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: Starting at the root node, we have $|S| = 4$ and $p_S = 0.75$, thus the entropy is 2.25.

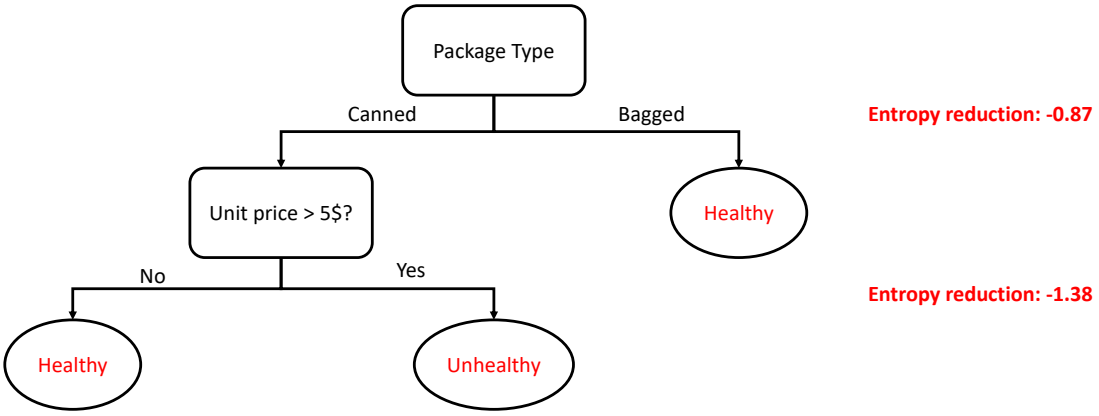
At the first node, we find that all three splits have the same entropy:

- Package type: Canned = {no,yes}, Bagged = {yes,yes}; entropy = 1.38
- Unit Price > \$5: Yes = {no,yes}, No = {yes,yes}; entropy = 1.38
- Contains > 5 grams of fat: Yes = {no,yes}, No = {yes,yes}; entropy = 1.38

As such, there is no difference between which split we choose. Randomly selecting package type, we get, for canned:

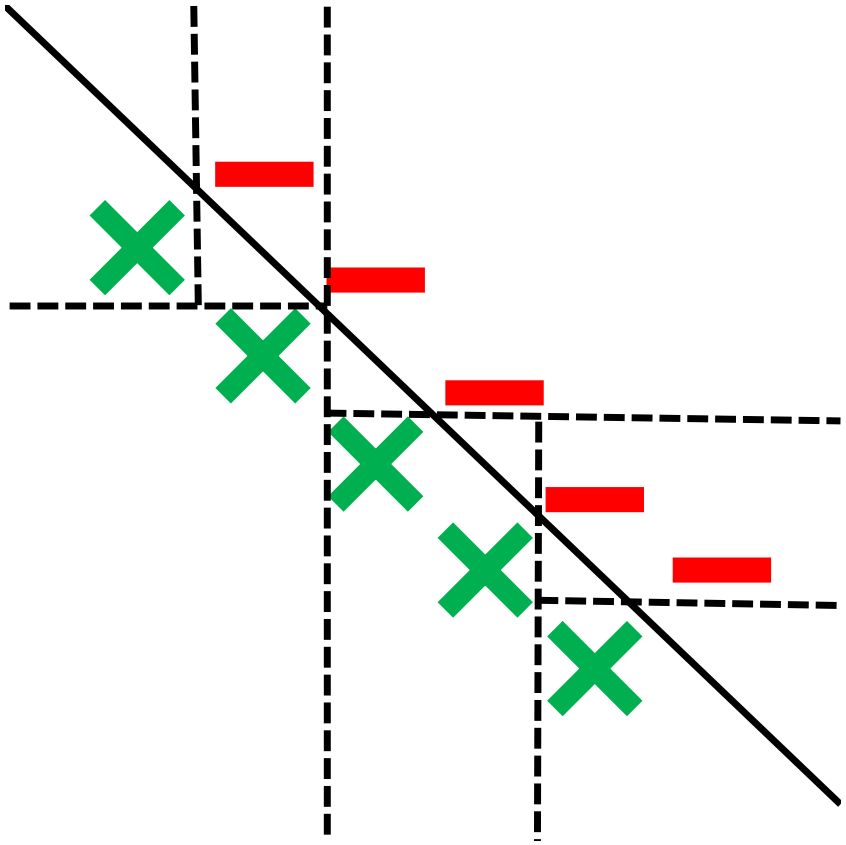
- Unit price > \$5: Yes = {no}, No = {yes}; entropy = 0.00
- Contains > 5 grams of fat: Yes = {no}, No = {yes}; entropy = 0.00

