Machine Learning & Data Mining
Set 3
Got a 2 Day Extension from Professor Lee

Caltech CS/CNS/EE 155
February 2nd, 2023
Manuel Rodriguez

Policies

- Due 9 PM PST, January 31th 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”

Machine Learning & Data Mining — Caltech CS/CNS/EE 155
Set 3 — February 2nd, 2023
Got a 2 Day Extension from Professor Lee — Manuel Rodriguez

1 Decision Trees [30 Points]

Relevant materials: Lecture 5

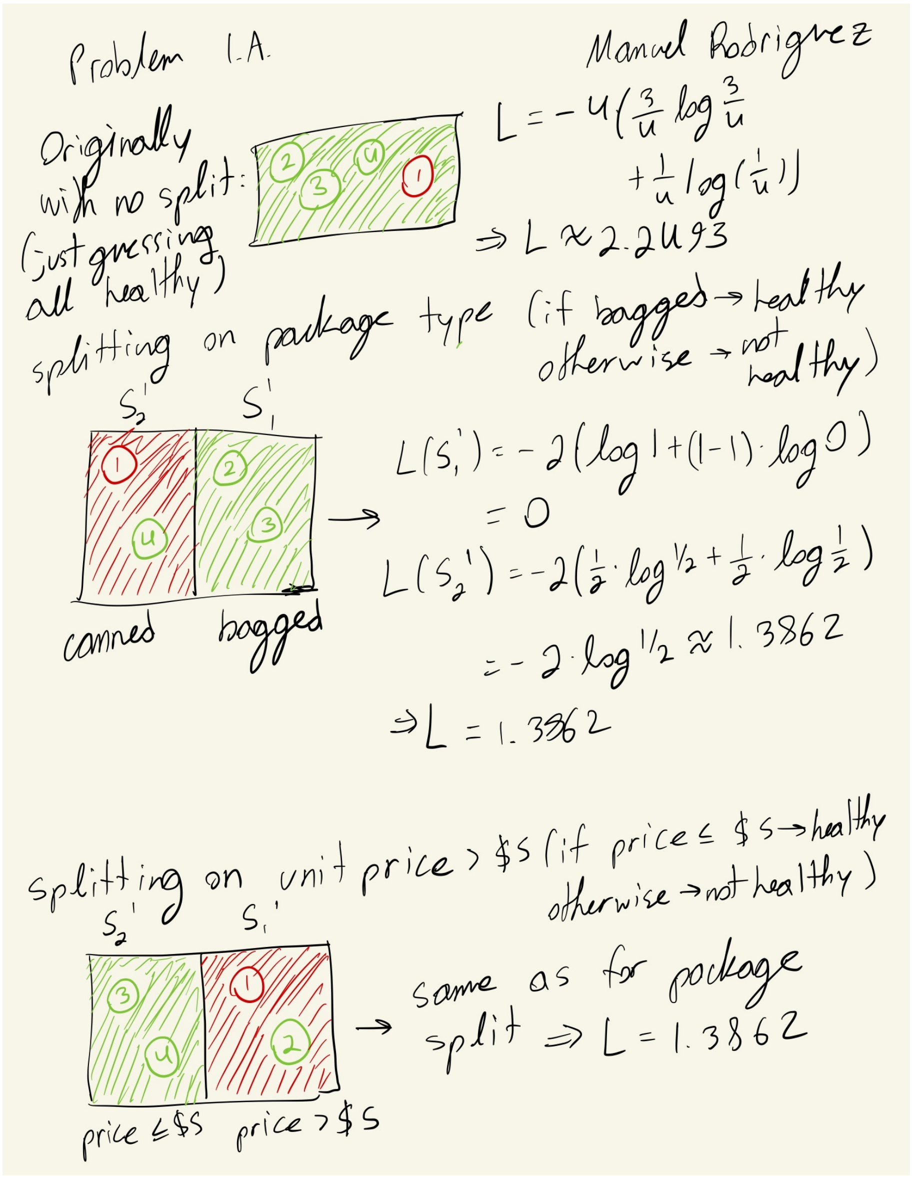
Problem A [7 points]: Consider the following data, where given information about some food you must predict whether it is healthy:

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 1.A. Manuel Rodriguez

Originally with no split: (just guessing all healthy)

2
3
4
1

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

$$\Rightarrow L \approx 2.2493$$

splitting on package type (if bagged → healthy otherwise → not healthy)

