Machine Learning & Data Mining
Derek Ing 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

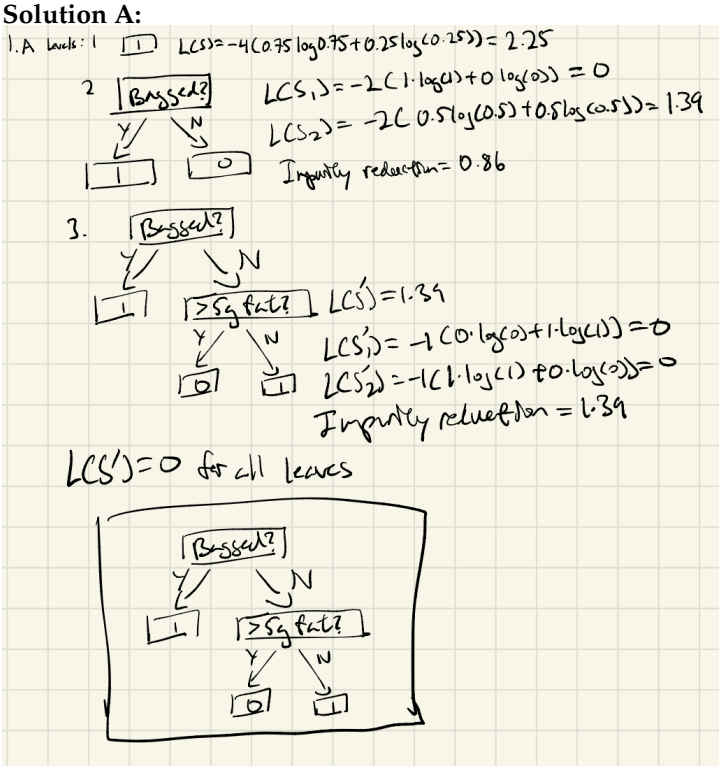
Problem A [7 points]: Consider the following data, where given information about some food you must predict whether it is healthy:

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.

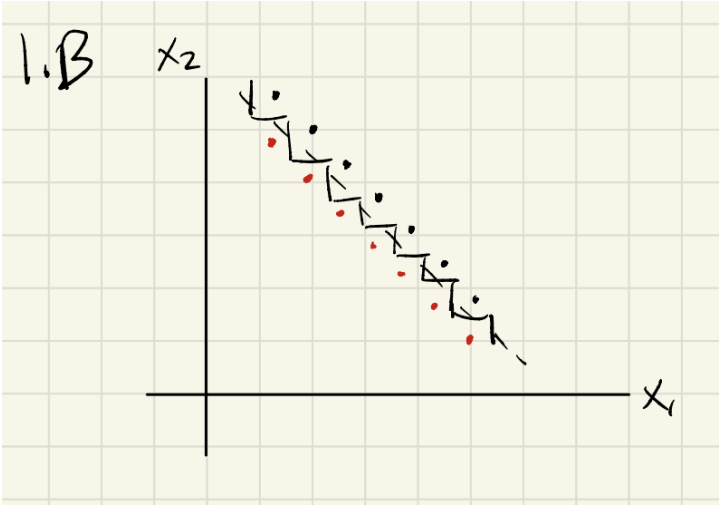
Submit a drawing of your tree showing the impurity reduction yielded by each split (including root) in your decision tree.

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: *No, decision tree is not always preferred for classification problems. Consider the example:*



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))?

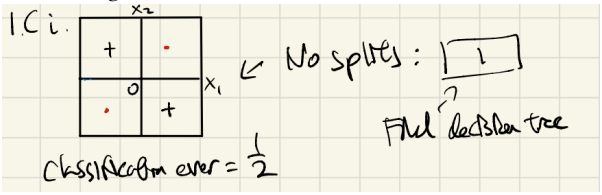
ii. [5 points]: Submit a drawing of a two-level decision tree that classifies the above dataset with zero classification error. (You don't need to use any particular training algorithm to produce the tree.)

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.

