CS 486/686 Assignment 2 (105 marks)

Alice Gao

Due Date: 11:59 pm ET on Wednesday, November 3, 2021

Changes

- v1.1: Added a description of how to choose the potential split points in Q2.

Academic Integrity Statement

If your written submission on Learn does not include this academic integrity statement with your signature (typed name), we will deduct 5 marks from your final assignment mark.

I declare the following statements to be true:

- The work I submit here is entirely my own.
- I have not shared and will not share any of my code with anyone at any point.
- I have not posted and will not post my code on any public or private forum or website.
- I have not discussed and will not discuss the contents of this assessment with anyone at any point.
- I have not posted and will not post the contents of this assessment and its solutions on any public or private forum or website.
- I will not search for assessment solutions online.
- I am aware that misconduct related to assessments can result in significant penalties, possibly including failure in the course and suspension. This is covered in Policy 71: https://uwaterloo.ca/secretariat/policies-procedures-guidelines/policy-71.

By typing or writing my full legal name below, I confirm that I have read and understood the academic integrity statement above.

Instructions

- Submit any written solutions in a file named `writeup.pdf` to the A1 Dropbox on `Learn`. Submit any code to `Marmoset` at https://marmoset.student.cs.uwaterloo.ca/. No late assignment will be accepted. This assignment is to be done individually.
- I strongly encourage you to complete your write-up in Latex, using this source file. If you do, in your submission, please replace the author with your name and student number. Please also remove the due date, the Instructions section, and the Learning goals section. Thanks!
- Lead TAs:
  - Dake Zhang (dake.zhang@uwaterloo.ca)
  - Blake Vanberlo (bvanberlo@uwaterloo.ca)

  The TAs' office hours will be posted on MS Teams.

Learning goals

Decision Trees

- Implement the decision tree learner algorithm to learn a decision tree using a data set with real-valued features.
- Determine the prediction accuracy of a decision tree on a data-set.
- Perform pre-pruning and post-pruning. Determine the best parameter value for pruning using cross validation.

1 Three Proofs (15 marks)

(a) Prove that A^* search is optimal.

Given an admissible heuristic function, if A^* search returns a path p as a solution, then p is the lowest-cost path from the start node to a goal node.

Hint: Try a proof by contradiction. Start by assuming that another path p' is the lowest-cost path from the start node to a goal node. Then, derive a contradiction by comparing the costs of p and p'.

Marking Scheme:

(4 marks) A correct proof.

(1 mark) The proof is clear, succinct, and easy to understand.

(b) Prove that A^* search is optimally efficient.

Given the same start node and the same heuristic function, every path expanded by A^* search will be expanded by another optimal search algorithm.

Hint: Try a proof by contradiction. Consider a path p' whose f value is less than the cost of the lowest-cost path. Assume that A^* search expands path p' but another optimal search algorithm A' does not expand path p'. Then, construct another search problem in which p' leads to a goal node and show that A' does not find the optimal solution on this problem.

Marking Scheme:

(4 marks) A correct proof.

(1 mark) The proof is clear, succinct, and easy to understand.

(c) Consider two variables X and Y, and a binary constraint $c(X, Y)$. Assume that $\langle X, c(X, Y)\rangle$ is NOT consistent and $\langle Y, c(X, Y)\rangle$ is consistent. Assume that after removing one value a from X's domain, $\langle X, c(X, Y)\rangle$ becomes consistent. Prove that after reducing X's domain, $\langle Y, c(X, Y)\rangle$ remains consistent.

Hint: A direct proof and a proof by contradiction both work. For a proof by contradiction, start by assuming that $\langle Y, c(X, Y)\rangle$ becomes inconsistent after removing a from X's domain. You will likely need to apply the arc consistency definition more than once.

Marking Scheme:

(4 marks) A correct proof.

(1 mark) The proof is clear, succinct, and easy to understand.

2 Decision Trees (90 marks)

You will implement an algorithm to build a decision tree for a HTRU2 data-set. Read more about the data set here.

