CS 5369 TEXAS STATE UNIVERSITY

Random and c4.5 Decision Tree Performance Comparisons

Occam's Razor Test

Luke Paireepinart

2/18/2010

Detailed description of performance measurements for Random and Decision trees; Evidence that Occam's Razor holds true for Decision Trees (that shorter trees tend to be more accurate.)

Contents

Introduction 2
What I Did and What I Reused 3
Data Sets 5
Dependencies 6
    Python 6
    Environment Variables 6
    Orange 7
Running the Program 9
Program Results 10
Analysis of Results 17
Conclusions 20
    Final Remarks 20

Introduction

Decision trees are valuable in many situations. They are popular in Data Mining, and are commonly used for classification of attributes. Given a training set, the decision tree can "learn" which attributes are most likely to determine which class a specific object is part of.

The first, and simplest, approach to a decision tree is simply to place every item in the tree, and split them up by a random attribute at each level. So, given a data set containing 3 items, Height, Width, and Length, and a class that we want to determine, Weight, we can split the tree on any one of these. First we may choose "Width", then "Height", and then "Length", or we may choose "Height" then "Length" then "Width". However, it is likely that one of these attributes is more influential on a particular datum's classification than the others. So the order that we choose to split attributes when we classify may have an effect on the accuracy of our classification.

Supposing our assumption above is true, that the order we choose to split attributes on when we classify will influence, in some way, our resultant accuracy, we must simply come up with a hypothesis that postulates this is the case, and after testing it we can prove whether or not the hypothesis holds.

The postulation we will make is this: "The greater the height of a tree, the worse it will be as a classifier." The reasoning behind this formulation is simple to follow: if a tree is split on its most determining attributes, then any item being classified will end as soon as possible, therefore making the tree shorter.

In order to test our postulation, we implement two decision trees: one c4.5 decision tree and one random decision tree. The random decision tree does not take into account the attributes' "gain ratios" (the "determining attributes" mentioned earlier), while c4.5 does. Therefore, we are hoping that c4.5 will:

1) Generate shorter trees
2) Classify more accurately

If these two items prove to be true, then we will have proven our hypothesis.

What I Did and What I Reused

My initial plan was to write the whole assignment from scratch. I lost a good 15-20 hrs of work down this path before I finally gave up.

Then I decided to use Orange since I was familiar with Python already. However I did not expect the complexity of the Orange API. Because Orange was originally designed with C++ in mind and they just glued Python on top of it later, it has a lot of quirks that do not make sense. In the python community Orange would be known as "non-pythonic". It still uses regular Python but the way methods and classes are overridden is very convoluted and difficult to understand.

I originally intended to modify the Orange builtin Tree data type to perform random selection. I tried a multitude of different approaches – I tried overriding the NodeType of the Tree to be a RandomClassifier, I tried passing RandomClassifier to one of the node constructors, I

I spent a few hours debugging a problem that it turns out was Orange's fault as well: normally this is valid in python:

```
>>> a = ['hello', 'how', 'are', 'you?']
>>> a[2:]
['are', 'you?']
>>>
>>> a = "hello how are you?"
>>> a[6:10]
'how '
>>>
```

If you notice, the syntax a[val:val2] will take a "slice" out of the list. This works for any type of Sequence object (strings, lists, lists of lists, tuples, or any classes derived from these, etc.). **HOWEVER** even though Orange's datatypes inherit from 'list' and even **allow** you to use the slice syntax, they do not return the correct data. It is duplicated in areas it shouldn't be, but very subtly so. It took me many hours to track it down and fix it.

Basically I am really frustrated with the way Orange's data types work, and I will probably try to avoid using it in the future unless I have to. I just got too invested in it and the opportunity cost of going back to the scratch solution was too high at that point so I just finished the project in Orange.

So what did I end up using from Orange and what did I write myself?

1) The file 'fixdata.py' that processes the data is written by me.
2) The file 'c45tree.py' contains functions that were modified by me, but I got them from the Orange tutorials page.
3) Main.py was written by me, it is the driver function.
4) Randtree.py was written by me completely.

