

Titanic: Getting Started With R - Part 3: Decision Trees

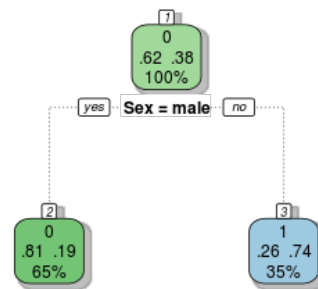
10.01.2014 19 Comments Share

Tutorial index

Last lesson, we sliced and diced the data to try and find subsets of the passengers that were more, or less, likely to survive the disaster. We climbed up the leaderboard a great deal, but it took a lot of effort to get there. To find more fine-grained subsets with predictive ability would require a lot of time to adjust our bin sizes and look at the interaction of many different variables. Luckily there is a simple and elegant algorithm that can do this work for us. Today we're going to use machine learning to build decision trees to do the heavy lifting for us.

Decision trees have a number of advantages. They are what's known as a glass-box model, after the model has found the patterns in the data you can see exactly what decisions will be made for unseen data that you want to predict. They are also intuitive and can be read by people with little experience in machine learning after a brief explanation. Finally, they are the basis for some of the most powerful and popular machine learning algorithms.

I won't get into the mathematics here, but conceptually, the algorithm starts with all of the data at the root node (drawn at the top) and scans all of the variables for the best one to split on. The way it measures this is to make the split on the variable that results in the most pure nodes below it, ie with either the most 1's or the most 0's in the resulting buckets. But let's look at something more familiar to get the idea. Here we draw a decision tree for only the gender variable, and some familiar numbers jump out:



Let's decode the numbers shown on this new representation of our original manual gender-based model. The root node, at the top, shows our **tutorial one** insights, 62% of passengers die, while 38% survive. The number above these proportions indicates the way that the node is voting (recall we decided at this top level that everyone would die, or be coded as zero) and the number below indicates the proportion of the population that resides in this node, or bucket (here at the top level it is everyone, 100%).

So far, so good. Now let's travel down the tree branches to the next nodes down the tree. If the passenger was a male, indicated by the boolean choice below the node, you move left, and if female, right. The survival proportions exactly match those we found in **tutorial two** through our proportion tables. If the passenger was male, only 19% survive, so the bucket votes that everyone here (65% of passengers) perish, while the female bucket votes in the opposite manner, most of them survive as we saw before. In fact, the above decision tree is an exact representation of our gender model from last lesson.

The final nodes at the bottom of the decision tree are known as terminal nodes, or sometimes as leaf nodes. After all the boolean choices have been made for a given passenger, they will end up in one of the leaf nodes, and the majority vote of all passengers in that bucket determine how we will predict for new passengers with unknown fates.

But you can keep going, and this is what I alluded to at the end of the last lesson. We can grow this tree until every passenger is classified and all the nodes are marked with either 0% or 100% chance of survival... All that chopping and comparing of subsets is taken care of for us in the blink of an eye!

Decision trees do have some drawbacks though, they are greedy. They make the decision on the current node which appear to be the best at the time, but are unable to change their minds as they grow new nodes. Perhaps a better, more pure, tree would have been grown if the gender split occurred later? It is really hard to tell, there are a huge number of decisions that could be made, and exploring every possible version of a tree is extremely computationally expensive. This is why the greedy algorithm is used.

As an example, imagine a cashier in a make-believe world with a currency including 25c, 15c and 1c coins. The cashier must make change for 30c using the smallest number of coins possible. A greedy algorithm would start with the coin that leaves the smallest amount of change left to pay:

- Greedy: $25 + 1 + 1 + 1 + 1 + 1 = 30c$, with 6 coins
- Optimal: $15 + 15 = 30c$, with 2 coins

Clearly the greedy cashier algorithm failed to find the best solution here, and the same is true with decision trees. Though they usually do a great job given their speed and the other advantages we already mentioned, the optimal solution is not guaranteed. Decision trees are also prone to overfitting which requires us to use caution with how deep we grow them as we'll see later.