S_2'
S_1'
1
4
2
3
canned
bagged

$$\rightarrow L(S_1') = -2(\log 1 + (1-1)\cdot \log 0)$$
$$= 0$$
$$L(S_2') = -2\left(\frac{1}{2}\cdot\log \tfrac{1}{2} + \frac{1}{2}\cdot\log\frac{1}{2}\right)$$
$$= -2\cdot\log \tfrac{1}{2} \approx 1.3862$$
$$\Rightarrow L = 1.3862$$

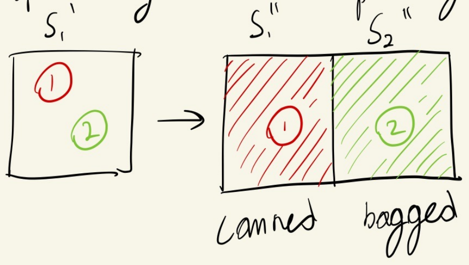
splitting on unit price > \$5 (if price ≤ \$5 → healthy otherwise → not healthy)

→ same as for package split ⇒ $L = 1.3862$

splitting on >5 grams of fat still has same loss $\Rightarrow L = 1.3862$

So we can choose to split on any of them at random since they all reduce the loss to the same number. I will choose to split on unit price > \$5.

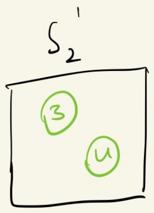
splitting next on package type (if bagged → healthy otherwise → unhealthy)



$L(S_2'') = 0$

$L(S_1'') = 0$

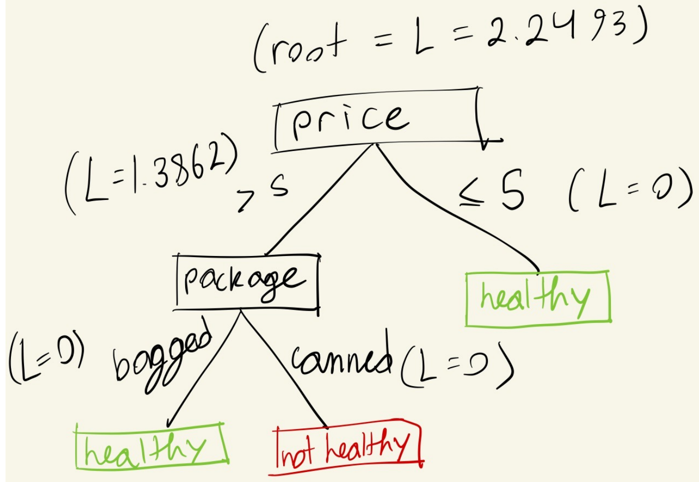
$\Rightarrow L = 0$



$\rightarrow$ already had $L(S_2') = 0$ so no need to split.

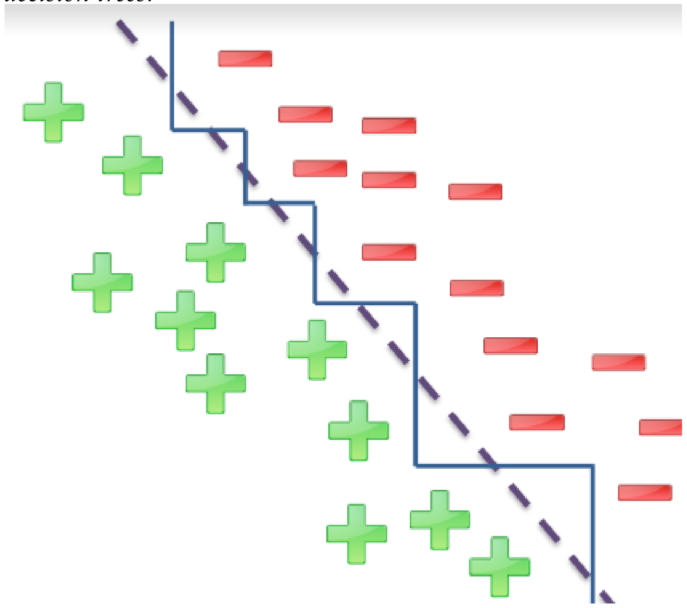
This will minimize the loss since you can't get lower than zero loss.