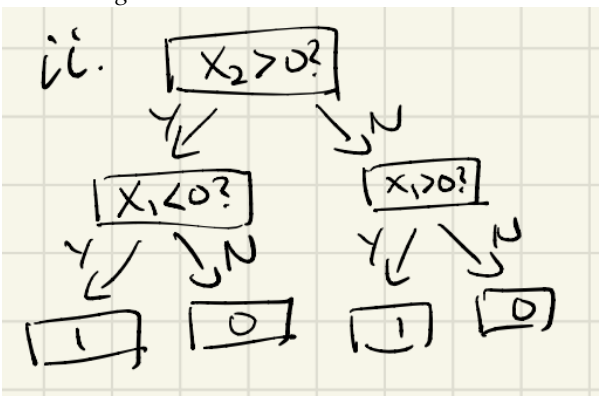
Solution C:

i. Drawing:



No splits because there will be no reduction in impurity. Classification error $= \frac{1}{2}$

ii. Drawing:



Yes, for example $L(S') = |S'|^2(p_{s'})(1 - p_{s'})$.

Pros: This impurity measure would have a non-zero impurity reduction even if the parent and children nodes have the same fraction of positive examples. As a result, the example above with the same stopping condition would have led to the two-level decision tree with zero classification error.

Cons: This impurity measure could sometimes lead a model to overfit because even if the classification error is

the same, the model will still be encouraged to make a split because impurity would be reduced. As a result, there could be too many splits, thus causing overfitting. Sometimes, we may only want to split if impurity is reduced by a certain percentage and not only if there is some non-zero impurity reduction.

iii. The largest number of unique thresholds is 99 in order to achieve zero classification training error with 100 data points. We see that with 4 data points in the example given, we are only able to reduce classification error to 0 with 3 splits. If we add another data point to make 5 points, it is either correctly classified or it is incorrectly classified and we need another split. Thus, the maximum would be 4 splits with 5 data points. We can see a pattern where if we are given N data points, we need at most $N - 1$ splits to achieve zero classification error.

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: *Worst-case complexity = $O(ND)$. The worst case would be to check each possible split that would result in unique child sets. We know that the number of boundaries that will result in unique child sets is $N * D$, which is the number of 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

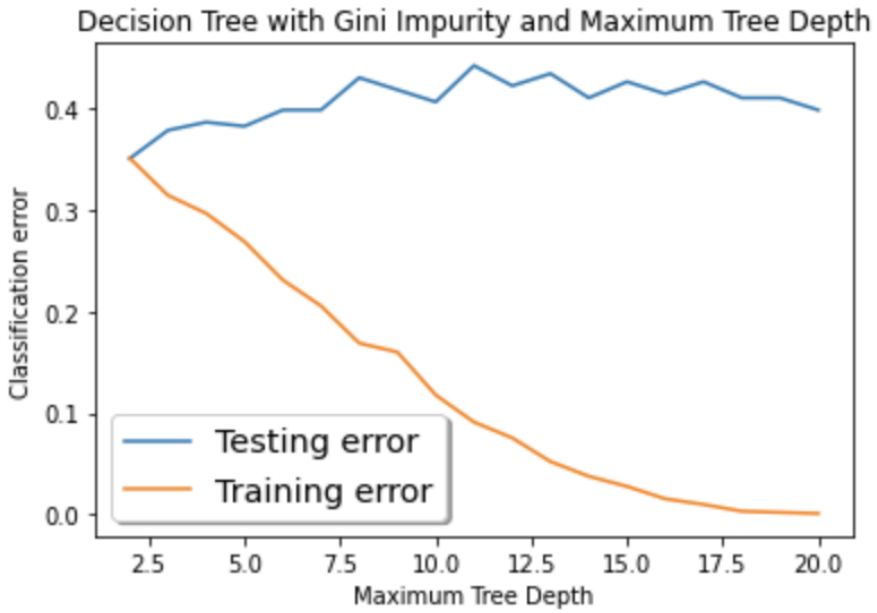
In the following question, your goal is to predict the diagnosis of diabetic retinopathy, which is the final column in the data matrix. Use the first 900 rows as training data, and the last 251 rows as validation data. Please feel free to use additional packages such as Scikit-Learn. Include your code 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:
ii:



