CS 486/686 Assignment 2 (128 marks)

Tianlu Feng

Due Date: 11:59 pm ET on Wednesday, June 30, 2021

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.

Tianlu Feng

Tianlu Feng

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)
  - Ethan Ward (e7ward@uwaterloo.ca)

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

Learning goals

Constraint Satisfaction Problem

- Formulate a given problem as a constraint satisfaction problem. Define the variables, the domains, and the constraints.
- Trace the execution of the AC-3 arc consistency algorithm on a given CSP.

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 Wooden Stick Grid Puzzle (38 marks)

We have a 4 by 4 grid and 16 wooden sticks of 4 different lengths. The possible lengths are 1, 2, 3, and, 4. There are 4 wooden sticks of each length. Our goal is to place the wooden sticks vertically on the grid while satisfying the following constraints:

- (All Different Constraints) In each row, the lengths of the sticks are all different. In each column, the length of the sticks are all different.
- (Viewing Constraints) There is a number on each side of a row or a column. This number describes the exact number of different sticks that a person can see if the person views the sticks from that side of the row or column. (This is a 3D puzzle.)

  See Figure 1 for an example. The number on the left side of a row is 2 and the number on the right is 3. [2 4 3 1] is a valid placement for this row. From the left, a person can see the sticks 2 and 4. From the right, a person can see the sticks 1, 3, and 4.

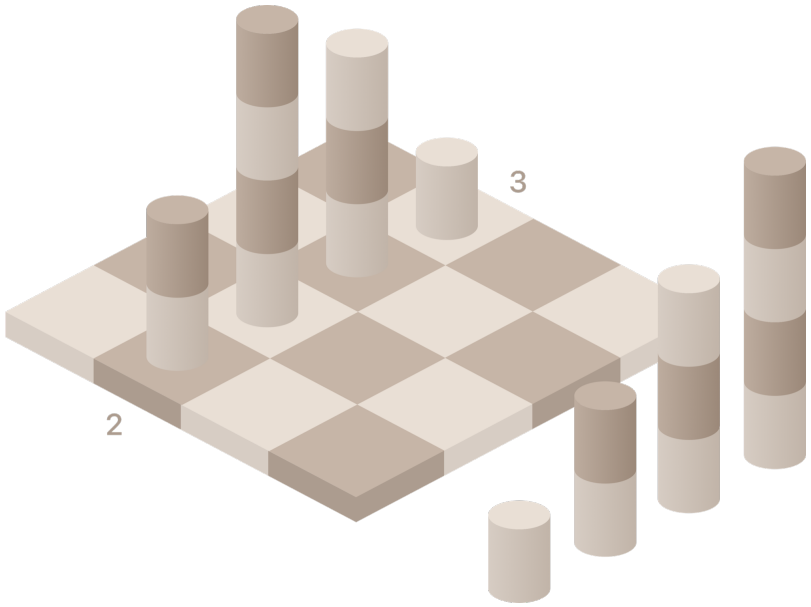

Figure 1: Stick puzzle example

Complete the following tasks using Table 1.

(a) Solve the puzzle in Table 1 by hand. Submit the final answer.

Marking Scheme: (2 marks) Correct solution.

	2	3	3	1	
2					1
1					3
3					2
2					2
	3	1	2	2	

Table 1: Wooden Stick Grid Puzzle Example 1

Solutions:

	2	3	3	1	
2	3	2	1	4	1
1	4	1	3	2	3
3	2	3	4	1	2
2	1	4	2	3	2
	3	1	2	2	

(b) Formulate the puzzle in Table 1 as a constraint satisfaction problem. We have provided the variable definitions below. Describe the domains of the variables and the constraints. Feel free to use words or mathematical expressions.

Variables:

A variable represents the sequence of four numbers from left to right in a row or the sequence of four numbers from top to bottom in a column.

Let R_i denotes the variable for the i-th row from the top. $i \in \{1, 2, 3, 4\}$ R_1 denotes the top row.

Let C_j denotes the variable for the j-th column from the left. $j \in \{1, 2, 3, 4\}$. C_1 denotes the leftmost column.

Marking Scheme: (8 marks)

- (2 marks) Domains
- (6 marks) Constraints

Solutions:
Since each variable represents the sequence of four numbers.