Below is a sketch of the final Decision Tree:

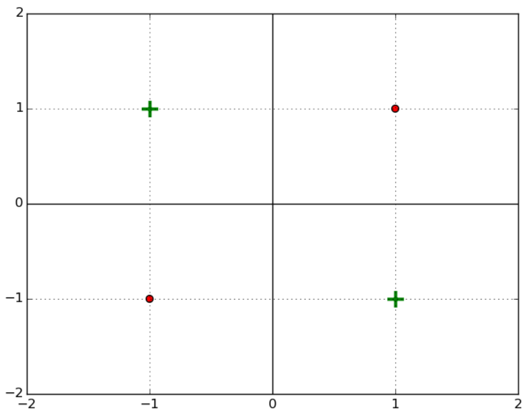


Problem B [4 points]: Compared to a linear classifier, is a decision tree always preferred for classification problems? Briefly explain why or why not. If not, draw a simple 2-D dataset that can be perfectly classified by a simple linear classifier but which requires an overly complex decision tree to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

Solution B: *A linear classifier is can be preferred to a decision tree when the data being observed is linearly separable or linearly separable plus some noise (fixed with regularization). A decision tree would need to be* *overly complex to account for diagonal separation in terms of the axes whereas a simple linear model could readily separate the data. Because this decision tree would be overly complex and decision trees in general are prone to overfitting, this decision tree would likely perform poorly on the out of sample data. Below is an example of a dataset (taken from the class lecture slides) that is simple to model using linear models but complex with decision trees:*



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.)

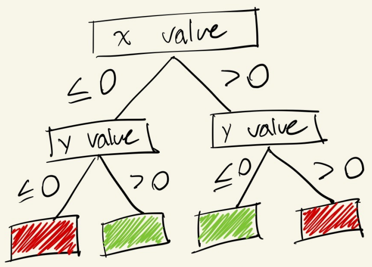
Is there any impurity measure (i.e. any function that maps the data points under a particular node in a tree to a real number) that would have led top-down greedy induction with the same stopping condition to produce the tree you drew? If so, give an example of one, and briefly describe its pros and 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:

ii.

3. ii



This tree classifies all four points correctly and thus has zero classification error.

iii. If there are 100 data points, the worst case scenario is that all the points are in a line along some axis and they are of alternating classification. For this, we would need 99 decision tree thresholds (one between each pair of consecutive points) to distinguish between them all.

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, the maximum number of splits (as mentioned in lecture) is equal to:*
(# *data points*) $\times$ (# *features per data point*) $= N \cdot D$
This makes sense because for each data point, there are D possible features that could be used for splitting and the number of data points determines how much each of these features can separate the data (since many feature splits can result in the same child sets and thus are equivalent when determining training error).

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

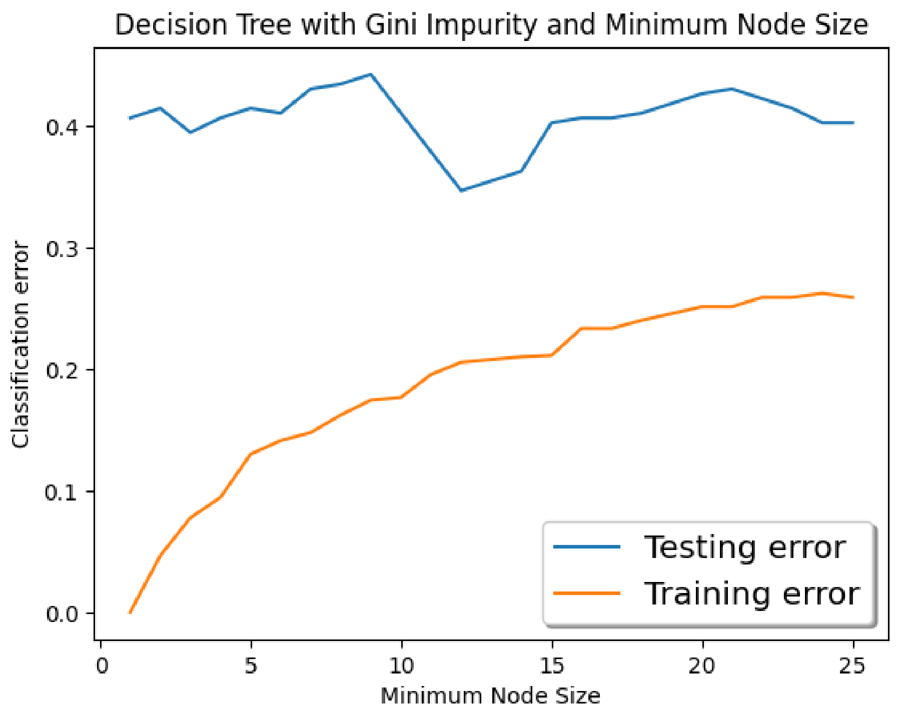
In the following question, your goal is to predict the diagnosis of diabetic retinopathy, which is the final column in the data matrix. Use the first 900 rows as training data, and the last 251 rows as validation data. Please feel free to use additional packages such as Scikit-Learn. Include your code 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/11ta2b8dDlTW09qEkw1LG6xlOYadDFmxc



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.

