CS-540 Homework Assignment #3.5: Decision Tree Implementation

Assigned: Saturday, March 4th
Due: Wednesday, March 29th

Hand-In Instructions

This assignment includes a programming portion in Java and a written problem. The programming portion may be done with a partner, while the written portion must be done individually. Hand in all parts electronically by uploading them in *a single zipped file* to the assignment page on Canvas.

For the programming problem, put *all* files needed to run your program, including ones you wrote and ones provided, into a folder called <wisc NetID>-HW3.5-P1.

For the written portion, add a PDF called <wisc NetID>-HW3.5-P2.

Every student should turn in at least the response problem and a README that mentions your username, your name, and the corresponding information of your partner, if you have one. If you don't have a partner, just write "I worked alone."

Once you are finished, put your programming component folder and your PDF file into a single directory. Zip it, name it <wisc NetID>-HW3.5, and upload it to the assignment Canvas page.

Make sure your program compiles on CSL machines this way! Your program will be tested using several test cases of different sizes.

Late Policy

All assignments are due **at 11:59 p.m.** on the due date. One (1) day late, defined as a 24-hour period from the deadline (weekday or weekend), will result in 10% of the <u>total points</u> for the assignment deducted. So, for example, if a 100-point assignment is due on a Wednesday and it is handed in between any time on Thursday, 10 points will be deducted. Two (2) days late, 25% off; three (3) days late, 50% off. No homework can be turned in more than three (3) days late. Written questions and program submission have the same deadline. A total of three (3) free late days may be used throughout the semester without penalty. Assignment grading questions must be raised with the instructor or a TA within one week after the assignment is returned.

Collaboration Policy

You are to complete this assignment individually or with at most one partner. However, you are encouraged to discuss the general algorithms and ideas with classmates, TAs, and instructor in order to help you answer the questions.

You are also welcome to give each other examples that are not on the assignment in order to demonstrate how to solve problems.

But we require you to:

• not explicitly tell each other the answers

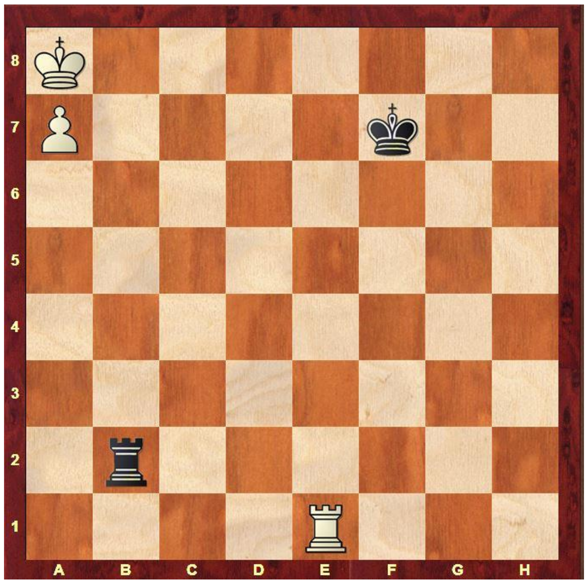
• not to copy answers or code fragments from anyone or anywhere

• not to allow your answers to be copied

• not to get any code on the Web

In those cases where you work with one or more other people on the general discussion of the assignment and surrounding topics, we suggest that you specifically record on the assignment the names of the people you were in discussion with.

Problem 1: [65] Fun with Chess (Decision Trees)



King+Rook vs King+Pawn Endgame

Goal: Learn a classifier that for some feature vector representation of a KR v. KP chess endgame returns "Win" if White can win the game from this position and "Yield" if White can stalemate or lose at best.

Imagine that you are a lazy chess player determined to *never* think through your endgames. Rook endgames are particularly common, and you've happened upon a database of K+R vs. K+P board positions (kindly provided by

Alen Shapiro back in the 1980's) labeled with final outcomes that you want to exploit through machine learning.

You want to know, given a board position encoded in a 35-feature vector, whether you can **win** from the current position (so you continue playing) or whether you **can't win** and are better off negotiating with the opponent for a draw.

In this problem, you are to implement a program that builds a decision tree for categorical attributes. You must build a tree from the training dataset and prune the learned decision tree using a tuning dataset.

You should implement five methods and one member for the class DecisionTreeImpl:

1: private DecTreeNode(root);
2: DecisionTreeImpl (DataSet train);
3: DecisionTreeImpl (DataSet train, DataSet tune);
4: public String classify (Instance instance);
5: public void rootInfoGain (DataSet train);
6: public double getAccuracy(DataSet ds);

DecisionTreeImpl(DataSet train) learns the decision tree from the given training dataset, while DecisionTreeImpl(DataSet train, DataSet test) not only learns the tree but also prunes it using the given tuning dataset. classify predicts the example's label using the trained decision tree.

rootInfoGain(DataSet train) prints the information gain (one in each line) for all the attributes at the root based on the train set. Finally, the root of your tree should be stored in the member root we have declared for you. The next sections describe other aspects in detail.

