Homework 1

Due: February 13, 2018 at 7:00PM

Written Questions

Problem 1: Decision Trees

Trapped in the Upside Down, Will Hawkins could only communicate with his mother Joyce in the real world by activating a series of lights strung up in his house. Each light was assigned a letter of the alphabet, and Will communicated words by activating lights corresponding to each letter in series. After a near death experience with the Demogorgon, Joyce decided the current communication paradigm was too inefficient. If all lights can be used simultaneously, there are many more available permutations of lights for communication. Joyce has taken down the lights labelled A through Z, and has raised lights labelled numerically 1 to d. Each light can exist in one of two states: on or off. Additionally, Joyce deduced that Will only ever communicates two distinct messages: RUN and STAY. In other words, Joyce must deal with a binary classification problem to construct a decision tree.

Part 1

As Will is still in the Upside Down, he cannot communicate with Joyce directly to verify the meaning of his code. Joyce hopes to employ PAC learning to understand Will's code. For every code Will sends, Joyce observes whether the code means STAY or RUN.

1. Suppose a decision tree is constructed by branching on the attributes in sequential order (1 to d). How many possible decision trees are there of full depth? In other words, how many coding schemes could Will create using all d lights? (We're looking for a simple upper bound, not an exact count of all distinct functions.)

2. Expressed in terms of d, δ and ϵ, how many observations are needed to conclude a hypothesis is probably, approximately correct? Is PAC learning feasible (polynomial) for this problem?

Part 2

Will is considering some alternative coding schemes, and wants to know how they would translate into decision trees. For each of the following coding schemes, how many leaves would there be in the smallest decision tree? Express in terms of d.

1. $$\begin{cases} \text{At least one light is on} \implies \text{RUN} \\ \text{No lights are on} \implies \text{STAY} \end{cases}$$

2. $$\begin{cases} \text{An odd number of lights are on} \implies \text{RUN} \\ \text{An even number of lights are on} \implies \text{STAY} \end{cases}$$

Problem 2: Partial Orderings

Background Information: A *partial ordering* (denoted $\preceq$) is a binary relation over a set S that satisfies the following properties:

- *Reflexivity*: $\forall s \in S, \quad s \preceq s$
- *Antisymmetry*: $\forall s, t \in S, \quad s \preceq t, t \preceq s \implies s = t$
- *Transitivity*: $\forall r, s, t \in S, \quad r \preceq s, s \preceq t \implies r \preceq t$

A *total ordering* differs from a partial ordering in providing an additional guarantee that all elements within S can be compared in the relation. As an example, the $\leq$ relation is a total ordering on the real numbers. Any two numbers a, b may be compared such that either $a \leq b$ or $b \leq a$. Additionally, the $\leq$ relation satisfies the conditions of reflexivity, antisymmetry and transitivity stated above. *Note: All total orderings are also partial orderings.*

One example of a partial ordering is the $\subseteq$ relation. The properties of reflexivity, transitivity and antisymmetry are inherent to the $\subseteq$ relation. However, given two sets A and B, there is no guarantee that either $A \subseteq B$ or $B \subseteq$ A. Now, consider a classification problem defined as follows:

Input Space: The set of all possible n-element permutations.
Hypothesis Space: Is the given ordering consistent with a hidden partial ordering?

The input space consist of all the possible ways that a given set of n elements can be arranged. The hypothesis space maps each permutation of the elements to 0 or 1, depending on whether or not the permutation is consistent with the partial ordering.

1. What is the hypothesis space H of representations the learner is searching?
2. Show that $|H|$ is between $n!$ and $3^{(n^2)}$. *Hint: Each $h \in H$ may be represented as a Directed Acyclic Graph (DAG).*
3. Show that the amount of data needed to ensure the ERM solution is PAC is polynomial in n.
4. Describe an algorithm you could use to efficiently find an ERM solution for a given sample of observations.

Programming Assignment

Introduction

The purpose of this assignment is to give you a chance to implement decision trees for binary classification in Python. By the end of this assignment, you will have implemented a generic `DecisionTree` class that you will use to predict the result of chess matches and classify spam emails.

Stencil Code & Data

The stencil code for this assignment is available on our course website. It contains all the necessary starter Python code and the two datasets that we will be using for this assignment. Both datasets are available as CSV files and are located in the `data` folder. We have taken care of all the data preprocessing required so that you can focus on implementing the machine-learning algorithms!

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 classify whether or not it is possible for white to win the game. For more information on the dataset, see **here**.

Spam Dataset

Each row of the `spam.csv` dataset represents an email. For each email, 46 common words were chosen and the data indicates whether or not a given email contains each of these 46 words. The task is to classify the email as a spam email or regular email. Our TA staff modified **this** dataset to create `spam.csv`.