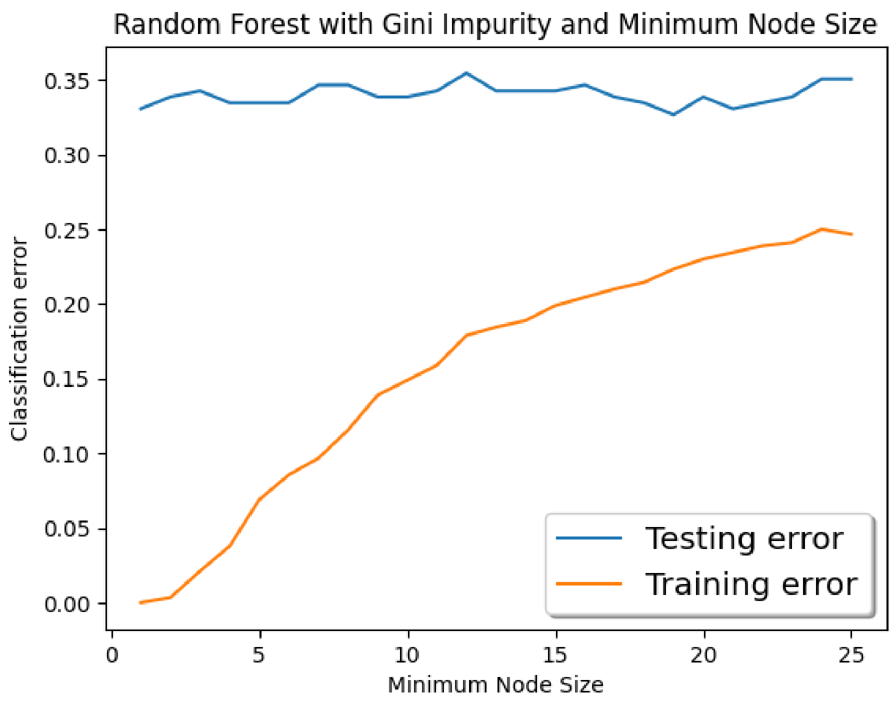
Solution B: *The minimal leaf node size parameter minimizes the test error with min_samples_leaf = 12 producing a test error of around 0.346614. The plot shows how early stopping can decrease the test error of a decision tree model significantly as it stops overfitting. We can see from the plot that if we would have allowed the 'Minimum Node Size' parameter to go all the way to 1, we would have very low training error but a large test error. Similarly, if we allow the decision tree to go to too high of a depth, we can see that the training error will go to the zero but the test error could get worse.*

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: *It seems that the minimal leaf node size parameter minimizes the test error with min_samples_leaf = 12 producing a test error of around 0.346614. The results of this parameter on a Random Forest are similar to that of a Decision Tree with the notable exception that the test error seems to be relatively stagnant and more consistent.*

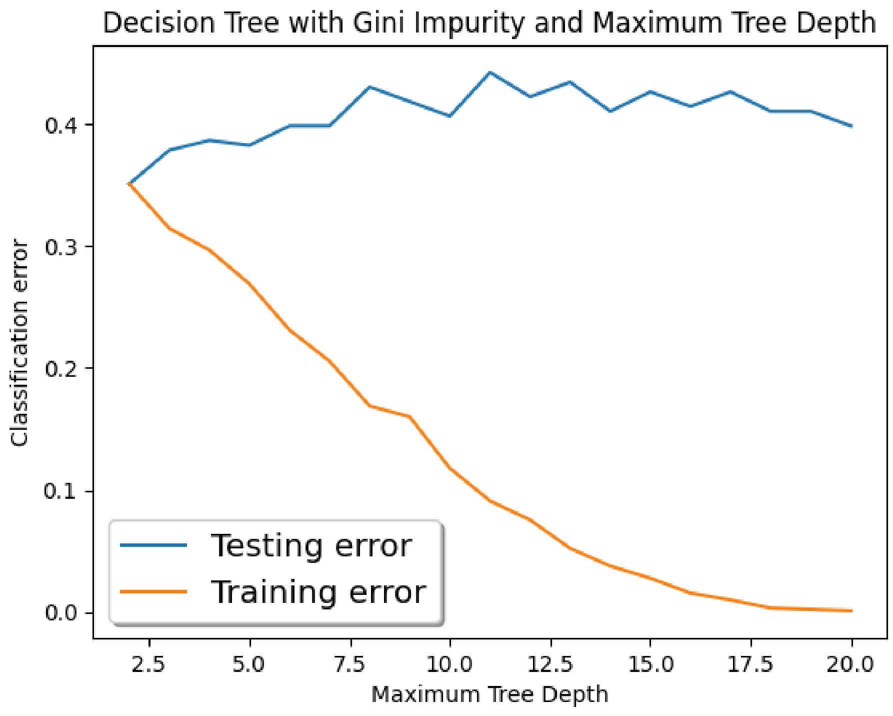
Problem E [4 points]: Do you observe any differences between the curves for the random forest and decision tree plots? If so, explain what could account for these differences.

