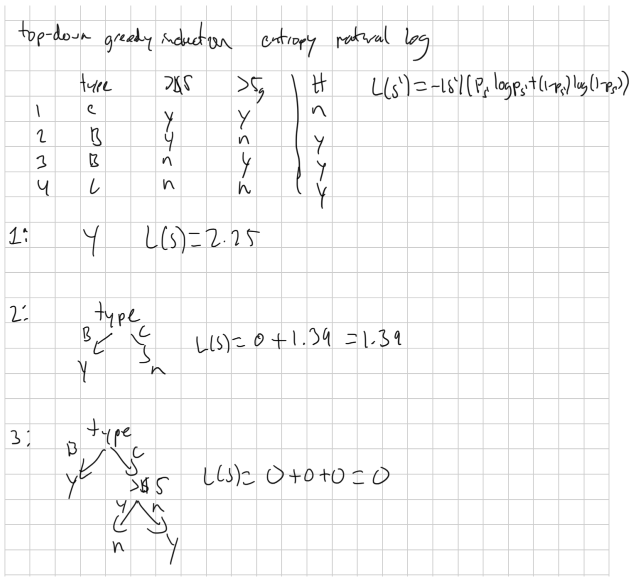
Problem A [7 points]: Consider the following data, where given information about some food you must predict whether it is healthy:

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:



top-down greedy induction entropy natural log

	type	>$5	>5g	H
1	c	y	y	n
2	B	y	n	y
3	B	n	y	y
4	C	n	n	y

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

1: y $L(S) = 2.25$

2:

$$\begin{tikzcd} & \text{type} \arrow[dl, "B"'] \arrow[dr, "C"] & \\ y & & n \end{tikzcd}$$

$L(S) = 0 + 1.39 = 1.39$

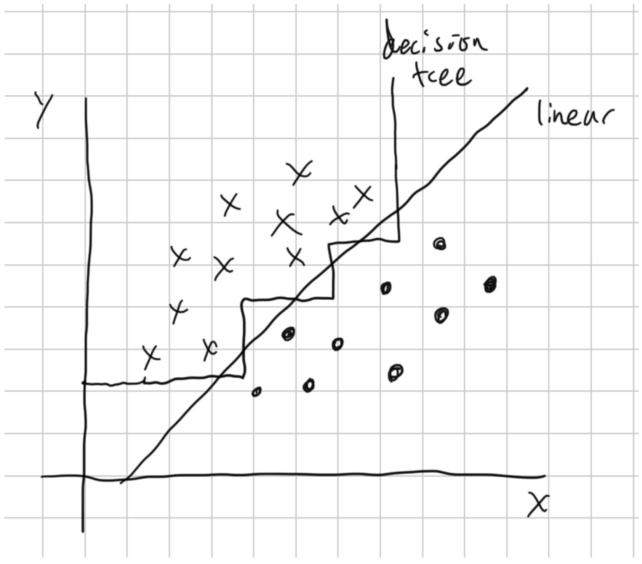
3:

$$\begin{tikzcd} & \text{type} \arrow[dl, "B"'] \arrow[dr, "C"] & & \\ y & & >\$5 \arrow[dl, "y"'] \arrow[dr, "n"] & \\ & n & & y \end{tikzcd}$$

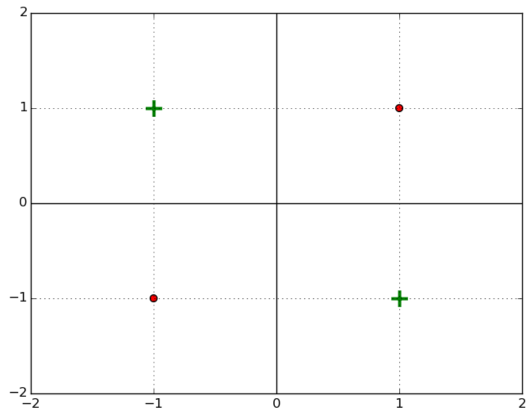
$L(S) = 0 + 0 + 0 = 0$

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: *A decision tree is not always preferred for classification problems; for example, a simple dataset that is classified along a diagonal will require an overly complex decision tree.*



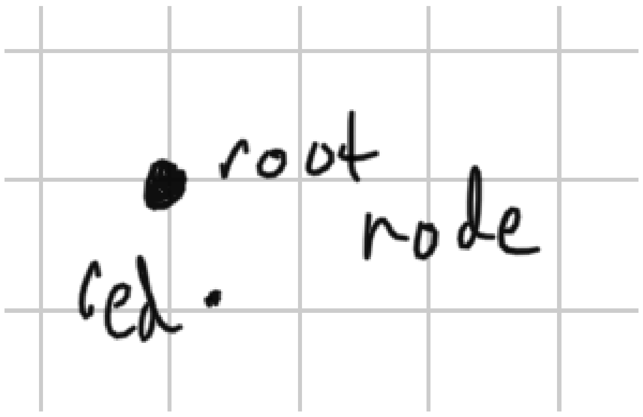
Problem C [15 points]: Consider the following 2D data set:



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))?