You may choose to restructure the return types and arguments of the DecisionTreeImpl code, but make sure the IO is consistent with the provided test cases. You must implement the ID3 algorithm in the slides and textbook.

Dataset

Even though IO is handled for you, I would recommend looking through the provided training and tuning sets to get a feel for the underlying data.

The data is in the following form.

```
// Description and comments about the dataset
%%, Class1,Class2
##, FeatureName,FeatureVal1,FeatureVal2
```

Implementation Details

Predefined Data Types
We have defined four data types to assist your coding, called Instance, DataSet, DecTreeNode and DecisionTreeImpl. Their data members and methods are all commented, so it should not be hard to understand their meaning and usage. You may choose to not use the DecTreeNode or DataSet classes in favor of your own structure for ID3 implementation, but the IO related code in HW3 should remain unchanged.

Building the Tree
In both DecisionTreeImpl methods you are first required to build the decision tree using the training data.

You should write a recursive function corresponding to the DecisionTreeLearning function in the textbook or the buildtree function in the lecture slides. As long as IO is left untouched, feel free to add helper classes.

When deciding the values at leaf nodes, you will have to determine the majority class. If there is a tie between two classes for the majority, pick the one that appears first in the training file, in other words: the one with the lowest index.

Pruning
For the constructor DecisionTreeImpl(DataSet train, DataSet tune), after building the tree you also need to prune it using the tuning dataset and the iterative Pruning Algorithm given in the lecture slides. I've copied it here with some implementation specifics for those interested.

```
Prune(tree T, TUNE set):
    //While Tuning Set Accuracy Continues to Not Decrease
    Acc(T) = Accuracy(Tuning Set on Tree T)
    For every internal node N in T:
        TN = copy of T, but prune (delete) the subtree under N
        N becomes a leaf node in TN.  The class label for N is the majority
        vote of TRAIN examples reaching N
        Acc(TN) = TN's accuracy on TUNE
    Let T* be the tree (among the TN's and T) with the largest A
    If Acc(T*) >= Acc(T):
        Set T = T* /* prune */
    Else:
        Break
```

In order to change the label of an internal node to be a leaf node with the class of the majority of the training examples that reach that node, you may also need to add some extra parameters to the function Prune.

Classification
public String classify(Instance instance) takes an example (called an instance) as its input and guesses the string label of the previously-built decision tree. *You do not need to worry about printing*. That part is already handled in the provided code.

Printing and Information Gain at the Root
The only printing you need to do is in the method public void rootInfoGain(DataSet train).

For each attribute print the output one line at a time: first the name of the attribute and then its information gain achieved by selecting that attribute at the root. The output order of the attributes and associated gains must be the same as the order that the attributes appear in the training set's header.

An example is given in the file output_chess0.txt. Print your results with 5 decimal places using System.out.format("%.5f", arg)

Testing

Do not hardcode your program to the chess dataset. We will test your program *using several training, tuning, and testing datasets, and the format of testing commands will be:*

java HW3 <modeFlag> <trainFile> <tuneFile> <testFile>where trainFile, tuneFile and testFile are the names of the training, tuning and testing datasets, respectively.

modeFlag is an integer from 0 to 4, controlling what the program will output:
0: Print the information gain for each attribute at the root node based on the training set
1: Create a decision tree from the training set and print the tree
2: Create a decision tree from the training set and print the classification for each example in the test set
3: Create a decision tree from the training set, prune it using the tuning set, and then print the tree
4: Create a decision tree from the training set, prune it using the tuning set, and then print the classification for each example in the test set

To facilitate debugging, we have also provided sample input files and their corresponding output files. They are called chess_train.txt, chess_tune.txt and chess_test.txt for the input, and out_chess0.txt to out_chess4.txt for the output.

An example command:

```
java HW3 1  chess_train.txt  chess_tune.txt  chess_test.txt
```

You are NOT responsible for any file input or console output other than public void rootInfoGain (DataSet train). We have written the class HW3 for you, which will load the data and pass it to the method you are implementing.
As part of our testing process, we will unzip the file you submit to Moodle, remove any class files, call javac HW3.java to compile your code, and then call the main method HW3 with parameters of our choosing.

Make sure that your code runs on one of the computers in the department because we will conduct our tests on these machines.

Deliverables
Hand in your modified version of the code skeleton we provided to you along with your implementation of the decision tree algorithm. Also include any additional java class files needed to run your program. You are permitted to write your own helper classes.

Once you are satisfied with your decision tree implementation, you (and or your partner) should perform separate writeups of Problem 2.

Problem 2: [10] Decision Tree Interpretation

a) [3] Encoding a single chess board position into 35 features takes time. Using the results of running the information gain heuristic at the root of the tree on chess_training.txt, which feature would you recommend removing from the feature vectors to minimize negative impact on accuracy? Why?

b) [3] What is the percentage *decrease* in *error rate* between the unpruned decision tree's performance on the test set and the pruned tree's performance on the test set. Justify your answer by referencing both figures.

c) [4] Explain why overfitting can be positively correlated to the number of the nodes in decision trees built on the same training data. Why does the pruned tree overfit less?