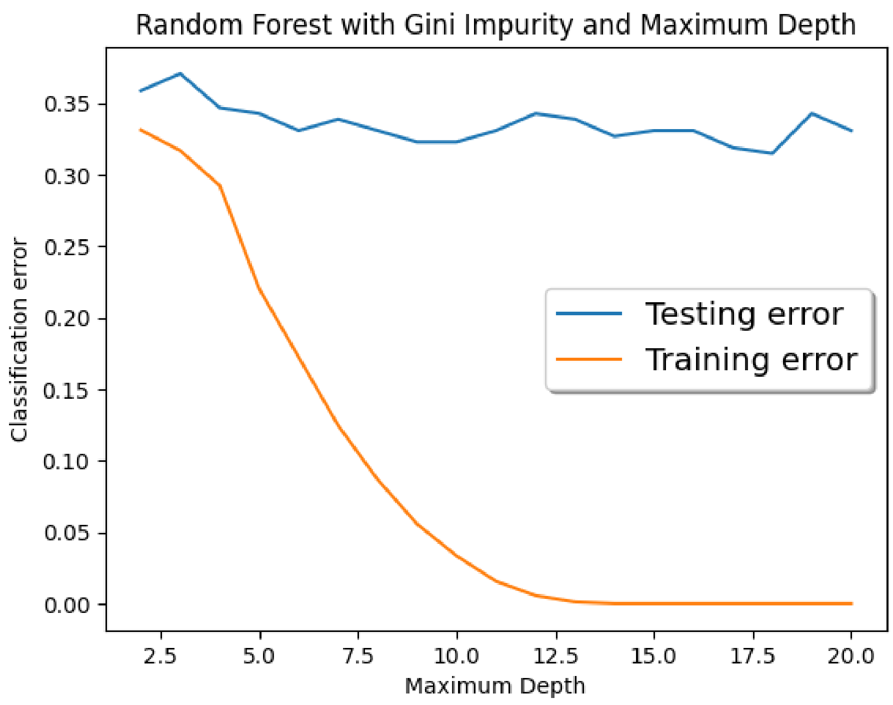
Solution E: *The Random Forest has a more consistent validation error than the Decision Tree, and this makes sense because Random Forests attempt to reduce the variance of Decision Trees by being the average of many Decision Trees. This results in a more consistent validation error and a model that is less prone to overfitting than a regular Decision Tree.*

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: *For the maximum depth parameter using a Decision Tree, max_depth = 2 produces a minimum test error of around 0.350598. For the maximum depth parameter using Random Forests, max_depth = 18 produces a minimum test error of around 0.314741.*

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

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

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: *We will look at each element x_i and show that the exponential loss is greater than the indicator loss. For each x_i, there are two cases: (1) the point is correctly classified $(H(x_i) = y_i)$ or (2) it is incorrectly classified $(H(x_i) \neq y_i)$.*
Let $I(x_i) = \mathbb{1}(H(x_i) \neq y_i)$ and let $E(x_i) = \exp(-y_i f(x_i))$
For the (1) case: We have that $I(x_i) = 0 \leq \exp(-y_i f(x_i)) = E(x_i)$ since the exponential function is always positive.
For the (2) case: We have that $I(x_i) = 1$ and note that $-y_i f(x_i) >= 0$ since y_i and $f(x_i)$ are of opposite signs. This implies that $E(x_i) = \exp(-y_i f(x_i)) \geq e^0 = 1 = I(x_i)$.
In either case, we have that $E(x_i) \geq I(x_i)$ for all x_i, and thus we now know that the final classifier is bounded above by the exponential loss function.

