ITI 1121. Introduction to Computing II
Winter 2021

Assignment 4
(Last modified on March 22, 2021)

Deadline: April 9, 2021, 11:30 pm

Learning objectives

- Working with linked structures
- Working with generics
- Recursive programming
- Exception handling

Introduction

We have already laid most of the groundwork for building decision trees and, at this point, we are only one well-crafted (recursive) algorithm away from building these trees! The description for Assignment 4 is shorter than those for the previous assignments, since we have covered most of what we need in these previous assignment descriptions. The amount of coding involved in Assignment 4 is also going to be less than in the previous assignments. **However, the smaller implementation does *not* mean that you should postpone it!** Assignment 4 requires familiarity with both (simple) **tree-based linked structures** as well as **recursive programming**. These concepts need time to digest. We therefore ask that you start working on the assignment **as soon as you receive this description.**

You will be implementing the following three tasks in Assignment 4. The tasks marked with a * will take more time to complete.

Task 1*.

Your first task is to build a decision tree for a given dataset. The implementation will be done in the `DecisionTree` class (whose template code has been provided to you). The nodes of the decision tree will be instances of the following (private) nested class in `DecisionTree`.

```
private static class Node<E> {
        E data;
        Node<E>[] children;

        Node(E data) {
                this.data = data;
        }
}
```

The `DecisionTree` class will instantiate this generic class with `VirtualDataSet` class as type. All the tree nodes that you will be working with are therefore instances of `Node<VirtualDataSet>`. The `DecisionTree` has an instance variable, named `root`, which maintains a reference to the root of the tree:

```
Node<VirtualDataSet> root;
```

The pesudo-code for the recursive `build(...)` method that you need to write is as follows:

```
private void build (Node<VirtualDataSet> node) {

    1. Edge cases:
        1.1. Are node and node.data non-null?

        1.2. Does node.data (virtual dataset) have at least one attribute?

        1.3. Does node.data have at least one datapoint?

    2. Base cases:
        2.1. node.data has only one attribute (this attribute has to be
             the "class" attribute; we can't split the data any further)

        2.2. The unique value set of node.data's last attribute, i.e., "class" attribute,
             has only one value in its unique value set (there is no uncertainty;
             we don't need to split any further)

        2.3. No (non-class) attribute of node.data has more than one unique value
             (there is nothing left to split on)

    3. Recursive case:
        3.1. Let a_max be the attribute in node.data that yields the
             best information gain

        3.2. Let [partition_0, ..., partition_(n-1)] be the result of splitting
            node.data over a_max (note: whether to apply numeric or nominal splitting
            depends on whether a_max is numeric or nominal)

        3.3. Instantiate the children array of node

        3.4. Populate node.children with the partitions resulting from Step 3.2.
             (note: children[0] will point to partition_0, children[1] to
              partition_1 and so on)

        3.5. (Recursive step) For every node.children[i] (0 <= i <= n-1) do:
             build(node.children[i])
}
```

Task 2*.

Write a `toString()` method that provides an if-else representation of the decision tree. The `toString()` method will call a recursive `toString(...)` method with the following signature:

```
private String toString(Node<VirtualDataSet> node, int indentDepth)
```

We want the if-else representation of a decision tree to be properly indented for readability. As we traverse deeper into the decision tree, the amount of indentation, captured by the `indentDepth` parameter, therefore has to increase (e.g., by one tab or one space). In the `DecisionTree` class, you have been provided with a simple method to create the desired indent to use as prefix for different depths during the traversal of the tree.

IMPORTANT: JDK 12 and upward provide an `indent(int n)` method for Strings. You are NOT allowed to use this method, as this will necessitate JDK 12+ for compiling your program. For marking, you can expect the TAs to be using JDK 11 but not higher. If your program does not compile due to using String's new `indent(int n)` method, you are solely responsible for any marks deducted because of non-compilation.

