CS540 – Assignment 4

CS540 – Fall 2015 – Sections 2 & 4
University of Wisconsin, Madison

Submission and Deadlines:

You will submit two separate files through Moodle for this assignment: (1) a .pdf file with your written answers to the theory questions, and (2) a .java file with your code implementation for the practical problems. Both of these files shall be submitted no later than **Noon on Friday, December4th** to receive full credit. Late submissions will be penalized 10% per 24 hours that they are received late. After five days (120 hours), late submissions will no longer be accepted for credit.

Collaboration Policy:

This entire assignment shall be completed individually. While you are encouraged to discuss relevant concepts and algorithms with classmates and TAs, you shall only use different examples and problems from those assigned below. Academic misconduct on this assignment includes, but is not limited to: sharing, copying, and failing to protect the privacy of your answers and code fragments with respect to the assignment described below.

Machine Learning (Theory)

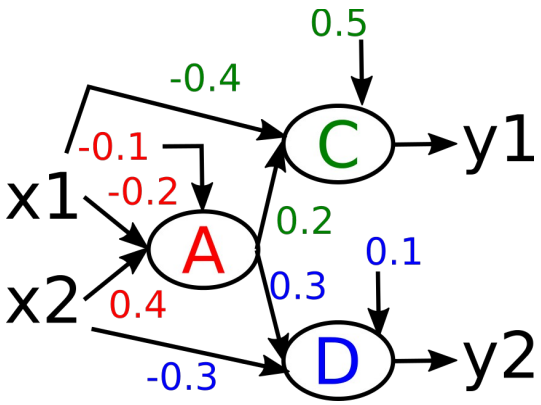
1. Suppose our learning algorithm has chosen a hypothesis from a set of 100,000,000 possibles. According to the PAC learning model, in order to be 99% confident that this hypotheses will generalize with an absolute error that is less than 5%, how many testing data points do will we need to validate our hypothesis against?

2. Calculate the slope (m) and y-intercept (b) of the line that best fits (minimizes the L2 loss) of the following points: (8, 9), (2,5), (6,6), (4,7), (5,7).

3. Assume that a single perceptron has the following weights: w0 = 0.2, w1 = 0.3, w2 = 0.4. What will the output of this perceptron be when the input is: x1 = 0.5, x2 = 0.6?

4. Continuing with the example of a single perceptron with weights w0 = 0.2, w1 = 0.3, and w2 = 0.4, how should these weights be adjusted when a learning rate α =0.1 is used with respect to a sample training point: x1 = 0.5, x2 = 0.6, and the expected output y is 1.0.

(5-7). For the next questions, consider the following neural network with three perceptons. The output of perceptron A feeds as input into both perceptrons C and D, and the weights of each perceptron are included in this diagram. Consider training this network with an input point x1 = 1, and x2 = 0, which has an expected output of y1 = 0, and y2 = 1.



5. What is the output of this artificial neural network when the input is (1,0)?

6. What is the delta associated with C and D when the input is (1,0), and the expected output is (0,1)?

7. What is the delta associated with A when the input is (1,0) and the expected output is (0,1)?

8. If a learning rate of $\alpha = 0.1$ is used, what will the new weights of node A be updated to as a result of training on this sample point?

Propositional Logic (Practical)

0. For this portion of the assignment, you will be implementing java class that first converts a propositional logic statement into conjunctive normal form, and then performs resolution to detect and report any resulting conflicts. Your implementation will be exclusively written into the provided class skeleton FIRSTNAME_LASTNAME_Resolution.java. After replacing the FIRSTNAME_LASTNAME portion of this filename with your own name, you will need to update the class and constructor names within this file, and the single references used to instantiate this class from Main.java. Your final solution should run without any other changes being made to Main.java, DrawableTree.java, or LogicParser.java. DO NOT add any package statements to any of these files, and DO NOT call exit from any of the methods that you are implementing.

   This code utilizes Processing version 3.0 and Antlr 4.5. The core.jar and antlr-4.5-complete.jar files for these versions of the libraries are available through the moodle page for this assignment. They should be added to your project and to your project's build path to run the provided code. You can test your setup by building and running the provided skeleton code. You will be prompted to enter a statement of propositional logic, and then be shown a graphical version of the resulting parse tree. Pressing the numbered keys 1-6, will verify that the stub methods in your FIRSTNAME_LASTNAME_Resolution.java class are being called by Main.

1. It will be helpful to start by implementing the conversion to conjunctive normal form (as described on pages 253, 254 of your text. The four steps in this process each correspond to a different method stub that you are required to implement: eliminateBiconditions(), eliminateConditions(), moveNegationInwards(), and distributeOrsOverAnds(). By setting the dirtyTree flag during each of these methods, you will be able to visually verify that these changes are implemented by testing them with a variety of propositional logic statements.

2. Once your logic is in CNF, I recommend next implementing some of the helper methods near the bottom of this class: isLiteralNegated(), getAtomFromLiteral(), clauseContainsLiteral(), setContainsClause(), and clauseIsTautology() are all methods that can be utilized in the implementation of the remaining collapse(), applyResolution(), and resolve() methods. While implementing these methods, here are some reminders about terminology in relation to our XML tree structures.

   - atom: a single named proposition with no children independent of whether it is negated
   - literal: either an atom-node containing a name, or a not-node with that atom as a child
   - clause: an or-node, all of the children of which are literals
   - set: an and-node, all of the children of which are clauses (disjunctions)

7. After converting your logic into CNF and implementing these helper methods, there is still a some pre-processing to do before we are ready for resolution: this work will be done through

the implementation of the collapse() method.

- The form we'd like our tree in for resolution, is a single and-node (under the logic-node root) that only has or-nodes as children. And each of these or-nodes should only have literals as children: either named atomic nodes or not-nodes with single named atomic node children. Since your tree is already in CNF, you can start by collapsing all of the binary and-nodes into a single and node. You can also collapse all of the binary or-nodes with their child or-nodes in a similar way. The last thing to watch out for is children of the and-node that are lone literals (not or-ed with any others). To get all of our disjunction in a common form (even those disjunctions with only a single literal), insert and or-node between the and-node and each of these lone literals.
- Next remove redundancy. Redundant literals and clauses will only cause more work for our resolution algorithm. So first remove redundant (repeated) literals from every clause, and then remove redundant clauses from the tree (children of the and-node). The clauseContainsLiteral() and setContainsClause() methods should be helpful in this process.
- Another optimization that you should implement at this stage is to remove clauses that are always true because they are tautologies. Make use of the clauseIsTautology() helper below for this purpose.

4. You are now ready to implement the final resolution algorithm within the applyResolution() and helper resolve() method stubs. When the applyResolution() method is called, it should continuously attempt to apply the resolution rule (by calling the resolve() method) on every pair of clauses in the tree. This should continue until either a conflict is detected, or no new resolvents can be found. When a conflict is detected, this method should return true. Otherwise this method should return false after adding all possible resolvents to the tree.

   The resolve method() takes two clauses as input, and attempts to resolve them using the resolution rule described on page 253 of your text. If resolution can be applied, your should remove any redundant literals from the resulting resolvent before returning it. If the resolution results in a conflict (empty disjuctions), you should simply return a clause without any literal children. If resolution cannot be applied to the provided clauses, then this method should instead return null.

5. Congratulations. Now that you have implemented each of the assigned methods, you are left only with the task of testing and validating the code that you have written.