- R_i, $C_j = \{a_1a_2a_3a_4\}$, $a_i \in \{1, 2, 3, 4\}$
- Domains of R_i and C_j are $\{1234, 1243, 1324, 1342, 1423, 1432, 2134, 2143, 2314, 2341, 2413, 2431, 3124, 3142, 3214, 3241, 3412, 3421\}$ with all different constraints

The constraints are mainly based on all different constraints and viewing constraints
for all different constraints, for R_i, $C_j = \{a_1a_2a_3a_4\}$, $a_i \neq a_j$ for $i \neq j$

for viewing constraints, let's define $L(x)$ such that it is the number of changes of current highest value from a_1 to a_4 with initial highest value = 0. For example, we have $R = 2143$, L(R) = 2 by the following steps:

- for $a_1 = 2$, $a_1 > max = 0$, count = 1, new max = 2
- for $a_2 = 1$, $a_2 < max = 2$, count = 1, max remains 2
- for $a_3 = 4$, $a_3 > max = 2$, count = 2, new max = 4
- for $a_4 = 3$, $a_4 < max = 4$, count = 2, max remains 4

Similarly, we define $R(x)$ to be the same functionality in reverse order. Thus we have the constraints below:

- $L(R_1) = 2, R(R_1) = 1$
- $L(R_2) = 1, R(R_2) = 3$
- $L(R_3) = 3, R(R_3) = 2$
- $L(R_4) = 2, R(R_4) = 2$
- $L(C_1) = 2, R(C_1) = 3$
- $L(C_2) = 3, R(C_2) = 1$
- $L(C_3) = 3, R(C_3) = 2$
- $L(C_4) = 1, R(C_4) = 2$

Intersection constraints: for R_i, C_j, a_j of $R_i = a_i$ of C_j

(c) Consider the puzzle in Table 1.

For each unary constraint on a variable, remove all the values in the variable's domain that violate the unary constraint. Afterwards, write down the updated domains of all the variables.

Marking Scheme: (8 marks)

- (4 marks) The domains for $R_1 \ldots R_4$.

- (4 marks) The domains for $C_1 \dots C_4$.

Solutions:

- $R_1 \in \{3124, 3214\}$
- $R_2 \in \{4132, 4231, 4312\}$
- $R_3 \in \{1243, 1342, 2341\}$
- $R_4 \in \{1423, 2413, 2143, 3142, 3241, 3412\}$
- $C_1 \in \{1432, 2431, 3421\}$
- $C_2 \in \{1324, 2134, 2314\}$
- $C_3 \in \{1243, 1342, 2341\}$
- $C_4 \in \{4123, 4213\}$

(d) Start with the reduced domains of the variables from the previous part. We will continue solving the puzzle using the AC-3 arc consistency algorithm.

Execute the AC-3 algorithm on the puzzle for 8 steps as described below.

For each step, we have provided the arc to be removed from the set S. list the values removed from the domain of the variable and whether we need to add any arcs back to S. We have completed step 1 as an example.

After the 9 steps, indicate whether the algorithm should terminate or not. If the algorithm should terminate, describe the outcome of the algorithm execution.

(1) Remove $\langle R_3, (R_3, C_1)\rangle$ from S.
Remove 1243, 1342 from the domain of R_3.
Add nothing back to S.

(2) Remove $\langle R_4, (R_4, C_1)\rangle$ from S.

(3) Remove $\langle C_1, (R_1, C_1)\rangle$ from S.

(4) Remove $\langle R_4, (R_4, C_1)\rangle$ from S.

(5) Remove $\langle C_4, (R_3, C_4)\rangle$ from S.

(6) Remove $\langle C_3, (R_4, C_3)\rangle$ from S.

(7) Remove $\langle C_2, (R_3, C_2)\rangle$ from S.

(8) Remove $\langle R_2, (R_2, C_2)\rangle$ from S.

(9) Remove $\langle R_1, (R_1, C_3)\rangle$ from S.

Marking Scheme: (18 marks)

- (2 marks) for each of the 8 steps.
- (2 marks) for the result after 8 steps.