We now illustrate the output returned by the `toString()` method of the `DecisionTree` class. Consider the `main(...)` method below:

```
public static void main(String[] args) throws Exception {
        ActualDataSet data1 = new ActualDataSet(new CSVReader("weather-nominal.csv"));
        DecisionTree dtree1 = new DecisionTree(data1);

        System.out.println("*** Decision tree for weather-nominal.csv ***");
        System.out.println();

        System.out.println(dtree1);

        System.out.println("*** Decision tree for weather-numeric.csv ***");
        System.out.println();

        ActualDataSet data2 = new ActualDataSet(new CSVReader("weather-numeric.csv"));
        DecisionTree dtree2 = new DecisionTree(data2);

        System.out.println(dtree2);

}
```

The output generated by the above `main(...)` method is as follows:

```
*** Decision tree for weather-nominal.csv ***

if (outlook is 'sunny') {
  if (humidity is 'high') {
    class = no
  }
  else if (humidity is 'normal') {
    class = yes
  }
}
else if (outlook is 'overcast') {
  class = yes
}
else if (outlook is 'rainy') {
  if (windy is 'FALSE') {
    class = yes
  }
  else if (windy is 'TRUE') {
    class = no
  }
}
```

Output continued on the next page ...

```
*** Decision tree for weather-numeric.csv ***

if (outlook is 'sunny') {
  if (humidity <= 70) {
    play = yes
  }
  else if (humidity > 70) {
    play = no
  }
}
else if (outlook is 'overcast') {
  play = yes
}
else if (outlook is 'rainy') {
  if (windy is 'FALSE') {
    play = yes
  }
  else if (windy is 'TRUE') {
    play = no
  }
}
```

To be able to implement the `toString(Node<VirtualDataSet> node, int indentDepth)` method properly, you need to take note of three factors:

1. In Assignments 2 and 3, we did not keep track of the split condition in the virtual datasets resulting from partitioning. Examples of split conditions in the above output are:

   - `outlook is 'sunny'`
   - `windy is 'FALSE'`
   - `humidity <= 70`
   - `humidity > 70`

   Compared to the reference implementations you were provided with for Assignments 2 and 3, the template code for Assignment 4 has a slightly updated `VirtualDataSet` class. Specifically, `VirtualDataSet` now keeps track of the split condition that induced the dataset during the partitioning process. You can now obtain the split condition associated with a (virtual) dataset by simply calling the newly added `getCondition()` method. You will therefore not need to change `VirtualDataSet`.

2. The `build(...)` method in Task 1 merely manages the splitting process and stops it where the process cannot or should not continue. That method, however, does not ascribe a decision to the leaf nodes of the decision tree. The decision (verdict) for each leaf node is computed by the `toString()` method. For `weather-nominal.csv`, the two possible decisions are: (i) `class = no` and (ii) `class = yes`. For `weather-numeric.csv`, the two possible decisions are: (i) `play = no` and (ii) `play = yes`.

   If the dataset is noisy or the attributes not chosen properly by the data scientists, the leaf nodes in the decision tree may have datapoints that disagree on their "class" attribute. In other words, one may have a *mix of yeses and noes* in the leaves. In our `weather-nominal` example, for instance, we could have had situations where both `class = no` and `class = yes` are supported by the datapoints remaining in a given leaf node. For the purposes of this assignment, if multiple decisions are supported by a leaf node, `toString(...)` can arbitrarily pick either of them[1].

Note: Your implementation should work on `credit-info.csv` and `diabetes.csv`, but we withhold the output for these two datasets. When we mark Assignment 4, the teams whose implementations produce correct output for `credit-info.csv` and `diabetes.csv` will get a 10% bonus on their Assignment 4 mark!

[1] The reference implementation that you will receive later for Assignment 4 simply returns, for each leaf node, the first value in the unique value set of the "class" attribute. More nuanced implementations are possible but are beyond the scope of this assignment.

Task 3.

In this task, you will implement exception handling for the methods in `DecisionTree`, as well as for the methods in the three classes you developed in Assignment 3. **Specifically, the methods in the following four classes require proper exception for all their edge cases:**

- DecisionTree.java (from the current assignment)
- EntropyEvaluator.java (from Assignment 3)
- GainInfoItem.java (from Assignment 3)
- InformationGainCalculator.java[2] (from Assignment 3)

For Task 3, we do *not* anticipate that you will need to define any new exception classes. **The exception classes already provided by Java should suffice.** In particular, the following exception classes are probably all that you need: `IOException`, `IllegalArgumentException`, `ArrayIndexOutOfBoundsException`, `IllegalStateException` and `NullPointerException`.

Implementation

Like in previous assignments, you **cannot change any of the signatures of the methods. You cannot add new *public* methods or variables either.** You can, however, add new *private* methods to improve the readability or the organization of your code.

Guidance is provided in the template code in the form of comments. For the `DecisionTree` class, the locations where you need to write code have been clearly indicated with an inline comment that reads as follows:

```
// WRITE YOUR CODE HERE!
```

For Task 3 (exception handling), *you* need to decide where and how to update the code, based on what you have learned during the lectures and the labs. No guidance is provided in the code for exception handling.

Academic Integrity

This part of the assignment is meant to raise awareness concerning plagiarism and academic integrity. Please read the following documents.

- Academic regulation I-14 - Academic fraud
- Academic integrity

Cases of plagiarism will be dealt with according to the university regulations. By submitting this assignment, you acknowledge:

1. I have read the academic regulations regarding academic fraud.
2. I understand the consequences of plagiarism.
3. With the exception of the source code provided by the instructors for this course, all the source code is mine.
4. I did not collaborate with any other person, with the exception of my partner in the case of team work.
    - If you did collaborate with others or obtained source code from the Web, then please list the names of your collaborators or the source of the information, as well as the nature of the collaboration. Put this information in the submitted README.txt file. Marks will be deducted proportional to the level of help provided (from 0 to 100%).

[2] The `main(...)` method in `InformationGainCalculator` is not relevant to Assignment 4. You do *not* need to implement exception handling for `InformationGainCalculator.main(...)`. Notice, however, that you do need to implement exception handling for `DecisionTree.main(...)`.

Rules and regulation

- Follow all the directives available on the assignment directives web page.
- Submit your assignment through the on-line submission system virtual campus.
- You must preferably do the assignment in teams of two, but you can also do the assignment individually.
- You must use the provided template classes below.
- If you do not follow the instructions, your program will make the automated tests fail and consequently your assignment will not be graded.
- We will be using an automated tool to compare all the assignments against each other (this includes both, the French and English sections). Submissions that are flagged by this tool will receive the grade of 0.
- It is your responsibility to make sure that Brightspace has received your assignment. Late submissions will not be graded.

Files

You must hand in a **zip** file (no other file format will be accepted). The name of the top directory has to have the following form: **a4_3000000_3000001**, where 3000000 and 3000001 are the student numbers of the team members submitting the assignment (simply repeat the same number if your team has one member). The name of the folder starts with the letter "a" (lowercase), followed by the number of the assignment, here 4. The parts are separated by the underscore (not the hyphen). There are no spaces in the name of the directory. The archive a4_3000000_3000001.zip contains the files that you can use as a starting point. Your submission must contain the following files.

- README.txt
  - A text file that contains the names of the two partners for the assignments, their student ids, section, and a short description of the assignment (one or two lines).
- ActualDataSet.java
- Attribute.java
- AttributeType.java
- CSVReader.java
- DataReader.java
- DataSet.java
- **DecisionTree.java** (Aside from exception handling, the new code you write in Assignment 4 is localized to DecisionTree.java)
- EntropyEvaluator.java
- GainInfoItem.java
- InformationGainCalculator.java
- StudentInfo.java (Make sure to update the file, so that the `display()` method shows your personal information).
- Util.java
- VirtualDataSet.java

Questions

For all your questions, please visit the Piazza Web site for this course:

- https://piazza.com/uottawa.ca/winter2021/iti1121/home

Last modified: March 22, 2021