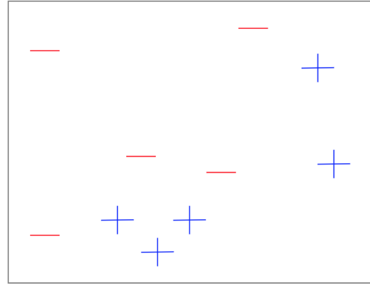
Homework 6

Due: Thursday, March 6, 2025 at 12:00pm **(Noon)**

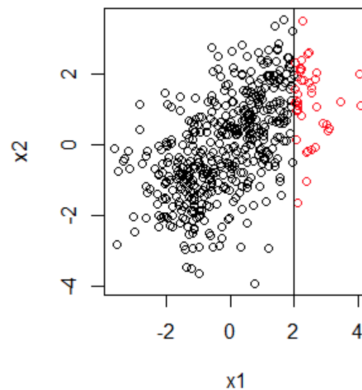
Written Assignment

Problem 1: AdaBoost and Decision Stumps

(15 points)
Below is a graph of points, which are classified as + or -. The features are the x and y coordinates.



Note: Decision stumps are 1D decision trees. For continuous features (in our case, x and y values) a threshold value is chosen, and there is a chosen classification for inputs falling above or below that threshold. For example, one decision stump may be the vertical line represented `x > 2`, with values satisfying this (i.e., to the right of the line) being classified one way, and values not satisfying this (i.e., to the left of the line) being classified separately. Note that decision stumps are **not** equivalent to halfspaces. See example below.



1. Using decision stumps, produce images representing a simulation of the Adaboost algorithm on this dataset. (Hand-drawn pictures are fine.) Represent the changing relative weight of each datapoint by enlarging or shrinking the symbol (+/-). Each image should represent one step of the algorithm, and the last image should be the final classification of the ensemble. Recall that decision stumps can only be vertical or horizontal. You can take a look at this interactive demo for an example of Adaboost running on a different dataset. (**5 points**)

2. What is the difference between weak learners and strong learners in terms of their guarantees on the bounds on the error? (**5 points**)

3. What are the bounds on error of Adaboost at `T=1`? Remember that `T=1` is the case when you have only one learner. What about as `T` approaches infinity? How does this relate to the idea of strong and weak learners? (**5 points**)

Programming Assignment

Introduction

In this assignment, you will be implementing Decision Trees to solve binary classification problems. By the end of this assignment, you will have a classifier that you will use to predict the result of chess matches and classify emails as spam. We recommend that you start this assignment early, as the logic you have to implement may be complicated.

Stencil Code

You can find the stencil code for this assignment using the GitHub link below. We have provided the following files:

- `main.py` is the entry point of your program, which will read in the data, run the classifiers and print the results. You will make small modifications to this file.
- `models.py` contains the `DecisionTree` class, which will contain the bulk of the code you write.
- `get_data.py` contains the data loading and processing. You do **not** need to change this file.

You should *not* modify any code in `get_data.py`. All the functions you need to fill in reside in `main.py` and `models.py`, marked by `TODO`s. To run the program, you just need to run `python main.py`.

You can find and download the assignment here: `HW6 on Github`. If you have any problems, please consult the Download and Submission Guide `here`.

Data

Spambase Dataset

You will be testing your Decision Trees on a real world dataset, the Spambase dataset. Our goal is to train a model that can classify whether an email is spam or not. The dataset features attributes such as the frequency of certain words and the amount of capital letters in a given message. You can find more details on the dataset **here**. We will only be using a subset of the full dataset.

Chess Dataset

Each row of the `chess.csv` dataset contains 36 features, which represent the current state of the chess board. Given this representation, the task is to use the Decision Trees to classify whether or not it is possible for white to win the game. For more information on the dataset, see **here**.

We have taken care of all the data preprocessing required so that you can focus on implementing the machine-learning algorithms! We hope that from these two examples, you can understand the versatility and power of your abstract decision tree classifier.

Decision Trees

Part I: Generic Decision Trees in Python

In this part, you will be implementing a generic decision tree for binary classification given binary features. Your decision tree will take training data $S = ((\mathbf{x}_1, y_1) \ldots (\mathbf{x}_m, y_m))$—where $\mathbf{x}_i \in \{0,1\}^d$ represent the binary feature vectors and $y_i \in \{0,1\}$ are the class labels—and attempt to find a tree that minimizes training error. Recall that the training error for a hypothesis h is defined as the *average* $0-1$ loss

$$L_S(h) = \frac{1}{m} \sum_{(\mathbf{x},y)\in S} \mathbb{1}(y \neq h(\mathbf{x})).$$

The primary methods of the `DecisionTree` class are as follows:

- **Functionality:**
  - `DecisionTree(data, validation_data=None, gain_function=node_score_entropy, max_depth=40)` creates a `DecisionTree` that greedily minimizes training error on the given dataset. The depth of the tree should not be greater than `max_depth`. If `validation_data` is passed as an argument, the validation data should be used to prune the tree after it has been constructed.
  - `predict(features)` predicts a label $y \in \{0,1\}$ given features $\in \{0,1\}^d$. Note that in our implementation features are represented as Python `bool` types (`True`, `False`) and class labels are Python `ints` (`0, 1`).
  - `accuracy(data)` computes accuracy, defined as $1 -$ `loss(self, data)`.
  - `loss(data)` computes the training error, or the *average* loss, $L_{\texttt{data}}(h)$.
- **Helper functions:** This is where most of the algorithmic work will take place. Each helper function begins with an underscore: `_predict_recurs, _prune_recurs, _is_terminal, _split_recurs, _calc_gain`. We already implemented _predict_recurs for you, so please do not modify that function. You should implement the other helper functions without changing the function signatures. **Note that when splitting on the i-th feature, the left child will have data points with i-th feature 0 and the right child will have data points with i-th feature 1.**
- **Debugging:** We have given you two functions to help you visualize your decision tree for debugging purposes. You are free to use (or not use) them. We will not be grading you on whether you use or modify this code.
  - `print_tree()` prints the tree to the command line. We have provided a working implementation, which you are free to improve. The current tree visualization works best for very shallow trees.
  - `loss_plot_vec(data)` returns a vector of loss values where the i-th element corresponds to the loss of the tree with i nodes. The result can be plotted with `matplotlib.pyplot` to visualize the loss as your tree expands.

  To use these debugging functions, you must store the maximum gain at each node and the number of data points that have made it to that node when training using _set_info().

Your task is to ensure that the `DecisionTree` class is fully implemented. If you are unsure where to begin, we have provided `TODO` comments in the stencil code to help get you started! We recommend testing your code incrementally. It would be easiest to program `_is_terminal, _calc_gain` and one of the gain functions first as they are all needed in `_split_recurs`. You should start working on pruning at the last step when you are sure that other functions work. You are free to write your own tests for any of the provided functions to ensure that they are working correctly.

Part II: Measures of Gain

As mentioned in lecture, there are multiple measures of gain that an algorithm can use when determining on which feature to split the current node. In this assignment, you will be implementing and comparing the results of three measures of gain: decrease in training error, information gain (entropy) and Gini index. We recommend reviewing the lecture slides or textbook if these terms sound unfamiliar.

The `DecisionTree` class takes an optional `gain_function` parameter. This function will be one of the three functions left for you to implement: `node_score_error`, `node_score_entropy` and `node_score_gini`. All of these gain functions should return a float.

Part III: Chess Predictions & Spam Classification

Once you have implemented the `DecisionTree` class, you are ready to explore the chess and spam datasets! You should now write code in `main.py` that will print the following loss values:

- For each dataset (`chess.csv`, `spam.csv`)
  - For each gain function (Training error, Entropy, Gini)
    - Print training loss without pruning
    - Print test loss without pruning
    - Print training loss with pruning
    - Print test loss with pruning

Your final program should print **exactly** 24 lines of output. Each line may contain text, but should end with the loss values defined above.

Project Report

(a). Comment on the results of your final program. Discuss the differences in training and test error of pruned and non-pruned trees. Which measure of gain most effectively reduced training error? Was pruning effective? Use tables or graphs to demonstrate your findings. **(5 points)**

(b). Using the `spam.csv` dataset, plot the loss of your decision tree on the *training* set for trees with maximum depth set to each value between 1 to 15. For these plots, the trees should not be pruned and you can use the entropy gain function. Discuss any trends you find and attempt to explain them in three to five sentences. **(10 points)**

(c). Read the short excerpt below from a paper written by Rachel L. Thomas and David Uminsky where they bring up the issue of Machine Learning's dependence on metrics:

> Goodhart's Law, that a measure that becomes a target ceases to be a good measure, remains a relevant lens to see the shortcomings of ML's reliance on metrics. By definition, machine learning is a process in which a measure is the target. Thus, there is an increased likelihood that any risks of optimizing metrics are heightened by AI (Slee 2019), and Goodhart's Law grows increasingly relevant. For instance, if an algorithm learns to produce answers that humans find appealing as opposed to accurate answers.'

What are your initial thoughts on reading this passage? Do you agree with the author? Why or why not? Please elaborate your answer.

Grading Breakdown

We will not be providing accuracy targets for this assignment. A correctly implemented classifier should be able to reach a fairly low loss on both datasets. The autograder will check your tree structure as well as measures of gain against a small toy dataset. As usual, your code will be evaluated for correctness.

Written Question	15%
Decision Tree Classifier	70%
Report	15%
Total	100%

Handing in

You will hand in both the written assignment and the coding portion on gradescope, separately.

1. Your written assignment should be uploaded to gradescope under "Homework 6."
2. Submit your hw6 github repo containing all your source code and your project report named **report.pdf** on gradescope under "Homework 6 Code". **report.pdf** should live in the root directory of your code folder; the autograder will check for the existence of this file and inform you if it is not found. For questions, please consult the download/submission guide.

Anonymous Grading

You need to be graded anonymously, so do not write your name anywhere on your handin. You will lose points on your assignment if your name appears on your handin.

Obligatory Note on Academic Integrity

Plagiarism—don't do it.

As outlined in the Brown Academic Code, attempting to pass off another's work as your own can result in failing the assignment, failing this course, or even dismissal or expulsion from Brown. More than that, you will be missing out on the goal of your education, which is the cultivation of your own mind, thoughts, and abilities. Please review this course's collaboration policy and, if you have any questions, please contact a member of the course staff.