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:

$$\begin{aligned}
D_{T+1}(i) &= \frac{D_T(i)\exp(-\alpha_T y_i h_T(x_i))}{Z_T} \\
&= \frac{D_{T-1}(i)\exp(-\alpha_{T-1} y_i h_{T-1}(x_i))}{Z_{T-1}} \cdot \frac{\exp(-\alpha_T y_i h_T(x_i))}{Z_T} \\
&\vdots \\
&= D_1(i) \cdot \frac{\exp(-\alpha_1 y_i h_1(x_i))}{Z_1} \times \cdots \times \frac{\exp(-\alpha_T y_i h_T(x_i))}{Z_T} \\
&= D_1(i) \cdot \prod_{j=1}^{T} \frac{\exp(-\alpha_j y_i h_j(x_i))}{Z_j} \\
&= \frac{1}{N} \cdot \prod_{t=1}^{T} \frac{\exp(-\alpha_t y_i h_t(x_i))}{Z_t}
\end{aligned}$$

where $Z_t = \sum\limits_{i=1}^{N} D_t(i)\exp(-\alpha_t y_i h_t(x_i))$

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:

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

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: *From Problem B, we start with and simplify from:*

$$\begin{aligned} D_{T+1}(i) &= \frac{1}{N} \cdot \prod_{t=1}^{T} \frac{\exp(-\alpha_t y_i h_t(x_i))}{Z_t} \\ &= \frac{1}{N} \cdot \frac{\exp(\sum_{t=1}^{T} -\alpha_t y_i h_t(x_i))}{\prod_{t=1}^{T} Z_t} \\ &= \frac{1}{N} \cdot \frac{\exp(-y_i f(x_i))}{\prod_{t=1}^{T} Z_t} \end{aligned}$$

Getting the sum of all the $D_{T+1}(i)$ for all i:

$$\begin{aligned} 1 = \sum_{i=1}^{N} D_{T+1}(i) &= \sum_{i=1}^{N} \frac{1}{N} \cdot \exp(y_i f(x_i)) \\ &= \frac{1}{\prod_{t=1}^{T} Z_t} \cdot E \\ \implies E &= \prod_{t=1}^{T} Z_t \end{aligned}$$

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:
From what was given: $Z_t = \sum_{i=1}^{N} D_t(i)\exp(-\alpha_t y_i h_t(x_i))$. We know that $y_i h_t(x_i)$ is either going to be equal to $+1$ or -1 Depending on whether $h_t(x_i) = y_i$ or not. So we can say:

$$Z_t = \sum_{i=1\,:\,h_t(x_i)=y_i}^{N} D_t(i)\exp(-\alpha_t \cdot 1) + \sum_{i=1\,:\,h_t(x_i)\neq yi}^{N} D_t(i)\exp(-\alpha_t \cdot -1)$$

For any given element x_i, we can know which category it should be placed into by using the classifier of the

$\mathbb{1}(h_t(x_i) \neq y_i)$ function. Rewritting Z_t we get:

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

Looking at the sums of $D_t(i)\mathbb{1}(h_t(x_i) = y_i)$ and $D_t(i)\mathbb{1}(h_t(x_i) \neq y_i)$ we see that they are complements and we also know that $\sum_{i=1}^{N} D_t(i) = 1$. So if we let $\epsilon_t = \sum_{i=1}^{N} D_t(i)\mathbb{1}(h_t(x_i) \neq y_i)$, then we know that $\sum_{i=1}^{N} D_t(i)\mathbb{1}(h_t(x_i) = y_i) = 1 - \epsilon_t$. Substituting this into the equation for Z_t, we get 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: *To greedily minimize Z_t, we just try to find where $\frac{\partial Z_t}{\partial \alpha_t} = 0$.*

$$\begin{aligned} 0 = \frac{\partial Z_t}{\partial \alpha_t} &= -(1-\epsilon_t)\exp(-\alpha_t) + \epsilon_t \exp(\alpha_t) \\ (1-\epsilon_t)\cdot e^{-\alpha_t} &= \epsilon_t \cdot e^{\alpha_t} \\ \frac{1-\epsilon_t}{e_t} &= e^{2\alpha_t} \\ \implies \alpha_t^* &= \frac{1}{2}\ln(\frac{1-\epsilon_t}{\epsilon_t}) \end{aligned}$$