Again, both entropies are identical and give us an impurity of 0.00, as such, by picking either of the above, we have reached our stopping criteria, giving the following tree with their respective reductions:



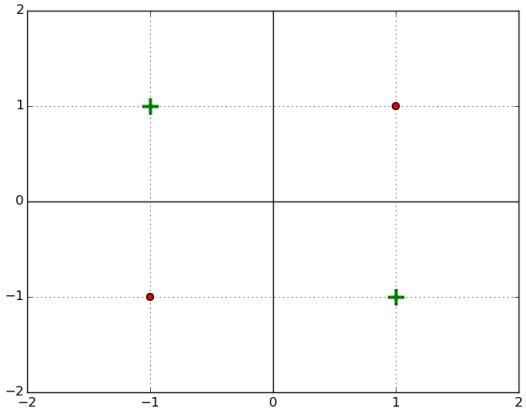
Problem B [4 points]: Compared to a linear classifier, is a decision tree always preferred for classification problems? If not, draw a simple 2-D dataset that can be perfectly classified by a simple linear classifier but which requires an overly complex decision tree to perfectly classify.

Solution B: The tree classifier model is not always preferrable to the linear classifier. An example of this is shown below:



As we can see above, there are many wasted boundaries.

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))?

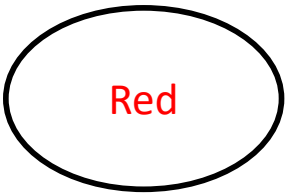
ii. [5 points]: Submit a drawing of a two-level decision tree that classifies the above dataset with zero classification error. (You don't need to use any particular training algorithm to produce the tree.)

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).

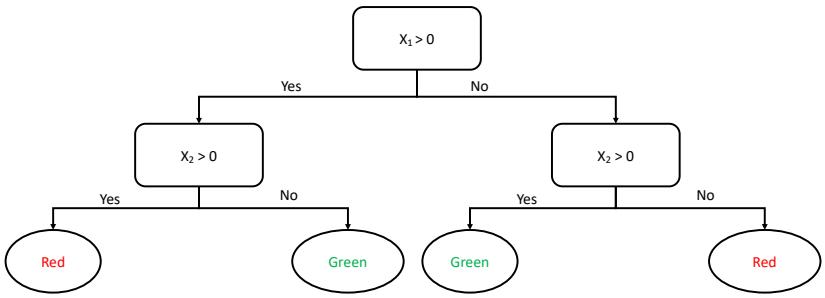
iii. [5 points]: Suppose there are 100 data points in some 2-D dataset. What is the largest number of unique thresholds (i.e., internal nodes) you might need in order to achieve zero classification training error (on the training set)? Please justify your answer.

Solution C:

- Trying to split this data set, we find that all linear, single-level splits will result in the same impurity of 2.0. As such, the classification error will be (2/4)=0.5. The true is simply:



- The following tree will result in a classification error of 0:



  The following impurity measure would lead to a top-down greedy induction with the same stopping conduction to produce the above tree:

$$\text{impurity} = \frac{\#\text{pos} \times \#\text{neg}}{\sqrt{|S'|}} \tag{1}$$

  Pros: This effectively ensures that, if the fraction of correct classifications is the same before and after the split, we will split.

  Cons: This preference towards splitting may lead to unnecessary splits which can lead to overfitting. For example, in the case where the fraction of correct classifications is the same before and after the split, this split may actually result in worse generalisability of the tree.

- If we imagine a case where the points are alternating between classifiers along both of the axis, the number of splits required to give a 0 training error will be however many data points - 1; this is the worst-case scenario for classifying a data set. As such, for a data set of 100 points, the maximum number of splits required with be 99 splits.

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: In the worst-case complexity scenario, where the data set can only be split into the specific coordinates (or 'cells') of the data points, we will need $\mathcal{O}(ND)$ splits. The best way to illustrate this is, like in the 2D example above, if the data points are alternating in all dimensions D, we will need $\mathcal{O}(ND)$ 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.