The Assignment

We have provided the following files as stencil code:

- `decision_tree.py` contains the `DecisionTree` class. We have left most of the methods in this class for you to implement.
- `get_data.py` contains the data loading and processing. You do **not** need to change the contents of this file.
- `main.py` is the entry point for your program. It calls the `get_data()` function and passes the necessary data to your `DecisionTree` class. You are free to add code to this file to explore both datasets and generate graphs for your final report. (See the Report section below for more details.)

To run the program, you should run `python main.py` from your terminal.

Please do **not** edit the filenames or the general structure of the program. You may add helper functions to either file, but you should not rename any of the functions that have already been provided. The programs we wrote to help grade everyone accurately and efficiently depend on this structure.

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 $S = ((x_1, y_1) \dots (x_m, y_m))$—where $x_i \in \{0,1\}^d$ represents the binary feature space and $y \in \{0,1\}$ are the class labels—and attempt to find a tree that minimizes training error. Recall that training error for a hypothesis h is defined as

$$L_S(h) = \sum_{(x,y) \in S} (y \neq h(s)).$$

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

- `DecisionTree(data, max_depth, validation_data=None, gain_function=train_error)` 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`).
- `loss(data)` computes the *average* loss, $L_{\texttt{data}}(h)$/`len(data)`.
- `accuracy(data)` computes accuracy, defined as $1 -$ `loss(self, data)`.
- `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.

Most of the algorithmic work will actually take place in the helper functions, each beginning with an underscore: `_predict_recurs, _prune_recurs, _is_terminal, _split_recurs, _calc_gain`. You should implement these functions without changing the function signatures.

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: `train_error`, `entropy` and `gini_index`.

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.

Report

Introduction to Project Reports

While learning to implement machine-learning algorithms is incredibly important, it is arguably more important to be able to effectively interpret the results of these algorithms. The results, of course, will depend on a variety of factors including the type of dataset, the size of the dataset and the hyperparameters of the algorithm.

To thoroughly develop these skills, each programming assignment in this course will be accompanied by a short report in which you will answer a few guiding questions about the results of your algorithm. To answer these questions, you may need to write new code that generates some output (potentially a value or graph) that you will want to include in your writeup. For example, you may be asked to contrast the results of running the same algorithm on different datasets or to explore the effect of changing a certain hyperparameter in your algorithm. By the end of this course you will not only be able to implement common machine-learning algorithms but also develop intuition as to how the results of a given algorithm should be interpreted.

The next section outlines some guiding questions that you should answer in your report. Please leave any code that you use in your final handin but make sure that it is **not** run by default when your program is run. We ask that your final report be a PDF file named `report.pdf`, which must be handed in along with your code. You may use any program to create the PDF file, but we highly recommend using LaTeX. We have provided an example report available on our course website to get you started.

Decision Trees: Guiding Questions

In your report, please answer the following guiding questions:

- 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? Could you use graphs or numbers to explain what you have found?
- 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. Discuss your findings in a paragraph.
- (Extra credit) In this assignment, you used a greedy algorithm in an attempt to minimize the training error of your decision tree. Think of another algorithm that could optimize the construction of your decision tree, implement it and compare the results. Include at least one graph.

In addition to answering these guiding questions, you should treat the report as you would a `README`—you should document any known bugs and make a note of other students who you collaborated with for the programming portion of this assignment.

Important: Please do **not** print your report. This portion of the assignment must be handed in with your code and **should not** be handed in with the written component. We will not grade printed reports.

Handing in

Written Questions

Answers to the written questions should be printed, stapled and labeled with the date, homework number and your unique course ID. You should place your handin in the CS1420 handin box located on the second floor of the CIT.

Programming Assignment

To hand in the programming component of this assignment, first ensure that your code runs on *Python 3* using our course `virtualenv`. You can activate the `virtualenv` on a department machine by running the following command in a Terminal:

```
source /course/cs1420/cs142_env/bin/active .
```

Once the `virtualenv` is activated, run your program and ensure that there are no errors. We will be using this `virtualenv` to grade all programming assignments in this course so we recommend testing your code on a department machine each time before you hand in. Note that handing in code that does not run may result in a significant loss of credit.

To hand in the coding portion of the assignment, run `cs142_handin hw1` from the directory containing all of your source code and your report in a file named `report.pdf`.

Anonymous Grading

You need to be graded anonymously, so do not write your name anywhere on your handin. Instead, you should use the course ID that you generated when filling out the collaboration policy form. If you do not have a course ID, you should email the HTAs as soon as possible.

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.

Feedback

Many aspects of this course have been radically redesigned compared to previous iterations of the course. To continually improve this class and future iterations of the class, we are offering an online **anonymous feedback form**. If you have comments, we would love to hear your constructive feedback on this assignment and on any other aspects of the course.