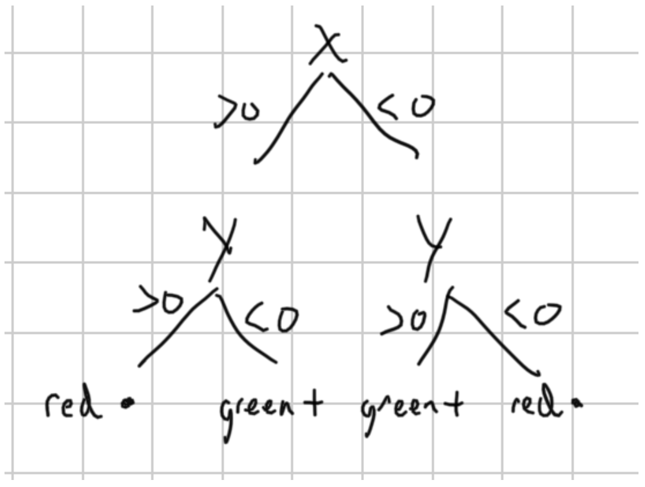
Solution C: *The classification error is 0.5; we get this because any single split of the root node will not result in any reduction in impurity.*



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 create this tree using the same stopping condition, we will use an impurity measure that penalizes larger leaf sizes so we can get around the initial split that doesn't reduce classification error. To do this we will use the Gini Index and change the $|S'|$ term in the Gini Index to a $|S'|^2$. The pro of this is that we are able to go down levels that would have the same impurity to reach better splits later on for a better overall model. The con of this is that we may create an over-complicated model if there aren't better splits later on.

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: *For a 100 data point dataset, the largest number of unique thresholds to reach zero classification training error is 99 internal nodes. If we have a dataset where it isn't possible to split the nodes such that two data points are in the same classification region, then 99 thresholds are required; for example, alternating classifications of data points in a line.*

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: *To split a leaf node that contains N data points composed of D continuous features, we need to check splitting on each of the D possible features and, for each feature, we need to check the N-1 possible ways to split the N data points on that feature. This gives a worst-case complexity of O(ND).*

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.

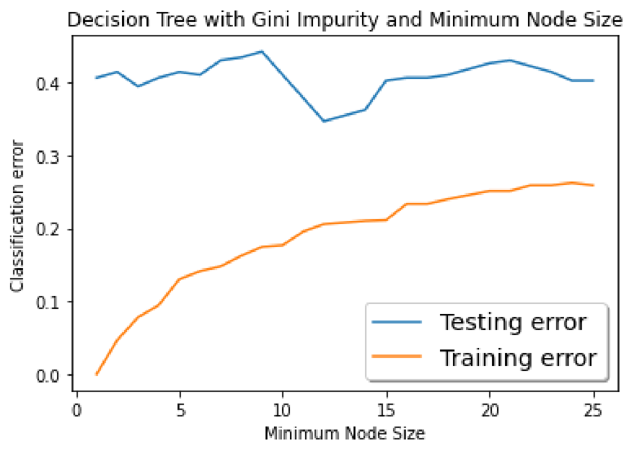
Problem A [10 points]: Choose one of the following from i or ii:

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:

https://colab.research.google.com/drive/1-QP22TAtPfSKu5WkOyb4J4n6D92zgJIc?usp=sharing



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.