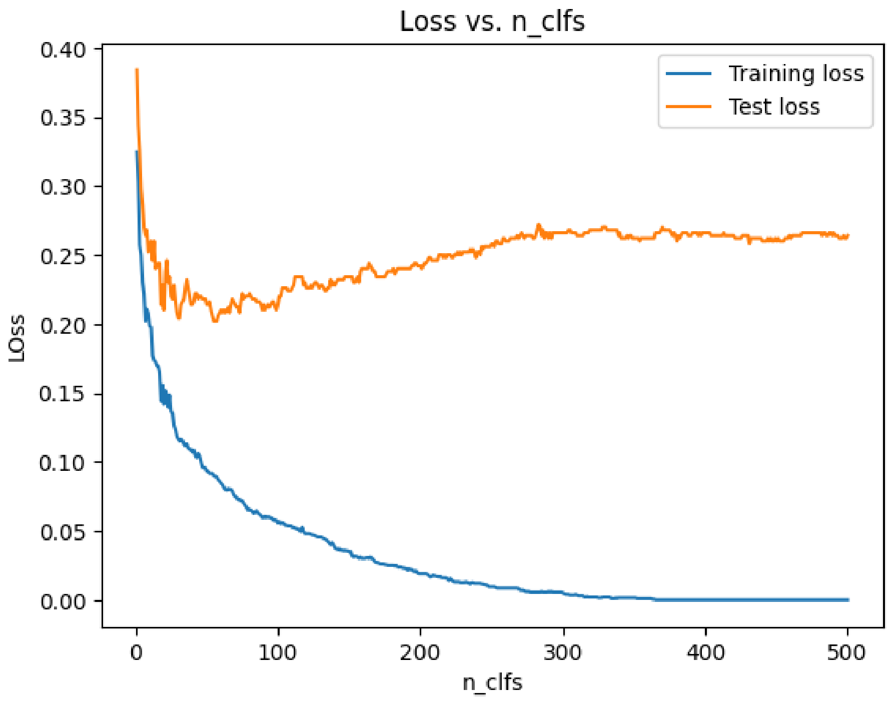
Problem G [14 points]: Implement the `GradientBoosting.fit()` and `AdaBoost.fit()` methods in the notebook provided for you. Some important notes and guidelines follow:

- For both methods, make sure to work with the class attributes provided to you. Namely, after `GradientBoosting.fit()` is called, `self.clfs` should be appropriately filled with the `self.n_clfs` trained weak hypotheses. Similarly, after `AdaBoost.fit()` is called, `self.clfs` and `self.coefs` should be appropriately filled with the `self.n_clfs` trained weak hypotheses and their coefficients, respectively.

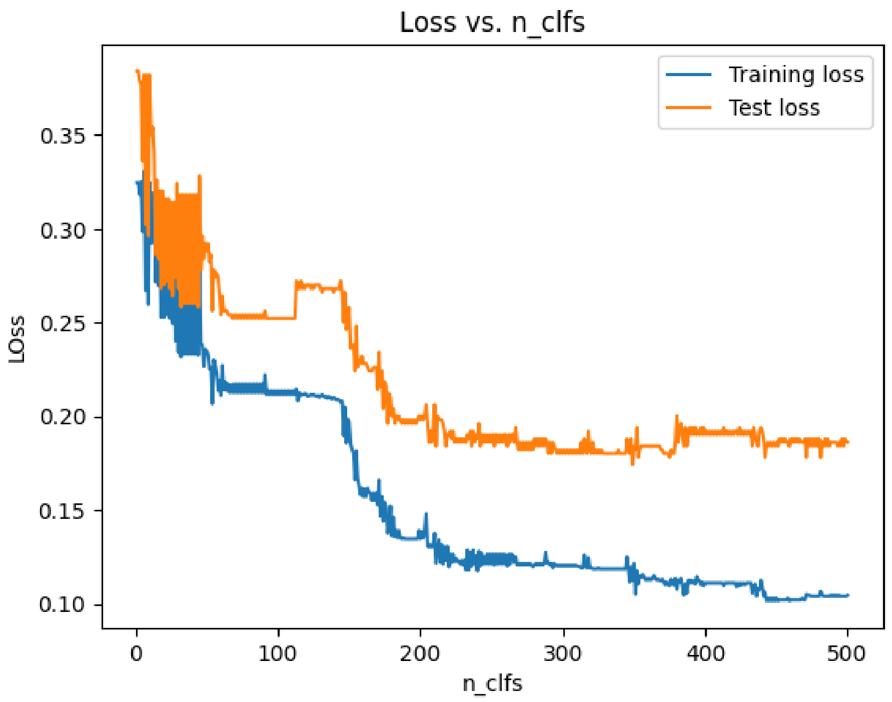
- `AdaBoost.fit()` should additionally return an (N, T) shaped numpy array `D` such that `D[:, t]` contains D_{t+1} for each $t \in \{0, \ldots, \texttt{self.n\_clfs}\}$.
- For the `AdaBoost.fit()` method, **use the 0/1 loss** instead of the exponential loss.
- The only Sklearn classes that you may use in implementing your boosting fit functions are the DecisionTreeRegressor and DecisionTreeClassifier, not GradientBoostingRegressor.