Orange is used for data input and for everything involving c4.5. For the random tree it was all written by me, even the cross-validation. I could not use the Orange method for cross-validation because its types

were not compatible with my random tree, because it's basically really hard to derive from their classes and still build the trees the way I wanted to. I really wanted to get their cross-validation routine from orngTest to work, but it would have been too much work. However I was able to use their cross-validation routine for the c4.5 since it's their own builtin class.

Data Sets

The data sets used were gathered from the UCI repository located at http://archive.ics.uci.edu/ml/ . Only the ones for Classification were used, and only those with categorical attributes and no missing values. Also, a few datasets were eliminated just because they didn't follow the conventional formats of the majority, so would be harder to convert over. The chosen data sets cover the data sizes quite well (they range in number of attributes from around 4 to around 35) and they cover data set sizes well (from 2kb of data up to 4 megabytes), and the gain from including the few extra datasets seemed limited.

The included datasets are those below:

Adult+stretch, Adult-stretch, Yellow-small, Yellow-small+adult-stretch, Tic-Tac-Toe, Nursery, Kr-vs-Kp, Connect-4, Car, and Balance.

The format of the data sets that were used from the source website is: R,1,1,3,3

That is to say, the values are comma-separated. In order to convert the files to the format used in my program, I have included the program fixdata.py. So if you would like to run a dataset that is not included in the deliverable, simply place it in the datasets folder and run fixdata.py on it. An example session should make its usage quite clear:

```
F:\deliverable\datasets>dir
 Directory of F:\deliverable\datasets
02/22/2010  03:13 AM    <DIR>          .
02/22/2010  03:13 AM    <DIR>          ..
02/20/2010  02:46 PM             6,998 balance.tab
06/19/1997  12:32 PM            51,867 car.data
02/21/2010  02:50 PM             1,697 fixdata.py
02/21/2010  02:50 PM         1,072,496 nursery.tab
               4 File(s)      1,133,058 bytes
               2 Dir(s)  127,493,869,568 bytes free

F:\deliverable\datasets>python fixdata.py car.data car.tab
There are  7  attributes.
type the attributes in order, comma-separated, on the next line.
: buying, maint, doors, persons, lug_boot, safety, will_buy
Which attribute is a class?
: will_buy

F:\deliverable\datasets>
```

And that's all there is to it! Now this dataset can be used with my program for testing. You are free to remove car.data after car.tab is generated.

Dependencies

My implementation uses Orange and Python. In order to use the program you must install both Orange and Python. If Python is already installed, Orange can be installed on top of it using one of the installers provided on the site. However, if Python is not already installed, you can simply get one of the installers for Orange that includes Python.

Python

I personally recommend you use python 2.6. It is the most stable version. You can download it from http://www.python.org . Please be sure not to visit python.com accidentally as it contains some not-safe-for-work material. You can download the installer from their page for any platform you're running on.