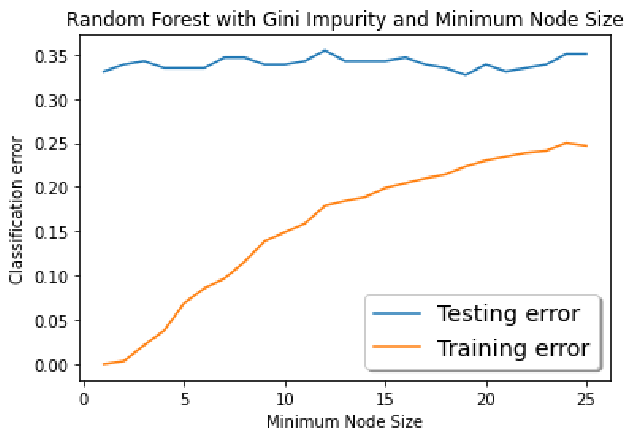
Solution B: *The test error for minimal leaf node size is minimized at 12. Early stopping does seem like it would improve the performance of the decision tree model as having smaller minimum node sizes increases testing error (dip in testing error when we increase minimal leaf node size to 12).*

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:



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 for minimal leaf node size for the random forest is minimized at 19. Early stopping does not seem like it would have much of an effect for the random forest as the testing error seems pretty equal for all node sizes, with the smaller minimum node sizes still having some of the best testing error.*

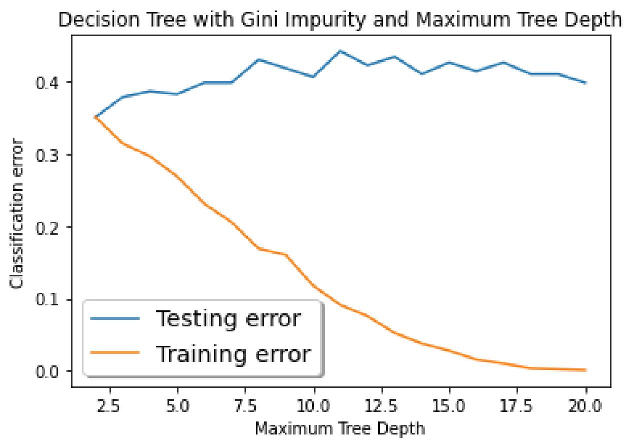
Problem E [4 points]: Do you observe any differences between the curves for the random forest and decision tree plots? If so, explain what could account for these differences.

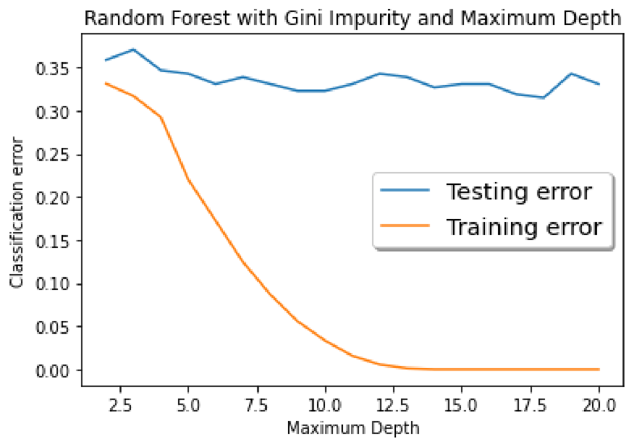
Solution E: *For the decision tree plot, we see higher testing error in general when compared to the random forest plot, with only the dip in the decision tree testing error being at a similar value to the random forest testing error. This is likely because the decision tree will be over-fitting so is benefited by early stopping, while random forests do not over-fit because they sample the data and the features so smaller minimum node sizes don't impact model performance.*

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 test error for maximum tree depth is minimized at 2. Early stopping does seem like it would improve the performance of the decision tree model as the larger depth the model has, the more testing error it has.*

The test error for maximum tree depth is minimized at 18. Early stopping does not seem like it would improve the performance of the random forest model as testing error seems pretty equal for all depths, with the larger maximum depths still having some of the best testing 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: *To prove that* $E = \frac{1}{N}\sum_{i=1}^{N} \exp(-y_i f(x_i)) \geq \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}(H(x_i) \neq y_i)$*, we will show that* $\exp(-y_i f(x_i)) \geq \mathbb{1}(H(x_i) \neq y_i)$ *for all i. We will first consider the case where $\mathbb{1}(H(x_i) \neq y_i) = 1$. Since $\mathbb{1}(H(x_i) \neq y_i) = 1$, we know that $H(x_i) = sign(f(x)) \neq y_i$, so y_i and $f(x_i)$ have opposite signs and we can see that $-y_i f(x_i) \geq 0$. This means that* $\exp(-y_i f(x_i)) \geq \exp(0) = 1 = \mathbb{1}(H(x_i) \neq y_i)$*. In the second case where $\mathbb{1}(H(x_i) \neq y_i) = 0$, we know that $e^x \geq 0\ \forall x$, so we can see that* $\exp(-y_i f(x_i)) \geq 0 = \mathbb{1}(H(x_i) \neq y_i)$*. Since* $\exp(-y_i f(x_i)) \geq \mathbb{1}(H(x_i) \neq y_i)$ *for all i, we know that* $E = \frac{1}{N}\sum_{i=1}^{N} \exp(-y_i f(x_i)) \geq \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}(H(x_i) \neq y_i)$.