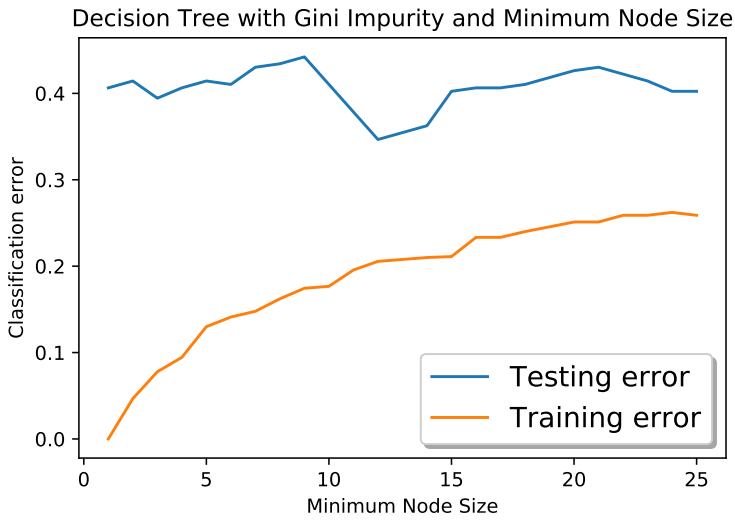
Problem A [10 points]: Choose one of the following from i or ii:

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:

i. The following figure was generated:



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.

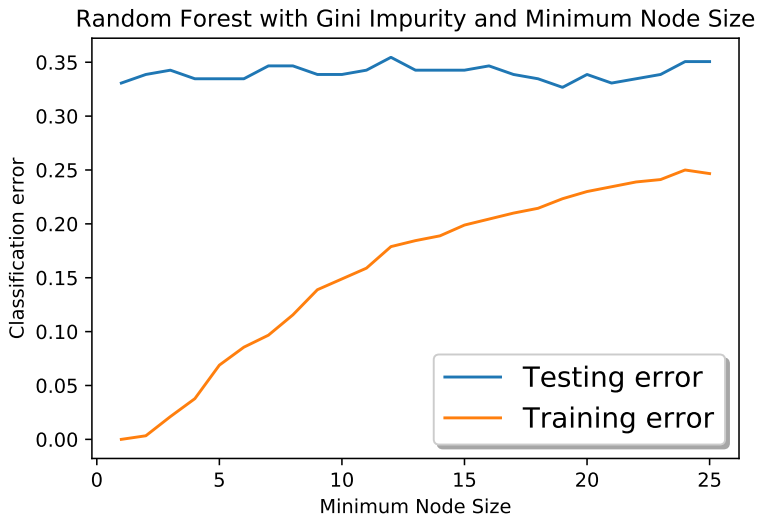
Solution B: In this case, the test error is minimised at a minimum node size of 12. We will have early stopping here at larger minimum node sizes; as such, in decreasing the minimum node size, we will begin to observe overfitting. This is illustrated in the figure with the increasing bias between the testing and training error. Once we introduce early stopping at greater minimum nodes sizes, where we have less splitting, the bias decreases, implying our model will generalise better.

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: Link: https://colab.research.google.com/drive/1UVimdtI8v23hzEZKEsAVK-flVlzsvp8Q?usp=sharing i. The following figure was generated:



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 minimal leaf node size, we find that a value of 19 minimises the test error. In contrast to the decision tree, the random forest isn't as significantly affected by the early stopping (here indicated by the larger minimal leaf node size). However, like the decision tree, there is overfitting occuring at smaller minimum node size as illustrated by the larger bias. Similarly, with earlier stopping, the bias is reduced, implying the model will generalise better.

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 impact of the minimal leaf node size is less pronounced in the case of the random forest plot, where our minimum test error is as significantly different compared to the rest of the plot. The magnitude of the testing error is also smaller in the case of the random forest, despite the training error being very similar between the two.

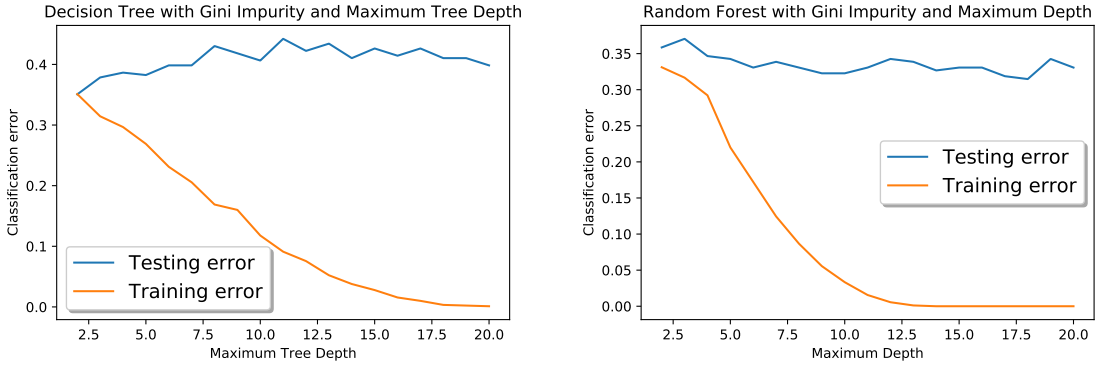
The reason for this can be accounted for by the fact that the random forest is an ensemble of decision

trees where we have generated bootstraps from the original training set and average the results. This will reduce variance in our predictions, as shown above.

Extra Credit [7 points total] :

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

Solution F:



(a) (b)

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 error is minimised at a max depth of 2 and 18, respectively.

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: Given all terms in the above sums are positive, all we need to prove is:

$$\exp\left(-y_i f(x_i)\right) \geq \mathbb{1}(H(x_i) \neq y_i) \tag{2}$$

For the case where y_i and $f(x_i)$ have different signs, we have:

$$\exp\left(-y_i f(x_i)\right) \geq 1 \tag{3}$$

and:

$$\mathbb{1}(H(x_i) \neq y_i) = 1 \tag{4}$$

Similarly, for y_i and $f(x_i)$ being the same sign, we have:

$$\exp\left(-y_i f(x_i)\right) \geq 0 \tag{5}$$

and:

$$\mathbb{1}(H(x_i) \neq y_i) = 0 \tag{6}$$

As such, we find:

$$\exp\left(-y_i f(x_i)\right) \geq \mathbb{1}(H(x_i) \neq y_i) \tag{7}$$

meaning:

$$E = \frac{1}{N}\sum_{i=1}^{N}\exp(-y_i f(x_i)) \geq \frac{1}{N}\sum_{i=1}^{N}\mathbb{1}(H(x_i) \neq y_i) \tag{8}$$

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 lectures, we know:

$$D_{t+1}(i) = \frac{D_t(i)\exp\left(-\alpha_t y_i h_t(x_i)\right)}{Z_t} \tag{9}$$

This is a recursive relationship which, when fully expanded, gives:

$$D_t(i) = D_1(i)\prod_{j}^{t-1}\frac{\exp\left(-\alpha_j y_i h_j(x_i)\right)}{Z_j} \tag{10}$$

Thus:

$$D_{T+1}(i) = D_1(i)\prod_{j}^{T-1}\frac{\exp\left(-\alpha_j y_i h_j(x_i)\right)}{Z_j} \tag{11}$$

Substituting $D_1(i) = \frac{1}{N}$ gives:

$$D_{T+1}(i) = \frac{\prod_{j}^{T-1}\frac{\exp\left(-\alpha_j y_i h_j(x_i)\right)}{Z_j}}{N} \tag{12}$$

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

Solution C: Given that:

$$f(x_i) = \sum_{j}\alpha_j h_j(x_i) \tag{13}$$

we obtain:

$$E = \sum_{i=1}^{N}\frac{1}{N}e^{\sum_{t=1}^{T}-\alpha_t y_i h_t(x_i)} \tag{14}$$

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: Substituting our definition of $D_t(i)$ into our expression for Z_t gives:

$$Z_t = \frac{1}{N} \sum_{i=1}^{N} \left(\prod_{j}^{t-1} \frac{\exp\left(-\alpha_j y_i h_j(x_i)\right)}{Z_j} \right) \exp(-\alpha_t y_i h_t(x_i)) \tag{15}$$

$$Z_t = \frac{1}{N} \sum_{i=1}^{N} \left(\prod_{j}^{t-1} \frac{1}{Z_j} \right) \left(\prod_{j}^{t} \exp\left(-\alpha_j y_i h_j(x_i)\right) \right) \tag{16}$$

$$Z_t = \frac{1}{N} \sum_{i=1}^{N} \left(\prod_{j}^{t-1} \frac{1}{Z_j} \right) \exp \sum_{j}^{t} \left(-\alpha_j y_i h_j(x_i)\right) \tag{17}$$

Thus:

$$\prod_{j}^{t} Z_j = \frac{1}{N} \sum_{i=1}^{N} \exp \sum_{j}^{t} \left(-\alpha_j y_i h_j(x_i)\right) \tag{18}$$

As we can see, we have:

$$E = \prod_{j}^{T} Z_j \tag{19}$$

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: Starting from:

$$Z_t = \sum_{i=1}^{N} D_t(i) \exp(-\alpha_t y_i h_t(x_i)) \tag{20}$$

Taking the case where $h_t(x_i) \neq y_i$, we have:

$$Z_t = \sum_{i=1}^{N} D_t(i) \exp(\alpha_t) \tag{21}$$

By the nature of distribution functions, this gives:

$$Z_t = \exp(\alpha_t) \tag{22}$$

From the definition of ϵ_t, if $h_t(x_i) \neq y_i$, we have:

$$\epsilon_t = \sum_{i=1}^{N} D_t(i) = 1 \tag{23}$$

Thus:

$$Z_t = (0) \exp(-\alpha_t) + \exp(\alpha_t) = \exp(\alpha_t) \tag{24}$$

Inversly, for $h_t(x_i) = y_i$, we have:

$$Z_t = \sum_{i=1}^{N} D_t(i) \exp(-\alpha_t) = \exp(-\alpha_t) \tag{25}$$

And, given $\epsilon_t = 0$:

$$Z_t = \exp(-\alpha_t) + (0) \exp(\alpha_t) = \exp(-\alpha_t) \tag{26}$$

Thus, both cases are satisfied.

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: Taking the derivative w.r.t. α_t gives:

$$\frac{\partial Z_t}{\partial \alpha_t} = -(1 - \epsilon_t) \exp(-\alpha_t) + \epsilon_t \exp(\alpha_t) = 0 \tag{27}$$

$$\frac{(1 - \epsilon_t)}{\epsilon_t} = \exp(2\alpha_t) \tag{28}$$

$$\alpha_t^* = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right) \tag{29}$$

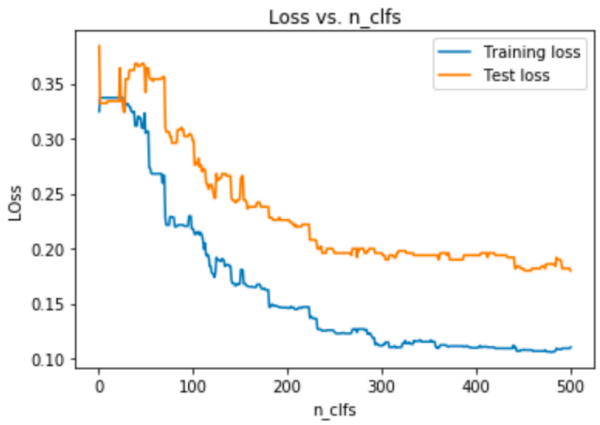
Problem G [14 points]: Link: https://colab.research.google.com/drive/1nVX72Sv4XCMkEagQP8hl0KE8AT2Pww9Y?usp=sharing Implement the GradientBoosting.fit() and AdaBoost.fit() methods in the notebook provided for you. Some important notes and guidelines follow:

- 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: The following figures were generated:

- For Gradient Boost:

- For Adaboost:



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: In terms of smoothness, gradient boost is smoother than adaboost; this is primarily because the adaboost method will be jumping between weights as we correctly or incorrectly classify data points, primarily through the definition of our loss function. In gradient boosting, all equations are continuous, resulting in a smooth loss.

In terms of final errors, because gradient boosting is weighing all datapoints equally, it eventually reaches an error of 0. However, in the case of adaboosting, because we are not fitting all the points equally, the error is non-zero even for large numbers of classifiers. However, the testing error for gradient boosting is greater than adaboosting, highlighting that the former is overfitting, primarily because it is fitting all data points equally.

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

Solution I: In terms of classification, Adaboost performed better. We can see they by considering that, not only was the final loss smaller than gradient boosting, but, in comparison to the training loss, adaboost had a smaller difference, indicative of a smaller bias, implying that the model generated will generalise 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: The data points in the center of the spiral have the smallest weights, primarily because they are already fitted correctly, whilst the points at the borders of the spirals have the largest weights, given these are the points we are most focused on fitting.