Information on the provided code

We have provided several files.

- `data.csv` includes the data set. `dt_global.py` defines several useful global variables. `dt_provided.py` defines some useful functions for reading the data-set and splitting the data-set into different folds.

  Do not modify these files and do not submit these files.

- `dt_core.py` contains empty functions for generating the tree and performing pruning. `dt_cv.py` contains empty functions for performing cross validation. You need to complete all the empty functions in these files.

Splitting Examples

The data-set has real-valued features. When generating a decision tree, use **a binary split** at each node. At each node in the decision tree, choose a feature and a split point for the feature using the procedure described on slide 44 of lecture 7 to choose potential split points. The node should test whether a feature has a value greater than the split point or not. Along a path from the root node to a leaf node, we may test a real-valued feature multiple times with different split points.

The Attributes in an Anytree Node

We use the `anytree` package to store the decision tree. Our unit tests make use of several custom attributes in an Anytree Node. Make sure that you use these attributes so that you will pass our tests.

- `name`: string. Each Node requires the `name` attribute. The `name` of each node should be unique. You can generate this attribute in any way you like. We won't test this attribute.

- `parent`: Node. To construct the tree properly, you either need to set a Node's `parent` attribute or a Node's `children` attributes. See the Node documentation for more details. We recommend setting the `parent` attribute of each Node. Once the `parent` attribute is set, the `children` attribute will be set automatically by Anytree. If you set the left child node's parent attribute before setting the right child node's parent

attribute, then you can retrieve the left child node as `children[0]` and the right child node as `children[1]`.

- The `feature` attribute stores the chosen feature as a string. The `split` attribute stores the split point value as a float. Any non-leaf node should have the `feature` and `split` attributes.
- The `decision` attribute stores the decision as an integer. Any leaf node should have the `decision` attribute.

Tie-Breaking Rules

Please use the following tie-breaking rules to ensure that your program passes the unit tests.

(1) If a leaf node has examples with different labels, we need to determine a decision using majority vote. If there is a tie, return the label with the smallest value. For example, if a leaf node has two examples with label 4 and two examples with label 5, the majority decision should be 4.

(2) Given a feature, if there are multiple split points with the same maximum information gain, choose the split point with the smallest value. For example, suppose that, the split points `9.3` and `9.5` tie for the maximum information gain among all the spilt points for the feature, we will choose to split on `9.3`.

(3) Suppose that, for each feature, we have identified the split point with the maximum information gain. Given the best split points for the features, if multiple split points have the same maximum information gain, choose the first feature based on the order of the features in the first row of the `data.csv` file.

For example, assume that feature A comes before feature B in the order. Suppose that the information gain for the best split point for A and the best split point for B tie for the maximum information gain (among the best split points for all the features), we will choose to split on the best split point for A.

Floating Point Comparisons

A correct implementation may fail the Marmoset tests due to floating point issues. To prevent this, please compare floating-point values using a tolerance. We have provided two functions `less_than` and `less_than_or_equal_to` in `dt_provided.py`. Make sure that you use these functions when comparing floating-point values.

Ten-Fold Cross-Validation

The purpose of performing cross validation is to choose the best value for a parameter. We will use cross validation to choose the best parameter value in pre-pruning and post-pruning.

In ten-fold cross-validation, each data point serves double duty — as training data and validation data.

1. Split the data into ten subsets using `preprocess` in `dt_provided.py`.
2. For each parameter value, perform ten rounds of learning. In the i-th round, the validation set is the i-th fold and the training set consists of the remaining 9 folds.
3. In the i-th round, learn a decision tree with the parameter value using the training set. Determine the prediction accuracy of the decision tree on the training set and on the validation set. The prediction accuracy is the fraction of examples that the tree predicts correctly. (Don't worry if you generate different trees in different rounds. We do not care about the trees generated. We only care about the prediction accuracies.)
4. After the ten rounds, calculate the average prediction accuracy of the decision tree on the training set and on the validation set, where the average is taken over the ten rounds. For each parameter value, you will produce two numbers, the average prediction accuracy on the training set and the average prediction accuracy on the validation set.
5. Choose the parameter value with the highest average prediction accuracy on the validation set.

