Decision Trees

Red is from Purtilo’s website, Black is the outline for us

DONE:

* Delivery manifest

TODO:

* Intellectual Property
  + Grading
    - Clarify encryption - comment
    - Check if what written is right
* Acceptance test plan
  + Grading
    - Include more detail (as much detail as you can). Details are in the comments in the section
  + Website

* Cost estimate
  + Grading and website
    - Need to estimate how much time it will take to complete each task you outlined in the timeline

Delivery manifest. *This is basically a checklist to summarize clearly what we're getting and how we will know we got it. It is used at delivery to, well, check that we got what you promised.*

The group Decision Tree in the fall 2017 class of CMSC435 is building an automatic grading platform of decision trees for Dr. Jeffrey Herrmann in the mechanical engineering department at the University of Maryland. There are going to be two parts to the website: an online tool to for students to create and be graded on decision tree homework assignments, and that allows pictures of decision trees to be scanned in and graded.

The instructor of the class will have the ability to create decision tree assignments that the student has to complete by the due date indicated by the instructor. For each assignment, the instructor will be able to create a decision tree which we will establish as the correct answer. The backend grading algorithm will compare the true decision tree provided by the instructor and with the student submitted decision tree. The instructor will be able to see the grades and assignments for each of the students, and for each class as a whole. The students will be able to view assignments and complete them by using an online tool on the website for creating trees. The website will only be accessible to students/instructors that are authorized through CAS authentication. Students will only be able to join a class if the instructor has allowed them permission to join.

The grading algorithm would be able to grade decision trees submitted by the students constructively by providing feedback. The feedback would tell the student where in their submitted tree their answer is wrong or if critical criteria (nodes/edges missing, labels/probabilities correct) is missing. Additionally, the grading algorithm will be able to give partial credit for a tree and not just give either 100% or 0% credit on a submitted tree. In addition, the algorithm for grading will be able to at least tell synonyms for labels from a pre-approved workbank of which students could possibly give, and account them into the grading process.

Once translating an image of a decision tree to a data structure is complete, the application will be very easily integrated with the website. Right now the prototyping of the imaging algorithm is done in Python and this can be integrated with the backend using the “rupy” library. The only input to the imaging algorithm is an image and the only output is a XML file of the same structure as the online tool to create decision trees. This imaging tool will create a richer workflow for the instructor because they would be able to automatically grade decision trees on exams instead of doing it by hand.

* Overall synopsis: We are creating an online grading system for decision trees such that it will compare students answers with a given through provided by a professor.
* The instructor and students will all have different kind of accounts. THE WEBSITE TEAM NEEDS TO GO OVER THESE GOALS TO SEE IF THEY’RE FEASIBLE
  + Instructor:
    - The instructor will be able to see the grades and assignments for each student, class.
    - The instructor will be able to make assignments for students. For each assignment, the instructor will be able to create a decision tree which we will establish as the right answer
  + Students:
    - Students will be able to view assignments
    - Students will be able to complete assignments using an online tool for creating trees
  + Accessibility
    - The website will only be accessible for students/instructors that are authorized through CAS authentication. Students will only be able to sign on if the instructor allows them for a certain class
      * I.e. students will not be able to access the website until the instructor has granted access.
* The website will grade the decision trees submitted by the students **constructively.** THE GRADING TEAM NEEDS TO GO OVER THESE GOALS TO SEE IF THEY’RE FEASIBLE
  + Constructively
    - Algorithm would be able to determine if a student solution is correct with respect to the given solution by the instructor
    - Algorithm would be able to give partial credit for trees that are wrong and not just give 100% or 0% grades
      * Analyzes the subtrees of the decision tree?
      * **Need to clarify this with Herrmann**
    - Algorithm will be able to give feedback to the student why a solution is wrong
      * Indicates which nodes or edges might be wrong?
      * Output divergent points between the two compared trees
      * Grade feedback based on critical criteria (similarity, not having all nodes/edges, etc)
      * **Need to clarify this with the team**

Intellectual property statement. Make clear expectations for ownership or restrictions on this product (which thus includes understanding what limits are placed on components you rely upon in your build.) Note carefully how this information will affect your process. For example, data which are proprietary to the customer would not be something you could or would leave for team members in an open dropbox.

The intellectual property (website, grading algorithm, computer vision software) are owned equally by each member of the team as a whole. The product developed can only be used by students and instructors who both have access to CAS authentication. The prototyping phase has restricted access to exclusively Dr. Jeffrey Herrmann of the mechanical engineering department at the University of Maryland.

All of the intellectual property of the assignments are owned by the instructor, and any information of tthe instructor’s respective students is owned by the instructor. All grades automatically processed can be changed by the instructor at any time for any reason.

If data is lost on the website due to an unexpected event, the creators of the website are not liable for lost data. The creators of the website are not liable to contact the instructors if information is lost, stolen, or for any reason corrupted. In addition, the creators are not responsible for the illegal distribution of data in the event of cheating or plagiarism. Instructors are highly encouraged to save their assignments and student grades in another medium website or database.

* Each team member owns an equal share of the final product.
* This Product is meant for students and instructors who have CAS accessibility
* The user defined content on the website is owned by the instructor
  + All of the assignment questions are owned by the instructor
  + All students work is owned and used to the instructor's discretion
  + All grades automatically processed and can be changed by the instructor for any reason at any time
* If data is lost on the website due to an unexpected event, the creators of the website are not liable for lost data. Instructors are highly encouraged to save their assignments and student grades in another medium website or database.
* All data is stored on the server
  + We use both Devise and omniAuth, which are the most popular gems for authentication and user management in Rails, to identify the user. After we verified the users, we use secret key and database key to encrypt and protect data.
  + We may adopt encryption/key-rotation scheme to allow the safe storage of user account information in Reverb, Encrypted data is obfuscated and retrieved by a secret digital key, usually a sufficiently long, random string, such as, AES-256 is a highly recommend algorithm for this. Therefore, we may encrypt with AES and attr\_encrypted.
  + We may contact the instructor if all the information has been lost or stolen on the web server in case if something wrong happened.

Acceptance test plan. (We will sometimes informally refer to this as the validation plan.) The hard deadline for construction is as addressed later in this assignment statement, but you should pay particular attention to the fact that it is earlier than end of the semester. We insist on projects being built in time for them to be substantively used. (Yes, this is one of the ways we give you a chance to see the consequences of technical decisions you made earlier in the semester.) Thus, you should ensure that part of what you propose in the green light is a plan for how you will have the system validated independently after completion, and in time for a report of the exercise the delivered at end of the semester (deadline as noted.) (We will surely talk a lot in class about how this is done!)

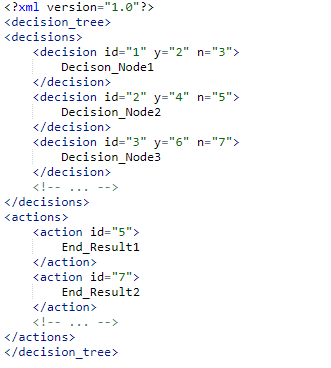
**Grading**

To begin for our grading team plan, our team viewed different samples of decision trees that were graded by our professor to gain some intuition into how validation of student produced trees was done, and different criteria that was considered for partial context. With this information we went forward with developing a schema that would assign a grade to a decision tree, and ensure that fair credit was delivered. The estimated cost for this portion is half a week to a week to familiarize ourselves with the professor’s grading procedures.

There are three main criteria that the grading algorithm will take into consideration when compared against a professor uploaded canonical solution.

1. Presence of all required nodes and edges (Percentage present from 0 to 100)
2. Similarity Estimation to canonical solution (Percentage from 0 to 100)
3. Performance (Various input -> Compare Output)

In addition to this, there will be an optional pre-processing step that will attempt to “collapse” both the canonical solution and the student tree to their respective minimal representation. Both the collapsed and uncollapsed tree will be compared, and the greater of the grades will be output. This step will be done in benefit of the student as the collapsing of the tree need not necessarily increase the given grade. Along with this step the Nokogiri Ruby library will be used to parse the nested XML data into a Ruby Decision Tree object, which the algorithm will operate over. Below is a sample decision tree XML file that will be parsed into a Ruby object for comparison. The estimated cost for this portion is a week to ensure we fully parse the XML decision tree into a workable Ruby object which contains all necessary fields that can be traversed efficiently.



The presence of all required nodes and edges will be done by iterating over the solution tree and populating a set with the correctly labeled nodes and edges. Then, the student solution is traversed in the same fashion, and the final components of each set compared. If each node present in the solution set is present in the student set, the student is given full credit for this section. If there are more than the required elements but the required elements are all present, the grade for this section will still be fully given. Otherwise, a percentage of the grade for this section will be given, scaled based upon the amount of missing nodes. The proposed grade weighting for this section will be **50%** of the final grade. This section should take approximately a week.

The similarity estimation will test the overall structure of the tree. The list of nodes and edges at will be pre-processed and sorted in alphabetical order to facilitate comparison. At each step, the being compared node will be checked against its corresponding node in the solution tree, checking for node type, name, along with amount and type of children and the contents of each edge. After traversing through the whole tree, a similarity score will be assigned out of 100, which will be the amount of credit the submitted tree receives for this section. The proposed grade weighting for this section will be **50%** of the final grade. This section should take approximately a week.

The final output of the algorithm will be the maximum of the two scores given by comparing the collapsed and uncollapsed submission with the canonical solution according to the criteria specified above. The grade is planned to be a floating point percentage rounded to two decimal places.

* + - We get some examples of how Herrmann graded some trees and compare our grading algorithm is to it. Herrmann said he expects it to be a harsh grader and he is fine with students complaining to a degree.
    - What we need is a range that is acceptable (maybe starting out as +-20%). We have talked to Herrmann about the necessity of this range, and while on multiple meetings he has agreed, he has still not finalized this item (as admittedly this is hard to finesse).
    - What is absolutely necessary is that the grading algorithm has to grade a perfect tree with 100% and anything else <100%.
    - Additionally must be able to “collapse” or reduce expanded tree into a generalized, standard answer that can be compared
      * Collapsing reduces similar nodes that connect with one common edge, and puts both nodes into one node
      * Allows for variety of trees to be standardized
      * Validation check must be matched to previously known Herrmann tests, but we will take every possible permutations of a known tree (and “expand” its form) to make sure the collapsing algorithm matches our “answer” tree
* CAS verification: we can use our eight people’s DID as verifiable examples. We use our own DIDs and passwords and input them into the UMD DID server, which contains all students and faculty, staff, affiliates for UMCP, to test if they match the return DID that server return to us. We also set up DID from one of us as the instructor DID to see if the desired person can access the instructor web pages. Furthermore, we may need an administration role, who can access and modify all the information in the database. We also verify his identity by using one of our own UMD DID.
* Website
  + - Generally, we use RSpec with Capybara to make general test for our application. For the Omniauth part, there’s no need to write tests for the functionality of gems, because it has been authorized by our umd server.

we also use a Factory to create an object with specific attributes you need in your test. The attributes that are significant to the test can be defined in the test and passed to the factory as parameters. One factory can create multiple objects, each with different attributes.

* + - * Interface testing: we mainly test for home pages, student pages, and instructor pages for detecting page logics and order
      * Functionality testing: we test the creation of tree, viewing tree, viewing student submitted information, and grading part.
      * Usability testing: we test the storage and correctness of database, including loading data, fetching data, and deleting data.
      * Compatibility testing: we test each API with our website, and enable each functionality of each API works.
    - FactoryGirl, we have the option of saving object to the database and testing a model’s validations or building your object in memory only . At the same time, FactoryGirl is the most popular gem for creating sample data in RSpec.
    - Security can be tested through attempting the standard “Big Ten” OWASP vulnerabilities
    - The decision tree sketch program will allow for users to easily create decision trees for online submission.
    - The tree sketch program will be its own black box coded in p5.js (which will be exported as javascript). When the sketching program is accessed it will open as a new window.
    - There will be a key displayed as an image on the top right of the window that will give a simple explanation about how to use the program. Users will be able to press the ‘d’ key (decision node), the ‘c’ key (choice node), and the ‘f’ key (final node) to switch between the creation of different types of nodes. Users will be able to click to create the first node (a decision node if they clicked the ‘d’ key, a choice node if they clicked the ‘c’ key, or a final node if they clicked the ‘f’ key.
    - After they create the first node, the user will be able click on a node to select it. Any clicks on empty space after selecting a single node will create a child node to that parent node at the location which was clicked. Upon creation, a line will be draw between each child node and its parent node.
    - Upon the creation of a node, the user will be able to input the text data for that node. The text data will be displayed as a text box above that node. Single clicking a node will selecting it and will highlight that node. If a node is double clicked, then the user will be able to edit the corresponding text for that node. The user will be able to click ‘enter’ to exit text editing mode for a single node. If a node is selected, pressing the ‘d’ key will delete it.
    - There will be a save and submit button for the user that will communicate with the database via xml file.
    - All node creation, node deletion, and node tree structure will be recorded/stored in a javascript data structure.
    - The data structure will consist of three nodes decision nodes, chance nodes, and final nodes.
    - Each node will store the necessary corresponding data for that node. Decision nodes and chance nodes will store one pointer to their parent node and a list of pointers to their children nodes.
    - The data structure will be equipped with an add, a delete method, a parse to xml method, and a parse from xml method.
    - Overall, the decision tree sketch program will be capable of creation of decision trees, storage of unfinished decision trees, submission of finished decision trees, and display of loaded decision trees.

Cost estimate. Make sure you have a clear and supported prediction of what it will take to get you from the point of the green light approval to the walk through (i.e. when the build is done.) To be clear: this is not an invitation to guess. Show us you can research various prediction methods which might be named in our supporting class materials, and then apply them here. This component of your submission should be sufficient to support its use later in the second half of a cost exercise, where you report how much effort the project did take, and then explain why they are the same or different. Presumably you would not want to submit a green light proposal with a cost estimate which predicts there isn't enough time left available to actually do the job, or give a plan which predicts peak burn rates might occur at times when you are heavily conflicted with other course or life obligations. Plan! (Note: if some of the cost of moving forward involves funds, not just effort, then this is also an important piece of the negotiation process. Hardware, supplies or service purchases are possible, but not assumed, so be sure this is also part of what we all work out.)

* Maintenance fee. After instructor and student begin to use it. We need find someone to keep updating the data and make sure they exist enough space for new data. We also need someone can directly access to the database to modify data if instructor or student need to change it.
  + Costs for one class is negligible and can be safely assumed to run fine within one year. Issue is if the data will need to be continually stored year after year, which would require us to migrate
* Time estimate for completion :: still need to do. Use the timeline as an outline
  + Grading
    - Anticipate lower cost when compared to website, as current team members already have preexisting similar Java code
    - Comparison will always require visiting all nodes of both trees, but allows comparison to be independent of collapsing of trees
      * Can Split tree/node comparison and collapsing into separate tasks, even though afterwards will be combined together
    - Comparison of synonyms can first be as simple as checking words within a wordbank (trivial), afterword will expand to using a wordnet on how closely two words are “related” to see if answer is correct
      * Can be separate from everything from above, as they only check nodes and tree structures and, not labels at each node/edge
    - Decision Tree grading familiarization (Done) - 1 week
    - Grading: Parse XML into traversable Ruby object - 1 week
    - Grading: Checking required elements presence - 1 week
    - Grading: Tree Similarity comparison - 1 week

Timeline. We will want you to call out at least the major tasks, story lines or activities necessary to complete your project, and show how composing them should lead to success. Then, as you proceed with the build, we will want you to maintain (e.g. on one of your groups' web sites) a graphic summary of the status of completion, so we can track progress. Are you getting closer or further away from success? The 'manager' will want to know at a glance. (We will talk more in class about a variety of tools for this, such as JIRA or Trello. We hope you will come to see such tools as aids to success, not busywork or barriers. Learn to use these to improve your productivity.)

* Sunday, October 29th
  + Website
    - Finish the CAS and Devise, set up database
    - Test the CAS and Devise database verification
* Monday, November 6th
  + Grading
    - Comparing trees absolutely
      * Either 100% or 0%
    - Parse into accurate tree representation
    - Collapsing the tree into its logically equivalent, most compact form
  + Website
    - Build general each web page structure
    - Test page structure by feed into independent data
* Friday, November 10th
  + Grading
    - Be able to compare sections of the tree for partial credit
    - Obtain a similarity score for both the collapsed and uncollapsed trees when compared to original
* Monday, November 13th
  + Website
    - Finish website sketch decision tree part and enable loading data from trees.
* Friday, November 17th
  + Grading
    - Be able to provide notes for the reasons why students got things off
  + Website
    - Integrate with the back-end algorithm part, so that they can receive information from database and sending information to database.
    - Enable to display student grade on our web page
* Friday, November 24th
  + Website
    - Integrate with the openCV part, so that we also can load information from images.
    - Testing all the feature and finding bugs and weakness
* Wednesday, November 29th
  + Website and Grading
    - Last minute touches on Project

Project advertisement. We will want a description of the project suitable for showing what 435 students do in the class, similar to (hopefully now better than) what you see at the SEAM site on the legacy tab. We recommend you add something soon as a mission statement and keep it up to date as the project moves forward, but at a minimum we want you to be able to tell the world what you did. Our ideal is that this remains as a persistent page, at least for some time following the semester, so you have something to link to during job searches. Remember: this is a formulation that fits into our server; preparing content on the assumption that you are free to stand up content on your favorite Erlang-on-rails engine might result in you being unpleasantly surprised later.

*Decision trees* team offers an online system that will facilitate the construction of decision tree assignments. It will allow the professor to post questions and correct answers to each corresponding question in decision tree assignments. Students will be able to view each assignment as well as each question on each assignment, and they will be able to construct, save, or submit a decision tree using an online interface. In addition, the professor will be capable of scanning in images of hand drawn decision trees for exams. Assignment submissions and scanned exam trees will be stored, constructively graded, and displayed along side their grade for the professor and the corresponding student to view.