Solutions:

- Remove $\langle R_3, (R_3, C_1)\rangle$ from S.
  Remove 1243, 1342 from the domain of R_3.
  $R_3 = \{2341\}$
  Add nothing back to S.
- Remove $\langle R_4, (R_4, C_1)\rangle$ from S.
  Remove 3142, 3241, 3412 from the domain of R_4.
  $R_4 = \{1423, 2413, 2143\}$
  Add nothing back to S.
- Remove $\langle C_1, (R_1, C_1)\rangle$ from S.
  Remove 1432, 2431 from the domain of C_1.
  $C_1 = \{3421\}$
  Add $\langle R_3, (R_3, C_1)\rangle$, $\langle R_4, (R_4, C_1)\rangle$ back to S.
- Remove $\langle R_4, (R_4, C_1)\rangle$ from S.
  Remove 2413, 2143 from the domain of R_4.
  $R_4 = \{1423\}$
  Add nothing back to S.
- Remove $\langle C_4, (R_3, C_4)\rangle$ from S.
  Remove 4123 from the domain of C_3.
  $C_3 = \{4213\}$
  Add nothing back to S.
- Remove $\langle C_3, (R_4, C_3)\rangle$ from S.
  Remove 1243, 2341 from the domain of C_3.
  $C_3 = \{1342\}$
  Add nothing back to S.
- Remove $\langle C_2, (R_3, C_2)\rangle$ from S.
  Remove 1324, 2314 from the domain of C_2.
  $C_3 = \{2134\}$
  Add nothing back to S.
- Remove $\langle R_2, (R_2, C_2)\rangle$ from S.
  Remove 4231, 4312 from the domain of R_2.
  $R_2 = \{4132\}$
  Add nothing back to S.

- Remove $\langle R_1, (R_1, C_3)\rangle$ from S.
  Remove 3124 from the domain of R_1.
  $R_1 = \{3214\}$
  Add nothing back to S.

After 9 steps, we get: $R_1 = \{3214\}, R_2 = \{4132\}, R_3 = \{2341\}, R_4 = \{1423\}, C_1 = \{3421\}, C_2 = \{2134\}, C_3 = \{1342\}, C_4 = \{4213\}$. However, the algorithm does not end since the set S is not empty

(e) We tackled the same problem with two methods: (1) solving it by hand, and (2) solving it by executing the AC-3 algorithm. How are these two approaches similar or different? Discuss your observations and thoughts in a few sentences.

Marking Scheme:

(2 marks) provide a reasonable discussion.

Solutions:

For these two methods, the similar part is how we use the constraints. We have the same idea to narrow the domain of each row and column. However, in (1) solving the problem by hand, I think it is for problems with trivial viewing constraints(it is easy to figure out the position of 4), it is better than AC-3 algorithm that we do not need so many steps to figure out the solution. But, if the viewing constraints are not trivial, solving problems by hand may cause some calculation errors and mistakes. Using AC-3 algorithm can guarantee to get the correct solution.

2 Decision Trees (90 marks)

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

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 expected information gain metric, and 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.

Information on the provided code

We have provided several files below.

- `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.

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. The order is given below.

`mcg, gvh, alm, mit, erl, pox, vac, nuc`.

For example, suppose that the information gain for the best split point for `mcg` and the best split point for `pox` 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 `mcg`.

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 10 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 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 all the packages below are available in our testing environment. If you want to use a different package, please post a question on Piazza.

- numpy (version 1.19.5)
- anytree (version 2.8.0)

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 1 second for part b1, within 20 seconds for part b2, and within 10 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 hard about how 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.

Solutions: The maximum depth of the tree is 20. The related code and result is attached as "q2b1.txt"

(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 $[0, 30]$ for the maximum depth.

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

Solutions: the best maximum depth for pre-pruning is 7. The related code and result is attached as "q2b2.txt"

(3) Suppose that we want to post-prune `tree-full` using the minimum number of examples criterion.

For post-pruning, grow a full tree first. Define a value for the minimum number of examples. 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 number of examples at the node 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 number of examples at every `leaf parent` node is greater than or equal to the pre-defined value.

What is the best value for the minimum number of examples through ten-fold cross-validation? Use the range $[0, 300]$ with an increment of 20 for the minimum number of examples.

Marking Scheme:
(1 mark) Correct value of the best minimum number of examples for post-pruning.

Solutions: the best minimum number of examples for post-pruning is 60. The related code and result is attached as "q2b3.txt"

(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

Solutions: From the previous parts, the best accuracy for both pre-pruning and post-pruning are 0.5807965860597439.
I would like to choose post-pruning since the implementation of the assignment, we only need to build the decision tree once for cross-validation. We can reuse the decision tree with post-prune function in increasing order. It guarantees the accuracy and have a great performance.