Packages:

You can assume that numpy (version 1.19.5) and anytree (version 2.8.0) are available in our testing environment. If you want to use a different package, please post a question on Piazza.

Efficiency:

The unit tests will evaluate your implementation for correctness and efficiency. If your implementation does not terminate within a pre-defined time limit, it will fail the unit test. We set the time limits by taking our run-times and multiplying it by a small constant factor. For your information, our program terminates within 5 seconds for part b1, 160 seconds for part b2, and 70 seconds for part b3.

Here are some advice for improving your program efficiency.

- Limit the external packages that your program uses. Do not use `pandas` — it will slow down your program considerably. Limit your use of `numpy`. It's sufficient to use arrays and dictionaries only.
- Think of ways to perform pruning efficiently. You may be able to performing pruning without modifying the tree. You may also be able to start with a previously generated tree instead of building a full tree all over again.

Please complete the following tasks.

(a) Complete all the empty functions in `dt_core.py` and `dt_cv.py` and submit both files on Marmoset.

Marking Scheme: (85 marks)

- `get_splits`
  (1 public test + 4 secret tests) * 2 marks = 10 marks
- `choose_feature_split`
  (1 public test + 4 secret tests) * 3 marks = 15 marks
- `split_examples`
  (1 public test + 4 secret tests) * 1 mark = 5 marks
- `learn_dt` (and `split_node`)
  1 public test * 1 mark + 2 simple secret tests * 2 marks + 2 full-tree secret tests * 5 marks = 1 + 4 + 10 = 15 marks
- `predict`
  (1 public test + 4 secret tests) * 1 mark = 5 marks
- `get_prediction_accuracy`
  (1 public test + 4 secret tests) * 1 mark = 5 marks
- `post_prune`
  (1 public test + 4 secret tests) * 2 marks = 10 marks
- `cv_pre_prune`
  (1 public test + 4 secret tests) * 2 mark = 10 marks
- `cv_post_prune`
  (1 public test + 4 secret tests) * 2 mark = 10 marks

(b) Complete the questions below. Include your answers in `writeup.pdf` and submit it on Learn.

(1) Suppose that we built a decision tree called `tree-full` using the data set. What is the maximum depth of `tree-full`?

Marking Scheme:
(1 mark) Correct value of the maximum depth of the tree.

(2) Suppose that we want to pre-prune `tree-full` using the maximum depth criterion and using majority voting to make decisions at leaf nodes. What is the best value of the tree's maximum depth through ten-fold cross-validation? Please use the range `list(range(0, 40))` for the maximum depth.

Marking Scheme:
(1 mark) Correct value of the best maximum depth for pre-pruning.

(3) Suppose that we want to post-prune `tree-full` using the minimum expected information gain criterion.

For post-pruning, grow a full tree first. Define a value for the minimum expected information gain. Next, keep track of all the nodes that only has leaf nodes as its descendants. Let's call these nodes `leaf parents`. For each `leaf parent` node, if the expected information gain is less than the pre-defined value, then delete its children and convert this node to a leaf node with majority decision. `Repeat` this process until the expected information gain at every `leaf parent` node is greater than or equal to the pre-defined value.

What is the best value for the minimum expected information gain through ten-fold cross-validation? Use the range `list(np.arange(0, 1.1, 0.1))` for the minimum expected information gain.

Marking Scheme:
(1 mark) Correct value of the best minimum expected information gain for post-pruning.

(4) In the previous parts, you have experimented with several ways of building a decision tree for the data-set. If you had a choice, what is the best strategy you can use to generate a decision tree for this data-set? You can choose one of the strategies in this assignment or you can think of any other strategies.

Explain your strategy and justify why it is your best choice.

Marking Scheme:
(2 marks) A reasonable explanation