One problem you may run into is that sometimes the Python installer will not properly add itself to your system path. It is necessary to add Python to the path so that you can run Python programs simply by typing "python programname.py". If you start a command prompt (by clicking Start and then Run and typing 'cmd' and hitting Enter (in Windows Vista / 7 you click Start and then type 'cmd' and hit Enter)) and then type "python" you should get a prompt that looks like this:

```
C:\>python
Python 2.6.4 (r264:75708, Oct 26 2009, 08:23:19) [MSC v.1500 32 bit (Intel)]
on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

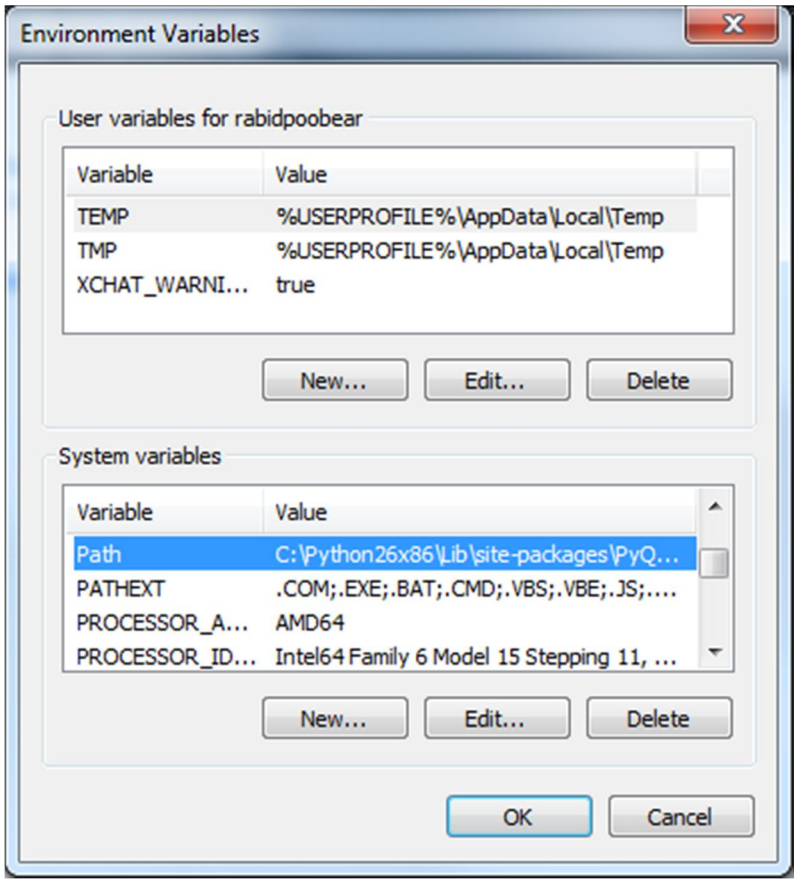
If you do not get this screen but instead get a "command not recognized", then I have written some directions for you below. If you are not using Windows you will have to google "setting environment variable path in osx" or "setting path environment variable in linux" to find a guide.

NOTE: if you get the prompt above, then DON'T bother doing the directions and skip the Environment Variables section and read the Setting up Orange section.

Environment Variables

In the case that you get an error when trying to type "python" at your command prompt, you will need to edit your environment variable to include the Python path. On Windows this is simple. If you are in Windows XP, press Windows-Key + Pause (hold down the windows key (the one between ctrl and alt) and press Pause key) and the System Preferences page will come up. Here you should click on the Advanced tab and you should see a button that says "Environment variables". Click on that button.

In Windows Vista / Windows 7, you do the same key combination, but you type "environment variables" in the "Search" box at the top right. "Edit the System Environment Variables" should be a search result. Click it, and then click the "Environment Variables" button. You will see this window (in both XP and Vista/7).



Notice that "Path" is highlighted in **System variables**. Highlight your Path and click "Edit". At the end of the **Variable Value:** field, add your Python directory. For example if Variable Value contains "C:\HelloWorld" then you would change it to "C:\HelloWorld;C:\Python26". **DO NOT** forget the semicolon! It is the separator for paths and the OS will not be able to interpret the path. Also do not use C:\Python26 if you have installed Python somewhere else. Save the changes and exit the screen. **NOTE**: If you already had a DOS window up, you must close and re-open it to get the new PATH environment variable.

Orange

You should be able to just install orange from the packages on the site. If you are planning on using other Python programs you should probably install Python and Orange separately, but if you are

only going to use Orange I would suggest you go ahead and just run the bundled “orange+python” installer. Obviously you should install orange for whatever platform you are currently using.

To test that orange is installed correctly, start Python and type “import orange”

```
C:\>python
Python 2.6.4 (r264:75708, Oct 26 2009, 08:23:19) [MSC v.1500 32 bit (Intel)]
on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import orange
>>> quit()

C:\>
```

In this case, I was able to import orange without any error message coming up. If I did not have orange installed correctly, I would have gotten a message that said ImportError: module “orange” was not found. Quit() just exits the Python interpreter when you’re done using it.

Running the Program

Running the program is very straightforward. If you would like the output of the program to go to the console, you can simply type

```
F:\deliverable> python main.py
Running... output will be in console.
============================================================
Dataset: balance
there are 625 data.
each example includes 5 attributes:

Left-Weight, Left-Distance, Right-Weight, Right-Distance, Balance

The class is: Balance
============================================================
Run 1 of Random Tree...
Accuracy: 0.256451612903
Height: 4.0


Run 2 of Random Tree...
Accuracy: 0.325806451613
Height: 4.0


Run 3 of Random Tree...
Accuracy: 0.298387096774
Height: 4.0


Total Accuracy Avg (over all 3 runs):  0.293548387097
Total Height  Avg  (over all 3 runs):  4.0