Problem H [2 points]: Describe and explain the behaviour of the loss curves for gradient boosting and for AdaBoost. You should consider two kinds of behaviours: the smoothness of the curves and the final values that the curves approach.

Solution H:
*https://colab.research.google.com/drive/1FC8WW9WLNmHabO2O83KtqNnRkLfPf6U_*
Gradient Boost Loss Curve:



AdaBoost Loss Curve:



The Gradient Boosting Model is smoother than the AdaBoost Model and this is likely due to Gradient Boosting using Regression values and Gradient Descent which provide gradual changes in its weights. In contrast, AdaBoost predicts classification values which makes it so that its predictions are blunter and more extreme. The Gradient Boosting Model gets a lower training error (≈ 0.00) than the AdaBoost Model (≈ 0.10) but a validation error of about 36% more than AdaBoost (≈ 0.19 for AdaBoost and ≈ 0.26 for Gradient Boosting). This coupled with the fact that Gradient Boosting achieves its lowest validation error after only around 100 clfs implies that Gradient Boosting is a more robust algorithm than AdaBoost and was overfitting when the number of clfs increased near the end. In contrast, AdaBoost appears to have plateaued from around 200 clfs.

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

Solution I: *Gradient Boosting performed better on the training data set, being able to get little to no training error, however, AdaBoost performed much better than Gradient Boosting on the validation set. This is likely due to overfitting during Gradient Boosting with so many cfls. If we took the best validation error from both models though, Gradient Boosting with around 100 clfs performs about as well the final AdaBoost Model with 500 clfs.*

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: *Where AdaBoost is most jagged and has jumps, like in the beginning of training, is where the weights are largest. This makes sense because at the beginning of training, there is a lot more variance and information to learn than near the end where only minor changes to the weights are being made, seen through how there is little change in the error between adding clfs.*

4 Convex Functions [7 points, EC 3 Points]

This problem further develops the ideas of convex functions, and provides intuition for why convex optimization is so important for Machine Learning.

Given a convex set $\mathcal{X}$, a function $f : \mathcal{X} \to \mathbb{R}$ is **convex** if for all $\mathbf{x}, \mathbf{y} \in \mathcal{X}$ and all $t \in [0, 1]$:

$$f(t\mathbf{x} + (1-t)\mathbf{y}) \leq tf(\mathbf{x}) + (1-t)f(\mathbf{y})$$

Problem A [3 points]: Let $\mathcal{X}$ be a convex set. If f is a convex function, show that any local minimum of f in $\mathcal{X}$ is also a global minimum.

Solution A: *Let $\hat{x}$ be a local minimum in $\mathcal{X}$ and let $x' \neq \hat{x}$ be the global minimum of $\mathcal{X}$. The line between the two points is in $\mathcal{X}$ by definition and can be denoted by: $x(t) = t\hat{x} + (1-t)x'$. Because f is a convex function on $\mathcal{X}$, we know that*

$$f(x(t)) \leq tf(\hat{x}) + (1-t)f(x') < f(\hat{x})$$

since $0 \leq t < 1$ and $f(x') < f(\hat{x})$. Now, note that if we let $\lim_{t\to 0} x(t)$, we get that points near $\hat{x}$ are less than $\hat{x}$, which is a contradiction since $\hat{x}$ is a local minimum. Therefore, we know that there cannot exist an $x' \neq \hat{x}$ which is a global minimum, and it must be the case that $x' = \hat{x}$ is the global minimum on $\mathcal{X}$.

Problem B [4 points]: *Using part A*, explain why convex loss functions are desirable when training learning models.

Solution B: *Convex loss functions are desirable for training learning models because a learning model usually is trained to minimize its loss locally without knowing whether it is going toward a global or local minimum. However, if we know that any minimum we find will be the global minimmum, then we do not have to worry about getting stuck in a pit and can know that the solution that we are approximating is the best solution.*

Problem C: [3 points, Extra Credit] The Kullback-Leibler (KL) divergence is a measure of statistical distance between two probability distributions (p, q), also called the relative entropy. KL divergence can be used to generate optimal parameters for visualization models (which we will also see in set 4).

$$\mathrm{KL}[P\|Q] = \sum_{x\in\mathcal{X}} p(x) \cdot \log \frac{p(x)}{q(x)}$$

Show that the KL divergence is a convex loss function.

Hint: *Use the log sum inequality*

Solution C: