Machine Learning & Data Mining
Set 3

Caltech CS/CNS/EE 155
January 28th, 2023

---

## Policies

- Due 9 PM PST, January 28th 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*

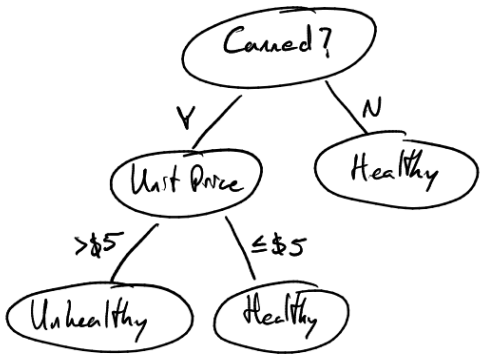
**Problem A [7 points]:** Consider the following data, where given information about some food you must predict whether it is healthy:

| 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:**




$L(S) = -4\left(\frac{3}{4}\ln\left(\frac{3}{4}\right) + \frac{1}{4}\ln\left(\frac{1}{4}\right)\right) \approx 2.25$

$\downarrow -0.86$

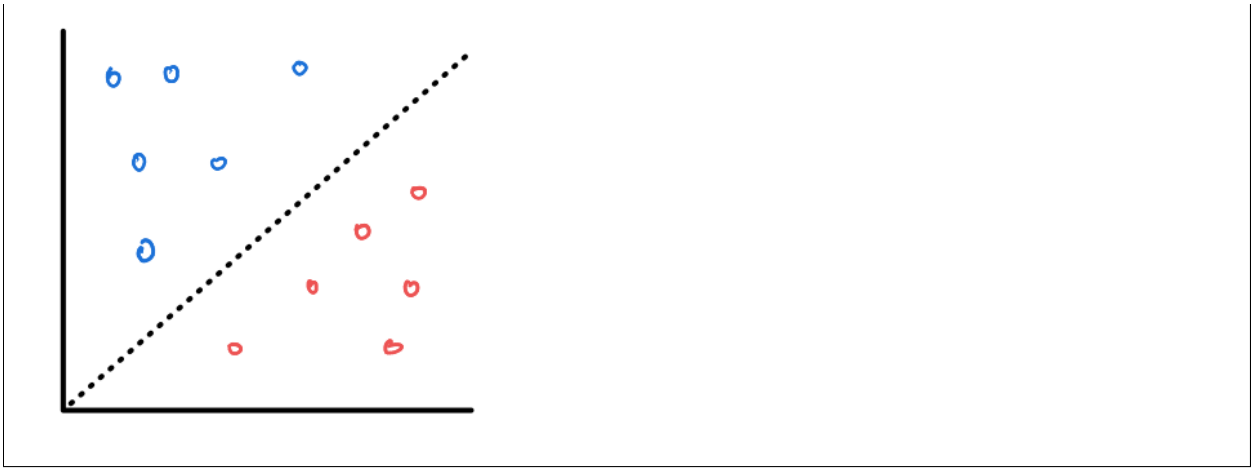
$L(S) = -2\left(\frac{1}{2}\ln\left(\frac{1}{2}\right) + \frac{1}{2}\ln\left(\frac{1}{2}\right)\right) \approx 1.39$

$\downarrow -1.39$

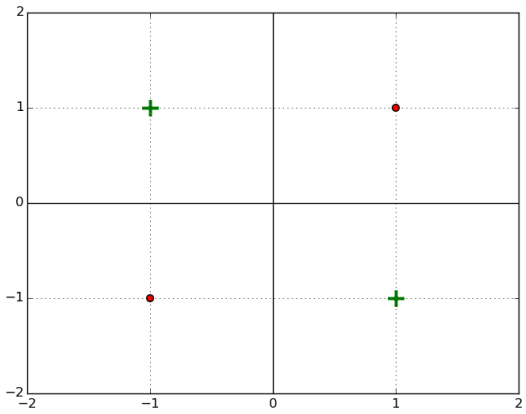
$L(S) = 0$

**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:** *Decision trees can't make diagonal splits, so fitting data that's linearly separable on a diagonal line requires many splits.*



**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))?

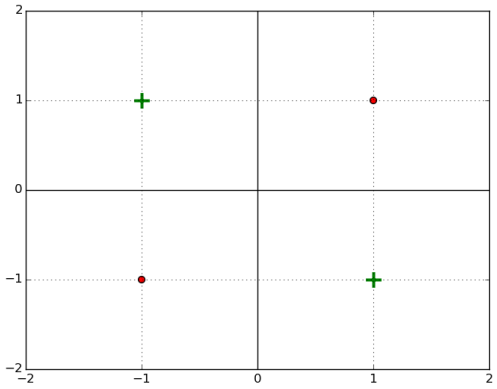
**ii. [5 points]:** Submit a drawing of a two-level decision tree that classifies the above dataset with zero classification error. (You don't need to use any particular training algorithm to produce the tree.)

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).

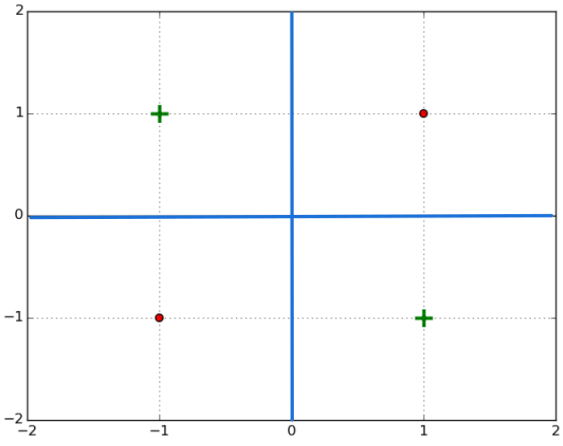
**iii. [5 points]:** Suppose there are 100 data points in some 2-D dataset. What is the largest number of unique thresholds (i.e., internal nodes) you might need in order to achieve zero classification training error (on the training set)? Please justify your answer.

**Solution C:**

**i** *No top-level split results in a reduction of impurity. Therefore no splits are made, and the classification error is 1/2.*




**i** *The blue lines indicate splits (one split at 0 for each axis).*




*An impurity measure where the sum of the impurity of two partitions is less than the impurity of the combined partition would have produced this tree using top-down greedy induction. An example of such a function is* $L(S') = |S'|^2(1 - p_{S'}^2, - (1 - p_{S'})^2)$.

*This has the effect that any impure sample will be split, even if the split doesn't reduce traditional impurity measures. The benefit is helping with situations like this where a single split can't separate the data but a*

*double split can. The downside is that this impurity measure penalizes large partitions, which would probably lead to more overfitting (since large partitions with only a few outliers usually signify a real trend in the data).*

**i** *In the worst case, the data are all in a line of alternating classes, so that zero classification error requires a split between every pair of adjacent data points. With 100 data points, we would need 99 thresholds to achieve zero training 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:** *Assuming that no data points have any identical features, for each feature we could make a split between each pair of consecutive nodes (in that feature). Therefore the number of splits is linear with D and with N.*

$$N \cdot D$$

## 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`

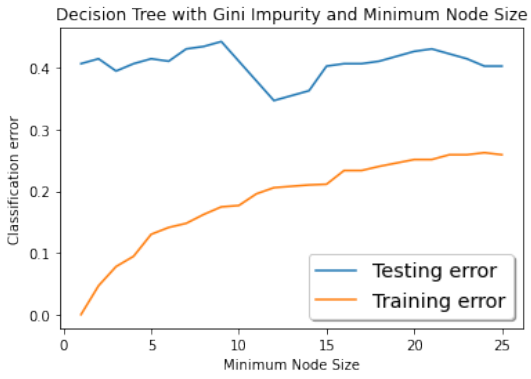
In the following question, your goal is to predict the diagnosis of diabetic retinopathy, which is the final column in the data matrix. Use the first 900 rows as training data, and the last 251 rows as validation data. Please feel free to use additional packages such as Scikit-Learn. Include your code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




**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.

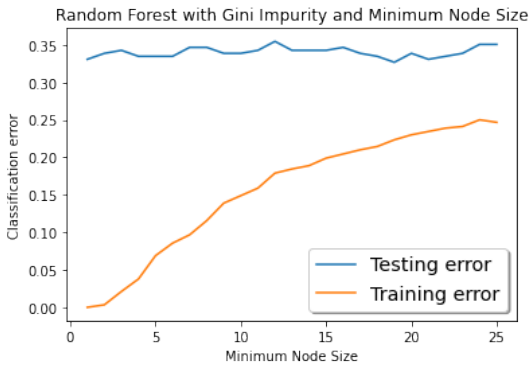
**Solution B:** *Minimum leaf size of 12 minimized test error. In the case of overfitting, early stopping helps prevent the decision tree from creating splits for small numbers of outliers, which reduces variance and improves the performance of the model. If the minimum leaf size is too large, however, the model will underfit because it can't make small enough partitions to capture the complexity of the data. Therefore we expect the optimal value to be somewhere in the middle, as observed in 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:**




**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:** *Minimum leaf size of 19 minimized test error. Early stopping in a random forest has a similar effect compared to decision tree (explained in 2B), but it has less of an impact overall because the use of a random forest already reduces variance, so the test error varies less with minimum leaf size as seen in the plot.*

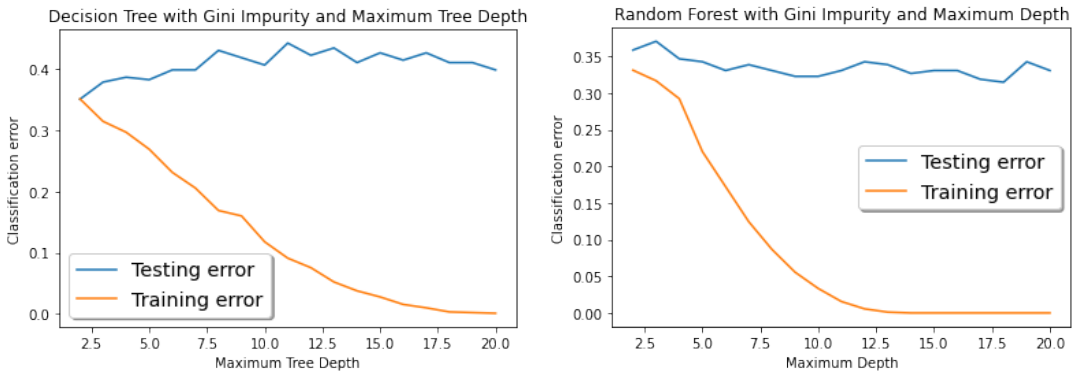
**Problem E [4 points]:** Do you observe any differences between the curves for the random forest and decision tree plots? If so, explain what could account for these differences.

**Solution E:** *Minimum leaf size has a much smaller impact on the random forest, because the random forest's use of many decision trees with sampled training data and feature sets already reduced the variance of the model.*

**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:**

*Random tree: test error minimized with max depth of 2*

*Random forest: test error minimized with max depth of 18*

# 3 The AdaBoost Algorithm [40 points]

*Relevant materials: Lecture 6*

In this problem, you will show that the choice of the $\alpha_t$ 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 $\alpha_t$ 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:** *If* $H(x_i) \neq y_i$, *then* $\text{sign}(f(x_i)) \neq y_i$, *so* $\exp(-y_i f(x_i)) > 1 = \mathbb{1}(H(x_i) \neq y_i)$. *Otherwise,* $\exp(-y_i f(x_i)) > 0 = \mathbb{1}(H(x_i) \neq y_i)$.

*Thus the exponential loss is* $\geq$ *to 0-1 loss for every data point, so it can act as an upper bound for training set error.*

**Problem B [3 points]:** Find $D_{T+1}(i)$ in terms of $Z_t$, $\alpha_t$, $x_i$, $y_i$, 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:**

$$D_1(i) = 1/N$$
$$D_{t+1}(i) = \frac{D_t(i)\exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$
$$D_{T+1}(i) = \frac{1}{N}\prod_{t=1}^{T}\frac{\exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

**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:**

$$\sum_{t=1}^{T} -\alpha_t y_i h_t(x_i) = -y_i \sum_{t=1}^{T} \alpha_t h_t(x_i) = -y_i f(x_i)$$

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

**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:**

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

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

**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 $\epsilon_t$ 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:** *Using the fact that* $h_t(x_i), y_i \in \{-1, 1\}$*:*

$$
\begin{aligned}
Z_t &= \sum_{i=1}^{N} D_t(i) e^{-\alpha_t y_i h_t(x_i)} \\
&= \sum_{i=1}^{N} D_t(i)\left[\mathbb{1}(h_t(x_i) = y_i)e^{-\alpha_t} + \mathbb{1}(h_t(x_i) \neq y_i)e^{\alpha_t}\right] \\
&= \sum_{i=1}^{N} D_t(i)\mathbb{1}(h_t(x_i) = y_i)e^{-\alpha_t} + \sum_{i=1}^{N} D_t(i)\mathbb{1}(h_t(x_i) \neq y_i)e^{\alpha_t} \\
&= \sum_{i=1}^{N} D_t(i)\left[1 - \mathbb{1}(h_t(x_i) \neq y_i)\right]e^{-\alpha_t} + \sum_{i=1}^{N} D_t(i)\mathbb{1}(h_t(x_i) \neq y_i)e^{\alpha_t} \\
&= \left(\sum_{i=1}^{N} D_t(i) - \sum_{i=1}^{N} D_t(i)\mathbb{1}(h_t(x_i) \neq y_i)\right)e^{-\alpha_t} + \sum_{i=1}^{N} D_t(i)\mathbb{1}(h_t(x_i) \neq y_i)e^{\alpha_t} \\
&= (1-\epsilon_t)e^{-\alpha_t} + \epsilon_t e^{\alpha_t}
\end{aligned}
$$

**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 $\alpha_t$ 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:**

$$0 = \frac{dZ_t}{d\alpha_t} = -(1 - \epsilon_t)e^{-\alpha_t} + \epsilon_t e^{\alpha_t}$$

$$\begin{aligned}
(1 - \epsilon_t)e^{-\alpha_t} &= \epsilon_t e^{\alpha_t} \\
\frac{1 - \epsilon_t}{\epsilon_t} &= e^{2\alpha_t} \\
\ln \left( \frac{1 - \epsilon_t}{\epsilon_t} \right) &= 2\alpha_t \\
\frac{1}{2} \ln \left( \frac{1 - \epsilon_t}{\epsilon_t} \right) &= \alpha_t
\end{aligned}$$

**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 $t \in \{0, \ldots,$ `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:** *Colab notebook*

*The gradient boosting curve was much smoother than AdaBoost. Gradient boosting also reached lower training loss, and overfit quickly, whereas with AdaBoost the test loss followed the training loss curve for much longer.*

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

**Solution I:** *AdaBoost performed better with test loss of 0.186 vs gradient boosting's test 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:** *AdaBoost put the most weight on the data points it got wrong, which in this case were the ones at the boundary of the two spirals.*

# 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 $x_1$ be a local minimum of $f$, and let $x_2$ be any other point in $\mathcal{X}$.*

*Since $x_1$ is a local minimum there exists some $t \in [0, 1)$ such that $f(x_1) < f(tx_1 + (1-t)x_2)$. And because $f$ is convex, $f(tx_1 + (1-t)x_2) \leq tf(x_1) + (1-t)f(x_2)$. Combining these inequalities gives*

$$\begin{aligned} f(x_1) &< tf(x_1) + (1-t)f(x_2) \\ f(x_1) - tf(x_1) &< (1-t)f(x_2) \\ (1-t)f(x_1) &< (1-f)f(x_2) \\ f(x_1) &< f(x_2) \end{aligned}$$

*Therefore $f(x_1)$ is less than $f(x_2)$ for any $x_2 \neq x_1$ in $\mathcal{X}$, so any local minimum is a global minimum.*

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

**Solution B:** *Gradient descent (and similar optimization algorithms) converge to local minima. If a local minimum is a global minimum, that means they will converge to the global minimum.*

**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:**