Run 1 of C45 Tree...
Accuracy: 0.643164362519
Height: 3.0
```

And the output continues so. However, if you want to save the output for later, you can simply type

```
F:\deliverable> python main.py out.txt
Running... output redirected to out.txt
Done...
F:\deliverable>
```

If you have any issues at all running the program, and you have tried the most common fixes (listed above in the Dependencies section) please contact me and I will walk you through setting it up.

Program Results

As was stated in the assignment document, the program was run on various data sets. The raw output of all used data sets follows. Skip this section if you do not care about the raw dump of the output.

```
=====================================================================
Dataset: adult+stretch
there are 20 data.
each example includes 5 attributes:

color, size, act, age, inflated

The class is: inflated
=====================================================================
Run 1 of Random Tree...
Accuracy: 0.9
Height: 3.2


Run 2 of Random Tree...
Accuracy: 1.0
Height: 3.6


Run 3 of Random Tree...
Accuracy: 1.0
Height: 3.8


Total Accuracy Avg (over all 3 runs):  0.966666666667
Total Height  Avg  (over all 3 runs):  3.53333333333

Run 1 of C45 Tree...
Accuracy: 1.0
Height: 2.0
```

```
=====================================================================
Dataset: adult-stretch
there are 20 data.
each example includes 5 attributes:

color, size, act, age, inflated

The class is: inflated
=====================================================================
Run 1 of Random Tree...
Accuracy: 1.0
Height: 3.2
```

```
Run 2 of Random Tree...
Accuracy: 0.9
Height: 3.4


Run 3 of Random Tree...
Accuracy: 1.0
Height: 3.9


Total Accuracy Avg (over all 3 runs):  0.966666666667
Total Height  Avg  (over all 3 runs):  3.5

Run 1 of C45 Tree...
Accuracy: 1.0
Height: 2.0




======================================================================
Dataset: balance
there are 625 data.
each example includes 5 attributes:

Left-Weight, Left-Distance, Right-Weight, Right-Distance, Balance

The class is: Balance
======================================================================
Run 1 of Random Tree...
Accuracy: 0.262903225806
Height: 4.0


Run 2 of Random Tree...
Accuracy: 0.246774193548
Height: 4.0


Run 3 of Random Tree...
Accuracy: 0.251612903226
Height: 4.0


Total Accuracy Avg (over all 3 runs):  0.25376344086
Total Height  Avg  (over all 3 runs):  4.0

Run 1 of C45 Tree...
Accuracy: 0.643164362519
Height: 3.0
```

```
======================================================================
Dataset: car
there are 1728 data.
each example includes 7 attributes:

Buying Price, Maintenance Price, Doors, Capacity, Trunk Size, Safety,
Desirability

The class is: Desirability
======================================================================
Run 1 of Random Tree...
Accuracy: 0.523709503966
Height: 6.0


Run 2 of Random Tree...
Accuracy: 0.495409329211
Height: 6.0


Run 3 of Random Tree...
Accuracy: 0.516678989111
Height: 6.0


Total Accuracy Avg (over all 3 runs):  0.511932607429
Total Height  Avg  (over all 3 runs):  6.0

Run 1 of C45 Tree...
Accuracy: 0.924754671327
Height: 5.0




======================================================================
Dataset: connect-4
there are 67557 data.
each example includes 43 attributes:

a1, a2, a3, a4, a5, a6, b1, b2, b3, b4, b5, b6, c1, c2, c3, c4, c5, c6, d1,
d2, d3, d4, d5, d6, e1, e2, e3, e4, e5, e6, f1, f2, f3, f4, f5, f6, g1, g2,
g3, g4, g5, g6, X Wins

The class is: X Wins
======================================================================
Run 1 of Random Tree...
Accuracy: 0.449996009797
Height: 41.0


Run 2 of Random Tree...
```

Accuracy: 0.443453010927
Height: 41.0

Run 3 of Random Tree...
Accuracy: 0.444296768966
Height: 41.0

Total Accuracy Avg (over all 3 runs): 0.44591526323
Total Height Avg (over all 3 runs): 41.0

Run 1 of C45 Tree...
Accuracy: 0.809390673487
Height: 20.8

==========
Dataset: kr-vs-kp
there are 3196 data.
each example includes 37 attributes:

bkblk, bknwy, bkon8, bkona, bkspr, bkxbq, bkxcr, bkxwp, blxwp, bxqsq, cntxt,
dsopp, dwipd, hdchk, katri, mulch, qxmsq, r2ar8, reskd, reskr, rimmx, rkxwp,
rxmsq, simpl, skach, skewr, skrxp, spcop, stlmt, thrsk, wkcti, wkna8, wknck,
wkovl, wkpos, wtoeg, Win State

The class is: Win State
==========
Run 1 of Random Tree...
Accuracy: 0.581480211599
Height: 36.0

Run 2 of Random Tree...
Accuracy: 0.652789968652
Height: 36.0

Run 3 of Random Tree...
Accuracy: 0.569877547022
Height: 36.0

Total Accuracy Avg (over all 3 runs): 0.601382575758
Total Height Avg (over all 3 runs): 36.0

Run 1 of C45 Tree...
Accuracy: 0.994372061129
Height: 12.6

```
======================================================================
Dataset: nursery
there are 12960 data.
each example includes 9 attributes:

parents, has_nurs, form, children, housing, finance, social, health, daycare

The class is: daycare
======================================================================
Run 1 of Random Tree...
Accuracy: 0.408262548263
Height: 8.0


Run 2 of Random Tree...
Accuracy: 0.351274131274
Height: 8.0


Run 3 of Random Tree...
Accuracy: 0.361081081081
Height: 8.0


Total Accuracy Avg (over all 3 runs):  0.373539253539
Total Height  Avg  (over all 3 runs):  8.0

Run 1 of C45 Tree...
Accuracy: 0.970216049383
Height: 7.0




======================================================================
Dataset: tic-tac-toe
there are 958 data.
each example includes 10 attributes:

top-left, top-middle, top-right, middle-left, middle-middle, middle-right,
bottom-left, bottom-middle, bottom-right, X Wins

The class is: X Wins
======================================================================
Run 1 of Random Tree...
Accuracy: 0.403782894737
Height: 8.0


Run 2 of Random Tree...
Accuracy: 0.449835526316
Height: 8.0
```

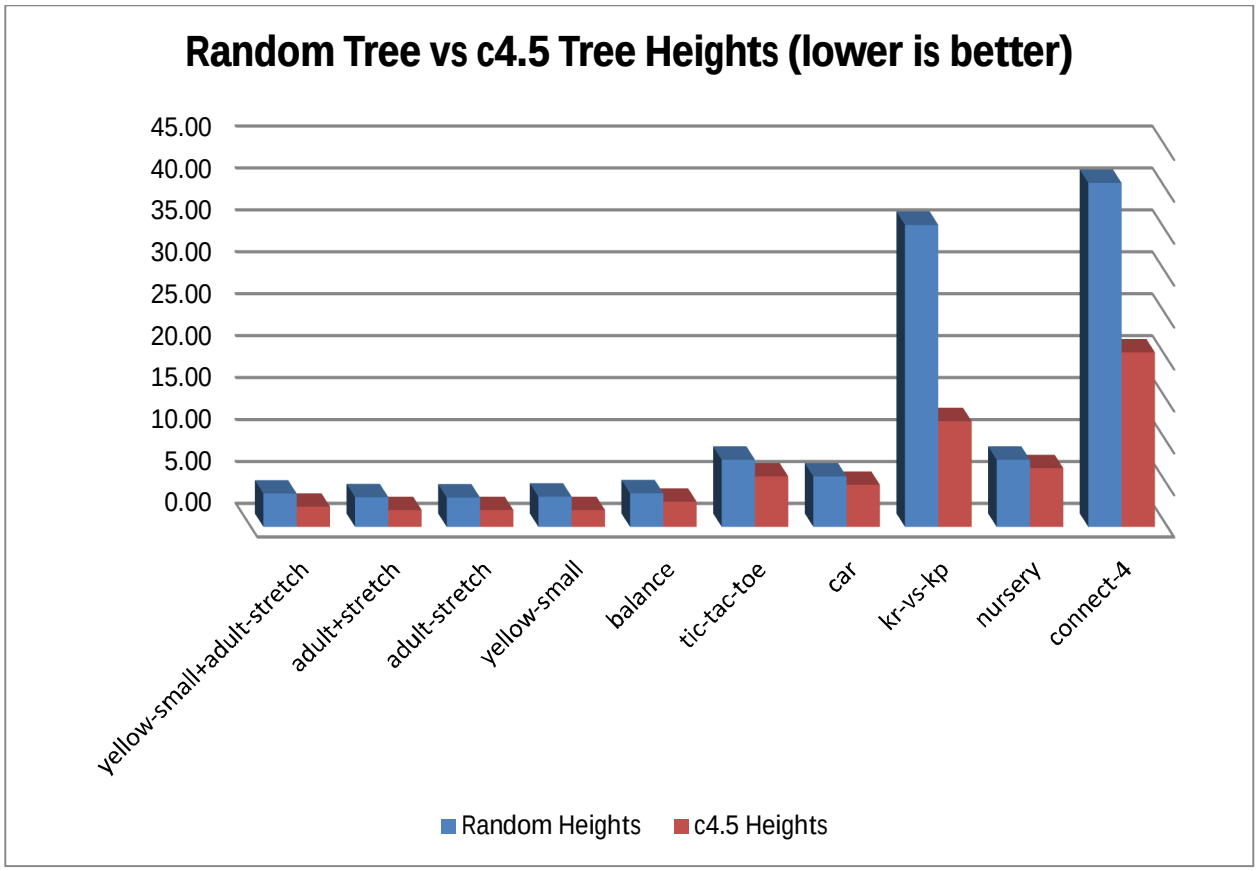
```
Run 3 of Random Tree...
Accuracy: 0.421765350877
Height: 8.0


Total Accuracy Avg (over all 3 runs):  0.425127923977
Total Height  Avg  (over all 3 runs):  8.0

Run 1 of C45 Tree...
Accuracy: 0.836030701754
Height: 6.0




======================================================================
Dataset: yellow-small+adult-stretch
there are 16 data.
each example includes 5 attributes:

color, size, act, age, inflated

The class is: inflated
======================================================================
Run 1 of Random Tree...
Accuracy: 0.5
Height: 4.0


Run 2 of Random Tree...
Accuracy: 0.65
Height: 3.9


Run 3 of Random Tree...
Accuracy: 0.45
Height: 4.0


Total Accuracy Avg (over all 3 runs):  0.533333333333
Total Height  Avg  (over all 3 runs):  3.96666666667

Run 1 of C45 Tree...
Accuracy: 0.65
Height: 2.4




======================================================================
Dataset: yellow-small
```

```
there are 20 data.
each example includes 5 attributes:

color, size, act, age, inflated

The class is: inflated
======================================================================
Run 1 of Random Tree...
Accuracy: 0.9
Height: 3.8


Run 2 of Random Tree...
Accuracy: 0.9
Height: 3.4


Run 3 of Random Tree...
Accuracy: 0.8
Height: 3.6


Total Accuracy Avg (over all 3 runs):  0.866666666667
Total Height  Avg  (over all 3 runs):  3.6

Run 1 of C45 Tree...
Accuracy: 1.0
Height: 2.0
```

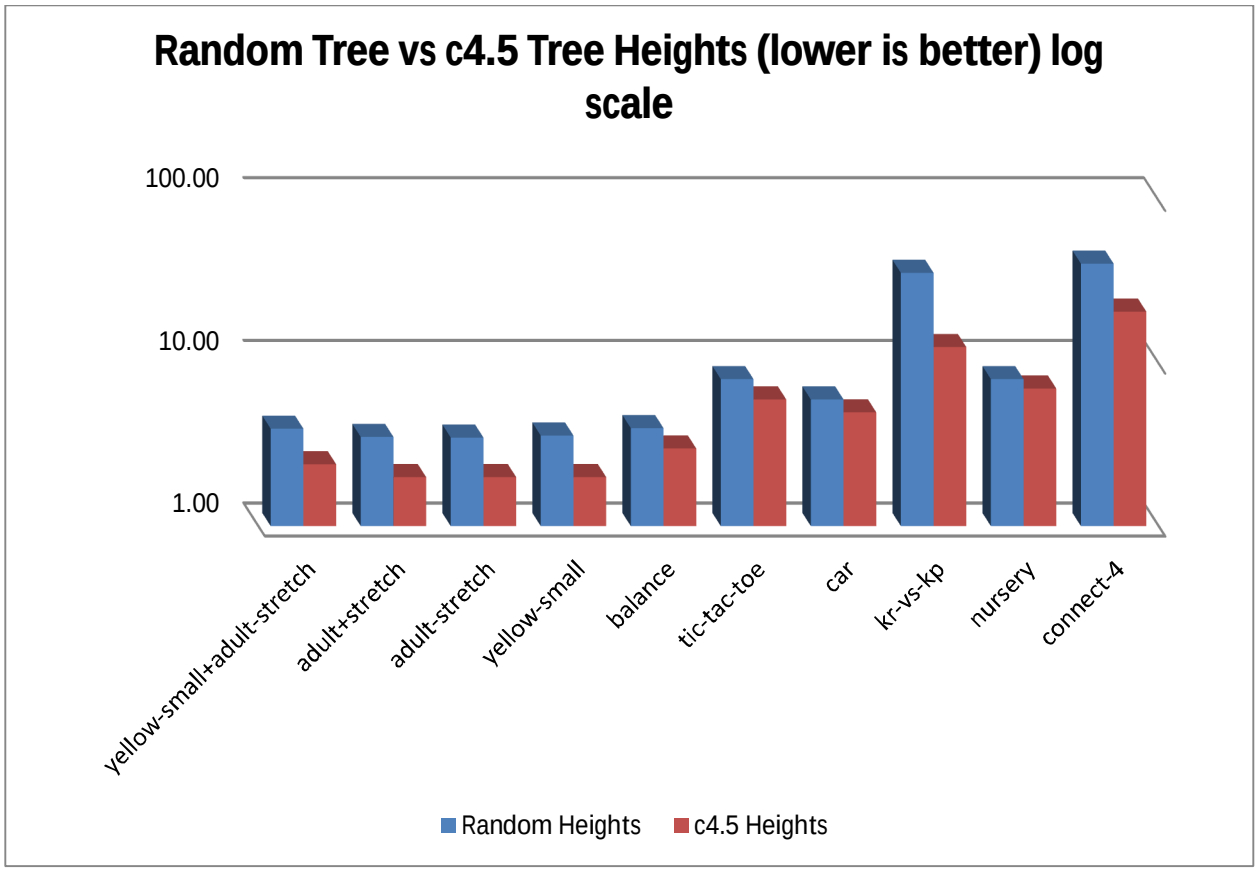
Analysis of Results

A dump of the raw program run is interesting and all, but how does the data correlate to our hypothesis?

Recall our hypothesis: **a shorter tree classifies more accurately**. Therefore it would be useful to examine the heights of the trees that were generated by the two routines.

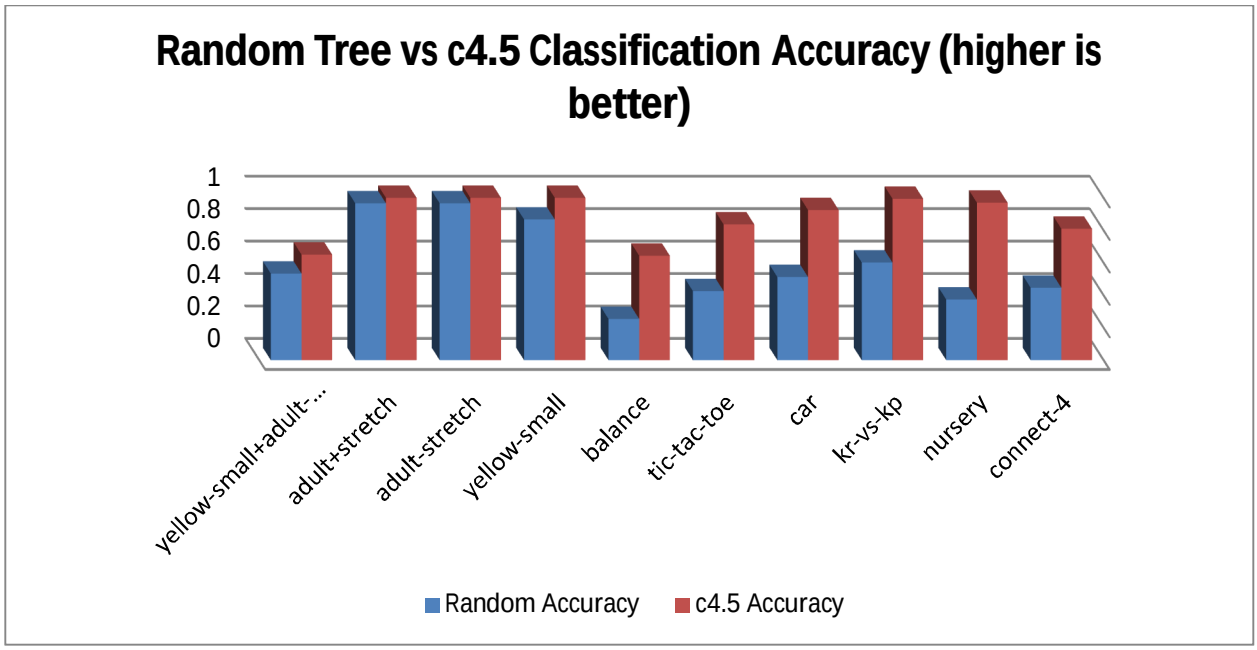


Looking at this graph it should be obvious that the heights of the "random" tree were higher on most of the data sets. On some of the data sets the data is rather close together though, because the graph is getting stretched out by those two values that are higher. It helps to look at a logarithmic graph to see the disparity more clearly, just keep in mind the scale of the graph:



Now it should be clearer that the random trees had a larger height than the c4.5 trees. Therefore, if we can prove that the random trees are **also** less accurate, on the same data sets, then we have proven our hypothesis!

This can easily be done with another graph:



As you can see, even though random trees came very close to matching c4.5 trees in some cases, for the most part c4.5 trees were quite a bit more accurate. The cases where random trees were close were the cases with the smallest number of data points. The first 4 items in that table only had around 20 data points each, meaning after stratification the testing set was only 2 items long. But even in this best-case situation for random trees, where their stupidity is less obvious, c4.5 trees were both more accurate (slightly) and shorter (slightly). For the more significant cases, like kr-vs-kp or connect-4, random trees were drastically outperformed by c4.5 trees.

Therefore, because c4.5 generates shorter trees, and c4.5 is more accurate, our hypothesis that **shorter trees are more accurate** and that Occam's Razor applies are sound. The hypothesis has the potential to be disproven in the future but it is likely that the hypothesis is correct in a general sense.

Conclusions

Occam's Razor seems to hold water… for now. We will see if it continues to in the future! It is good to see that the c4.5 really does work as well as it's claimed to. However one thing I did notice was that my random trees were not much slower than the c4.5 trees, which is unusual since the c4.5 trees should be implemented in C++, so they should be a great deal faster. I did not bother profiling / timing my code to see why this may be the case, I'm guessing that on larger data sets it is more obvious how fast c4.5 is, and perhaps Orange's method of exposing the methods and classes to Python slows it down more than the algorithm itself does.

Final Remarks

This project took WAY longer than I expected. I apologize for submitting it so late; I hope that you will accept it. I have put a great deal of effort into it and I hope that that shows, and I will try to give myself sufficient time to finish future projects by the deadlines. I did not contact you for an extension since we discussed in class that no one had finished it and when I asked if we could submit late you said that you left the submission directory open. I do not wish to abuse this, and I will work very hard to ensure that this is the only time I will have to submit something this late.

Regarding the project, I enjoyed it sometimes and completely hated it others. It was extremely difficult getting the trees to work correctly and I'm just glad that I finished it at all. I am glad that it turned out to validate the claims and that the data seems to be correct. I did my best to ensure that my random tree was implemented correctly, and it appears that it is, and I investigated its inner workings quite thoroughly while debugging it. I really hope future projects do not take this much time / effort because for just an assignment it was quite grueling. But that may just be because I do not understand Orange very well and had to fight with it quite a bit along the way.