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: *A maximum depth of 2 nodes minimizes the test error. Early stopping on a decision tree model improves the performance of a decision tree model by preventing overfitting. A greater maximum depth means that there can be more splits in the model, as more splits means that the tree will be deeper. We see from the plot as maximum tree depth increases, training error decreases. The maximum test error is at depth* $= 2$*. However, test error increases as maximum depth increases. Thus, the model is overfitting more as maximum depth decreases. Thus, early stopping would prevent the model from splitting too much and thus overfitting, which improves test error, as seen from the plot.*

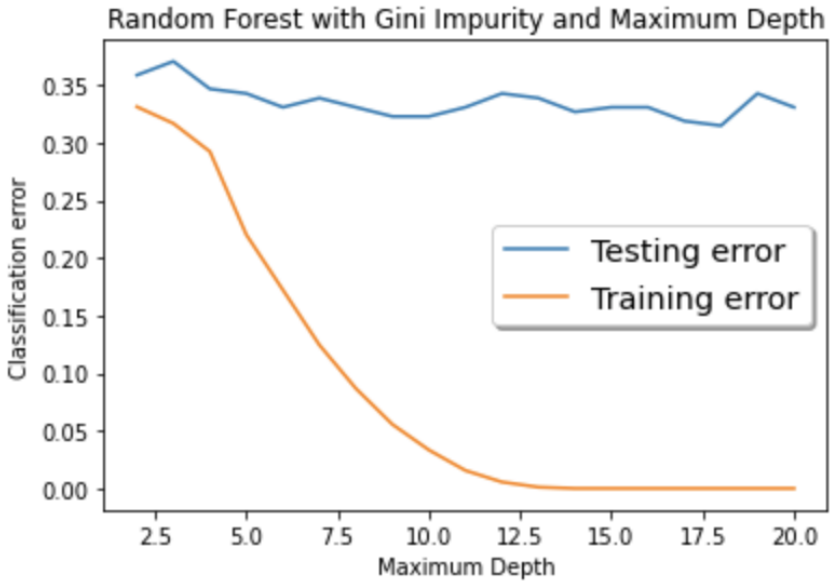
Problem C [4 points]: Choose one of the following from i or ii:

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:
ii:



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: *Maximum depth* $= 18$ *minimizes random forest test error. Early stopping does improve the performance of the random forest model. Training error and test error are both high at low maximum tree depth, which suggests underfitting. As maximum depth increases, training error decreases to 0. However, test error also decreases as maximum depth increases and reaches a minimum at maximum depth* $= 18$*. However, as maximum depth increases more, test error begins to increase, and the model begins to overfit.*

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: *One difference between the random forest and decision tree plots is the test error decreases as maximum depth increases until maximum depth* $= 18$ *for the random forest, but the test error increases for decision tree. Random forests resample training data as well as randomly sampling features at each split. The randomness decorrelates the trees in the forest and reduces variance, thus preventing overfitting. As seen in the random forest plot, test error decreases for greater maximum depths. This suggests the model is improving.*

Thus, the random forest allows for less strict early stopping conditions. However, the test error increases for all maximum depths for the decision tree, so the model is overfitting. Strict early stopping conditions will prevent the decision tree from overfitting.