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: *From lecture, we have the recursive definition that* $D_{t+1}(i) = \frac{D_t(i)\exp(-\alpha_t y_i h_t(x_i))}{Z_t}$ *and that* $D_1(x) = \frac{1}{N}$*. We can see that by recursively substituting $D_t(i)$ into the equation, we can get to the equation:*

$$D_{T+1}(i) = \frac{\exp(-\alpha_T y_i h_T(x_i))}{Z_T} * \frac{\exp(-\alpha_{T-1} y_i h_{T-1}(x_i))}{Z_{T-1}} * \ldots * \frac{D_1(i)\exp(-\alpha_1 y_i h_1(x_i))}{Z_1}$$

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

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:

$$\begin{aligned}
E &= \frac{1}{N}\sum_{i=1}^{N}\exp(-y_i f(x_i)) && (1)\\
&= \frac{1}{N}\sum_{i=1}^{N}\exp(-y_i\sum_{i=1}^{T}\alpha_t h_t(x_i)) && (2)\\
&= \sum_{i=1}^{N}\frac{1}{N}\exp(\sum_{i=1}^{T}-y_i\alpha_t h_t(x_i)) && (3)\\
&= \sum_{i=1}^{N}\frac{1}{N}e^{\sum_{t=1}^{T}-\alpha_t y_i h_t(x_i)} && (4)
\end{aligned}$$

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:

$$\begin{aligned}
D_{T+1}(i) &= \frac{1}{N}\prod_{t=1}^{T}\frac{\exp(-\alpha_t y_i h_t(x_i))}{Z_t} && (5)\\
D_{T+1}(i) &= \frac{1}{N}\prod_{t=1}^{T}\exp(-\alpha_t y_i h_t(x_i))\prod_{t=1}^{T}\frac{1}{Z_t} && (6)\\
D_{T+1}(i)\prod_{t=1}^{T}Z_t &= \frac{1}{N}\prod_{t=1}^{T}\exp(-\alpha_t y_i h_t(x_i)) && (7)\\
D_{T+1}(i)\prod_{t=1}^{T}Z_t &= \frac{1}{N}\exp(\sum_{t=1}^{T}-\alpha_t y_i h_t(x_i)) && (8)\\
\sum_{i=1}^{N}D_{T+1}(i)\prod_{t=1}^{T}Z_t &= \sum_{i=1}^{N}\frac{1}{N}\exp(\sum_{t=1}^{T}-\alpha_t y_i h_t(x_i)) && (9)\\
\prod_{t=1}^{T}Z_t\sum_{i=1}^{N}D_{T+1}(i) &= E && (10)\\
\prod_{t=1}^{T}Z_t &= E && (11)
\end{aligned}$$

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: *Since y_i and $h_t(x_i)$ both can be -1 or 1, we have that if $y_i = h_t(x_i)$ then $y_i h_t(x_i) = 1$, $\exp(-\alpha_t y_i h_t(x_i)) = \exp(-\alpha_t)$, and $\mathbb{1}(h_t(x_i) \neq y_i) = 0$ so we will want $(1 - \mathbb{1}(h_t(x_i) \neq y_i))\exp(-\alpha_t)$ for this term. By similar logic, we will want $\mathbb{1}(h_t(x_i) \neq y_i)\exp(\alpha_t)$ as if $y_i \neq h_t(x_i)$ then $y_i h_t(x_i) = -1$,*

$\exp(-\alpha_t y_i h_t(x_i)) = \exp(\alpha_t)$, *and* $\mathbb{1}(h_t(x_i) \neq y_i) = 1$.