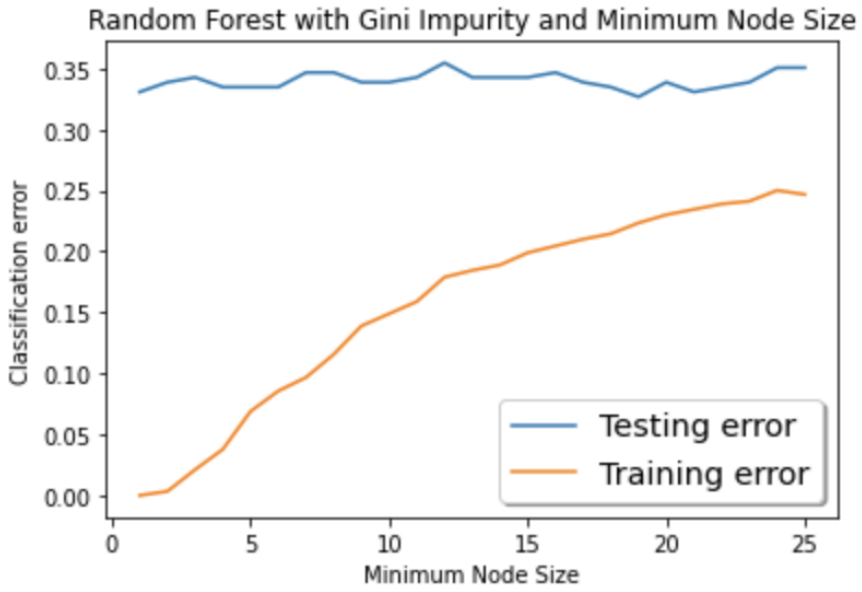
Extra Credit [7 points total] :

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

Solution F:

A i:

C i:



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: *Minimum node size = 12 minimizes decision tree test error. Minimum node size = 19 minimizes random forest 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

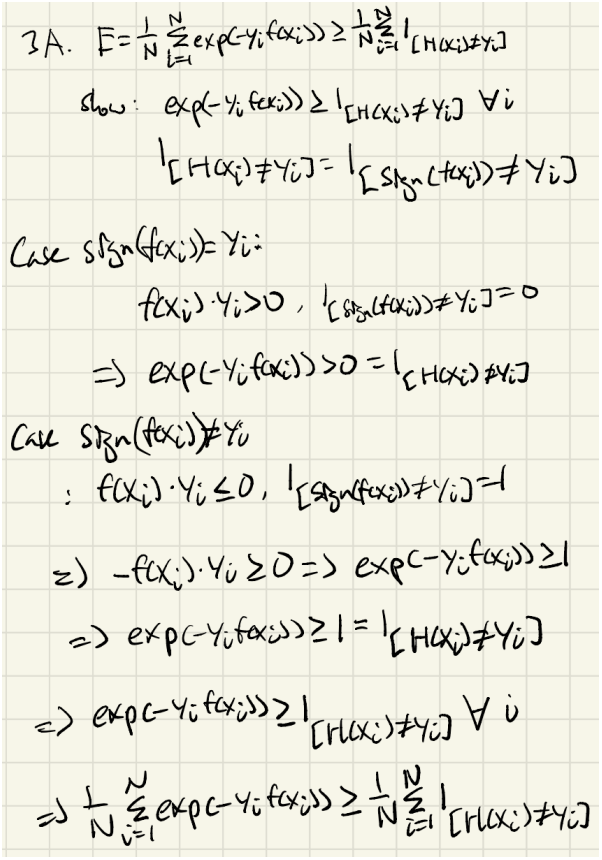
$$H(x) = \text{sign}(f(x)) = \text{sign}\left(\sum_{i=1}^{T} \alpha_t h_t(x)\right).$$

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.

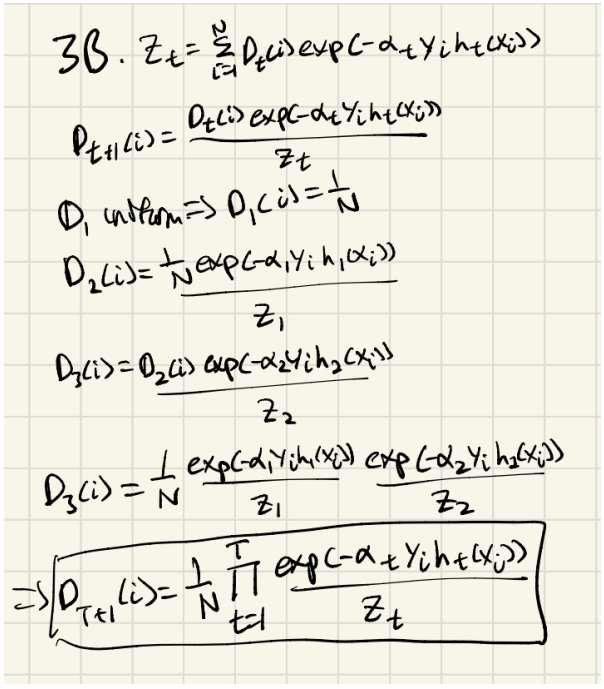
Solution A:



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}:

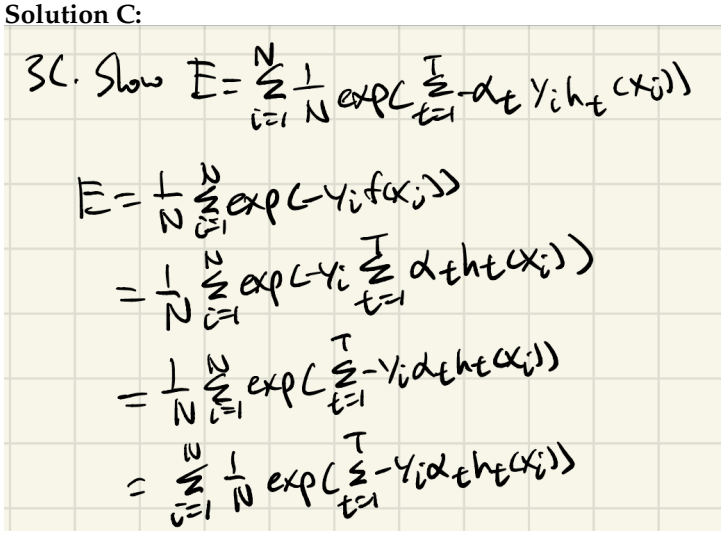
$$Z_t = \sum_{i=1}^{N} D_t(i) \exp(-\alpha_t y_i h_t(x_i)).$$

Solution B:



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:

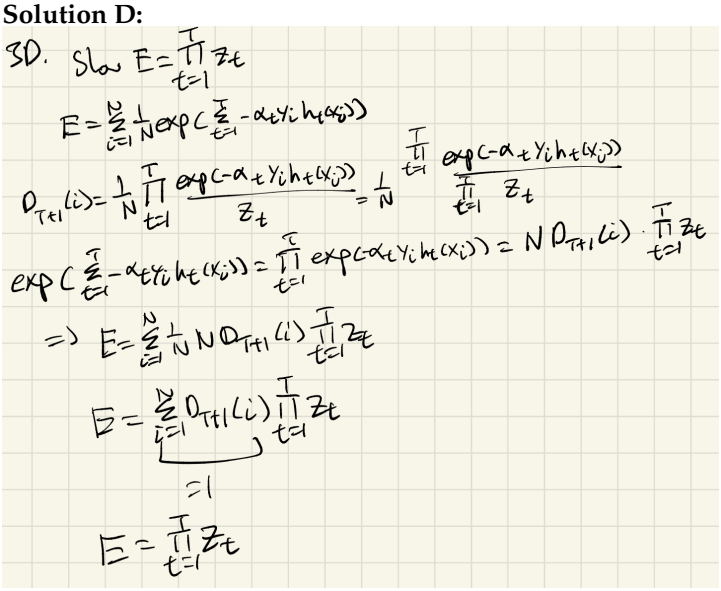


Problem D [5 points]: Show that

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

Hint: *Recall that* $\sum_{i=1}^{N} D_t(i) = 1$ *because* D *is a distribution.*

Solution D:



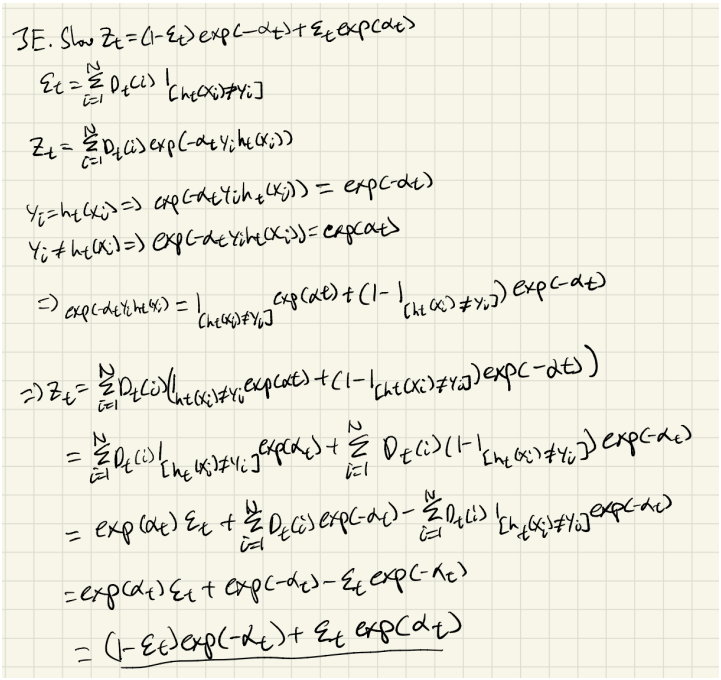
Problem E [5 points]: Show that the normalizer Z_t can be written as

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

where ϵ_t is the training set error of weak classifier h_t for the weighted dataset:

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

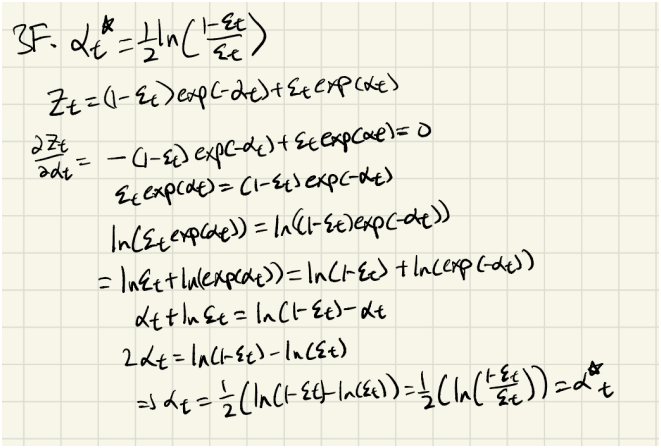
Solution E:



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:

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

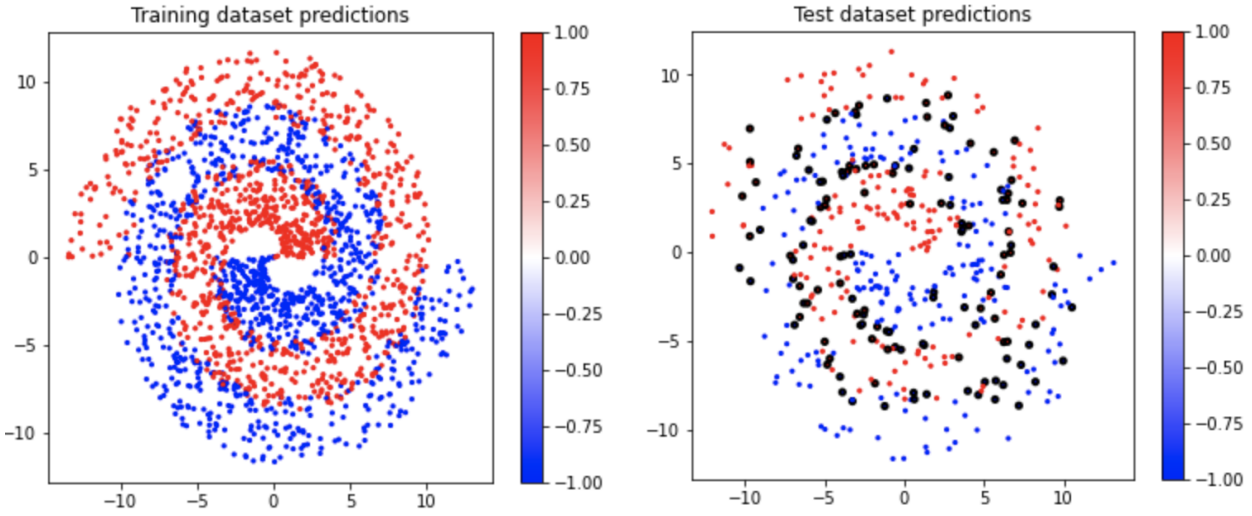
Solution F:



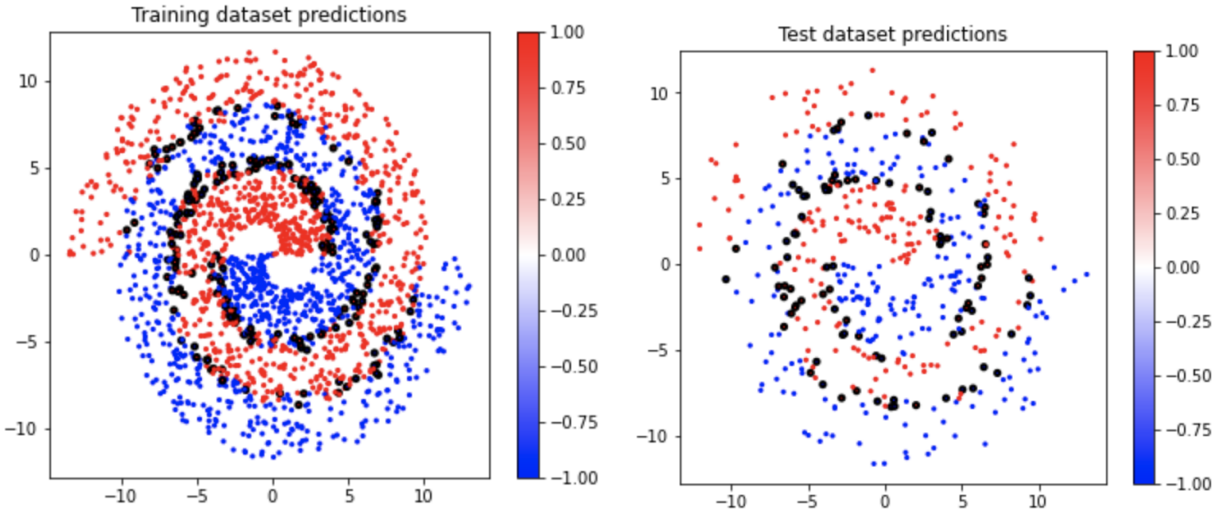
Problem G [14 points]: Implement the `GradientBoosting.fit()` and `AdaBoost.fit()` methods in the notebook provided for you. Some important notes and guidelines follow:

- 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.

Gradient Boosting:



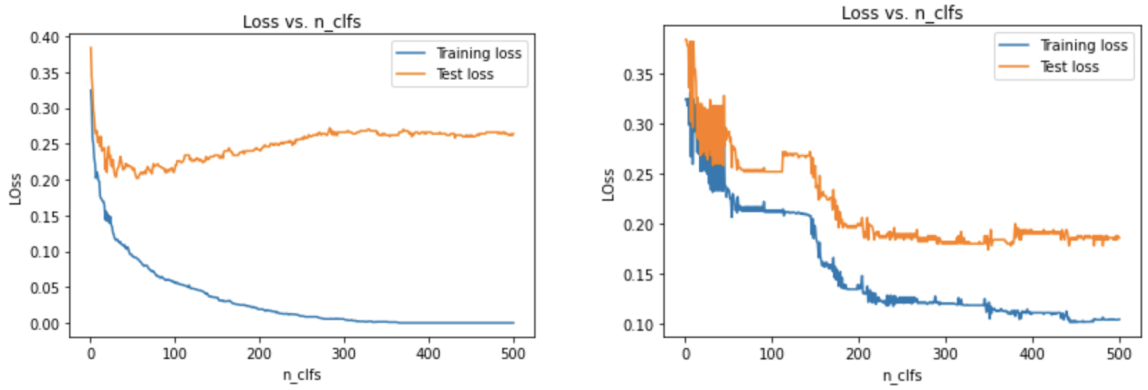
AdaBoost:



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:

Gradient Boosting (left) and Ada Boost (right) Loss Curves:



The loss curves for gradient boosting are much smoother than the loss curves of the AdaBoost, which also fluctuates. For both gradient boosting and AdaBoost, the loss curves become smoother as as the number of classifiers increases. When we use gradient boosting, we aim to decrease the residual loss with each classifier. Gradient boosting is an instance of functional gradient descent, so as residual loss is minimized, the training loss smoothly decreases. With AdaBoost, we change sample weights to better classify misclassified points and points close to decision boundaries with each classifier. The step size also depends on the classification error at each point, so large errors and step sizes can result in more fluctuations.

For gradient boosting, the training loss final value approaches 0, while the training loss final value for AdaBoost approaches 0.1. The gradient boosting test loss final value approaches 0.25, and the AdaBoost test loss final value approaches ≈ 0.2. AdaBoost prioritizes misclassified points, so each added classifier will help decrease classification error. However, as classification error plateaus, sample weights will not change as much and each added classifier will not have a significant impact on the overall model. Thus, the adding classifiers will not increase loss. For gradient boost however, test loss decreases initially but increases after ≈ 100 classifiers. Gradient boosting decreases training loss smoothly. However, adding more classifiers after a certain point can increase variance, so test error will increase.

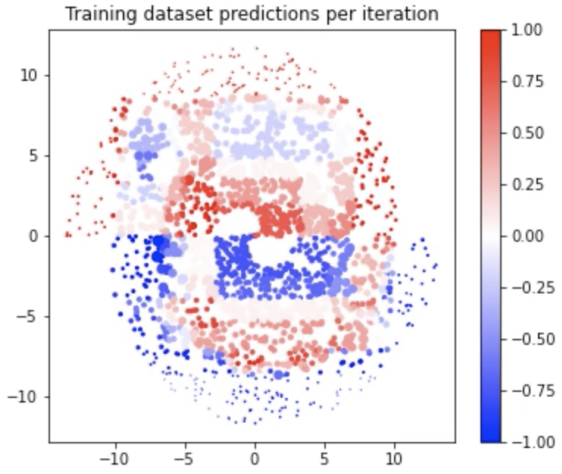
Problem I [2 points]: Compare the final loss values of the two models. Which performed better on the classification dataset?

Solution I: *Gradient Boosting: Training loss* $= 0$, *test loss* $= 0.264$.
AdaBoost: Training loss $= 0.1045$, *test loss* $= 0.186$.
AdaBoost performed better on the classification dataset.

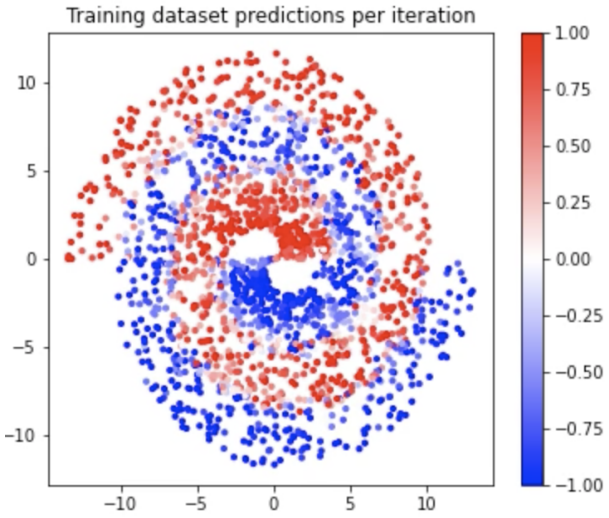
Problem J [2 points]: For AdaBoost, where are the dataset weights the largest, and where are they the smallest?

Hint: *Watch how the dataset weights change across time in the animation.*

Solution J: *The weights are the largest for the points with predictions close to 0, and these points are closer to the boundaries between the different colored points. The weights are the smallest for the points far away from those boundaries. The final misclassified points are also on the boundaries between the blue and red points, which makes sense because the weights should be largest on the misclassified points.*



Gradient Boosting Visualization:



Code Links:
Problem 2
Problem 3