$$\begin{aligned}
Z_t &= \sum_{i=1}^{N} D_t(i) \exp(-\alpha_t y_i h_t(x_i)) && (12)\\
&= \sum_{i=1}^{N} D_t(i)((1 - \mathbb{1}(h_t(x_i) \neq y_i)) \exp(-\alpha_t) + \mathbb{1}(h_t(x_i) \neq y_i) \exp(\alpha_t)) && (13)\\
&= (\sum_{i=1}^{N} D_t(i) - \sum_{i=1}^{N} D_t(i)\mathbb{1}(h_t(x_i) \neq y_i)) \exp(-\alpha_t) + \sum_{i=1}^{N} D_t(i)\mathbb{1}(h_t(x_i) \neq y_i) \exp(\alpha_t) && (14)\\
&= (1 - \epsilon_t) \exp(-\alpha_t) + \epsilon_t \exp(\alpha_t) && (15)
\end{aligned}$$

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: *We want to minimize by finding when the derivative is equal to 0:*

$$\begin{aligned}
Z_t &= (1 - \epsilon_t) \exp(-\alpha_t) + \epsilon_t \exp(\alpha_t) && (16)\\
\frac{\partial Z_t}{\partial \alpha_t} &= \frac{\partial}{\partial \alpha_t}((1 - \epsilon_t) \exp(-\alpha_t) + \epsilon_t \exp(\alpha_t)) = 0 && (17)\\
0 &= \frac{\partial}{\partial \alpha_t}(1 - \epsilon_t) \exp(-\alpha_t) + \frac{\partial}{\partial \alpha_t}\epsilon_t \exp(\alpha_t) && (18)\\
0 &= -(1 - \epsilon_t) \exp(-\alpha_t) + \epsilon_t \exp(\alpha_t) && (19)\\
0 &= \exp(\alpha_t)(-(1 - \epsilon_t) \exp(-2\alpha_t) + \epsilon_t) && (20)\\
0 &= -(1 - \epsilon_t) \exp(-2\alpha_t) + \epsilon_t && (21)\\
\exp(-2\alpha_t) &= \frac{\epsilon_t}{1 - \epsilon_t} && (22)\\
\alpha_t &= -\frac{1}{2} \ln(\frac{\epsilon_t}{1 - \epsilon_t}) && (23)\\
\alpha_t &= \frac{1}{2} \ln(\frac{1 - \epsilon_t}{\epsilon_t}) && (24)
\end{aligned}$$

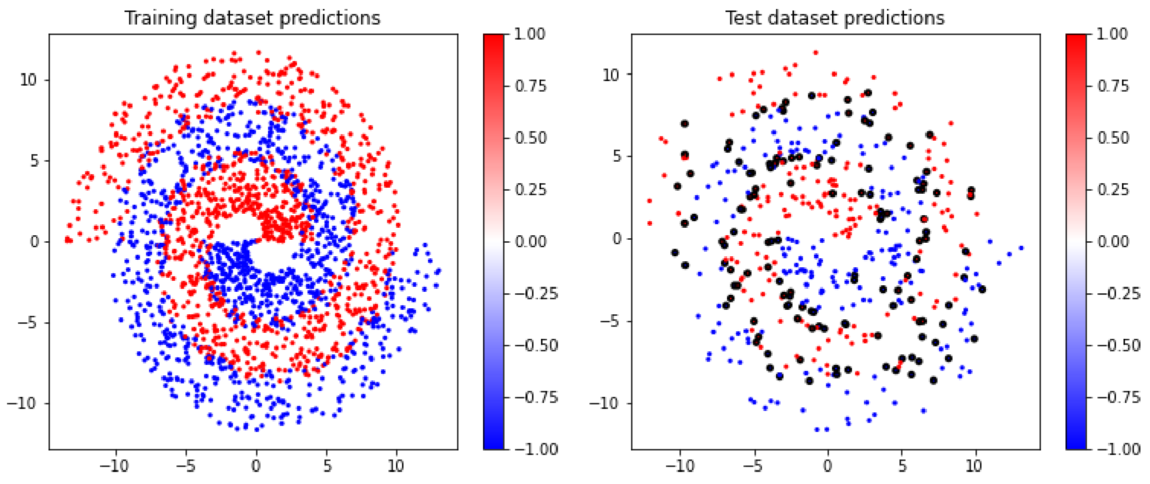
Problem G [14 points]: Implement the `GradientBoosting.fit()` and `AdaBoost.fit()` methods in the notebook provided for you. Some important notes and guidelines follow:

- 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 $t \in \{0, \ldots,$ `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.

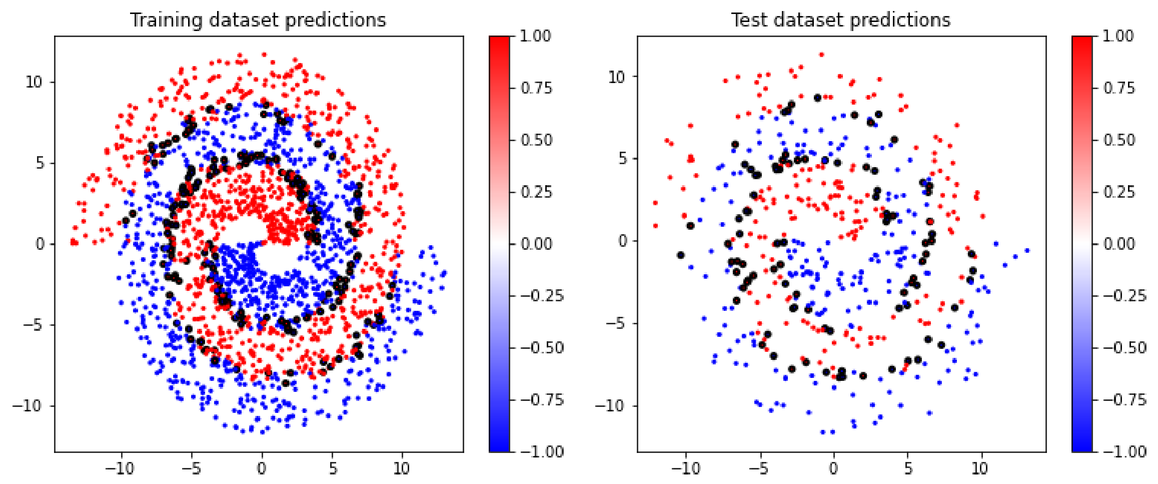
Solution G:

https://colab.research.google.com/drive/1DXqguiT4BwR2vn4eFAN1hZUISVY7WC6h?usp=sharing

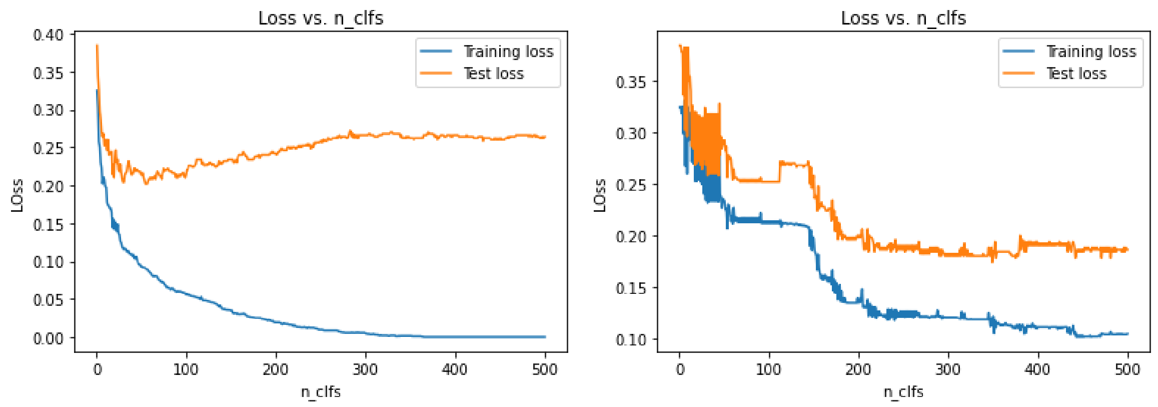
Gradient Boosting Training (left) and Testing (right) predictions:



Ada Boosting Training (left) and Testing (right) predictions:



Gradient Boosting (left) and Ada Boosting (right) Loss



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 boosting loss curves in general seem smoother than the AdaBoost loss curves; the classifications cause larger jumps, while the regressors smooth out the curve. As the number of classifiers/regressors increases for both, the curves in general seem to get smoother; with few classifiers/regressors, they have a larger impact. The gradient boosting training loss approaches a lower value than the AdaBoost training loss, but the gradient boosting testing loss approaches a higher value (after dipping down first) than the AdaBoost testing loss. Gradient boosting has better training loss because it directly reduces the distance, while AdaBoost* *has better testing loss because it focuses on correcting misclassifications.*

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

Solution I: *From the final loss values, we can see that the AdaBoost model with a final testing loss of 0.186 performed better than the gradient descent model with a final testing loss of 0.264.*

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: *From the animation, it seems that the dataset weights are strongest for the misclassified points as those are the ones that have the biggest change during the animation.*