So, let's get started with our first real algo! Now we start to open up the power of R: its packages. R is extremely extensible, you'd be hard pressed to find a package that doesn't automatically do what you need. There's thousands of options out there written by people who needed the functionality and published their work. You can easily add these packages within R with just a couple of commands.

The one we'll need for this lesson comes with R. It's called `rpart` for 'Recursive Partitioning and Regression Trees' and uses the CART decision tree algorithm. While `rpart` comes with base R, you still need to import the functionality each time you want to use it. Go ahead:

```
library(rpart)
```

Now let's build our first model. Let's take a quick review of the possible variables we could look at. Last time we used aggregate and proportion tables to compare gender, age, class and fare. But we never did investigate `SibSp`, `Parch` or `Embarked`. The remaining variables of passenger name, ticket number and cabin number are all unique identifiers for now; they don't give any new subsets that would be interesting for a decision tree. So let's build a tree off everything else.

The format of the `rpart` command works similarly to the `aggregate` function we used in tutorial 2. You feed it the equation, headed up by the variable of interest and followed by the variables used for prediction. You then point it at the data, and for now, follow with the type of prediction you want to run (see `?rpart` for more info). If you wanted to predict a continuous variable, such as age, you may use `method="anova"`. This would run generate decimal quantities for you. But here, we just want a one or a zero, so `method="class"` is appropriate:

```
fit <- rpart(Survived ~ Pclass + Sex + Age + SibSp + Parch + Fare + Embarked, data=train, meth
```

Let's examine the tree. There are a lot of ways to do this, and the built-in version requires running

```
plot(fit)
text(fit)
```

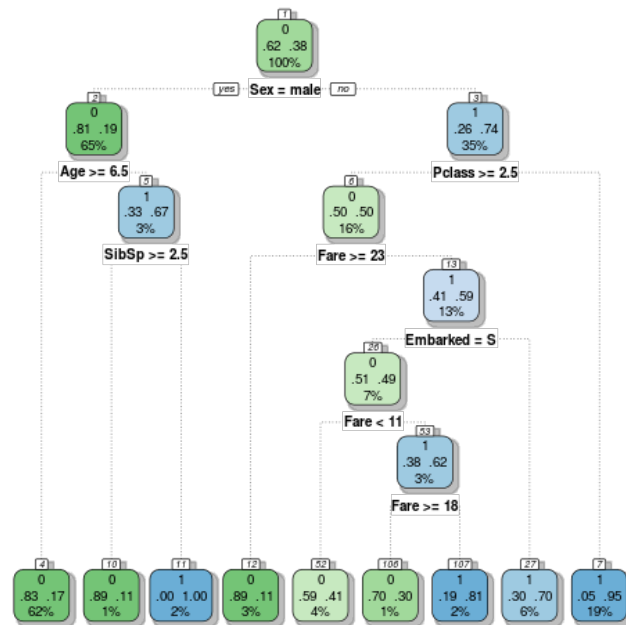
Hmm, not very pretty or insightful. To get some more informative graphics, you will need to install some external packages. As I mentioned, tons of world-class developers donate their time and energy to the R project by contributing powerful packages to CRAN, free of charge. You can install them from within R using `install.packages()`, and load them as before with `library()`. Here are the ones we need for some better graphics for `rpart`:

```
install.packages('rattle')
install.packages('rpart.plot')
install.packages('RColorBrewer')
library(rattle)
library(rpart.plot)
```

```
library(RColorBrewer)
```

Let's try rendering this tree a bit nicer with fancyRpartPlot (of course).

```
fancyRpartPlot(fit)
```



Okay, now we've got somewhere readable. The decisions that have been found go a lot deeper than what we saw last time when we looked for them manually. Decisions have been found for the SibSp variable, as well as the port of embarkation one that we didn't even look at. And on the male side, the kids younger than 6 years old have a better chance of survival, even if there weren't too many aboard. That resonates with the famous naval law we mentioned earlier. It all looks very promising, so let's send another submission into Kaggle!

To make a prediction from this tree doesn't require all the subsetting and overwriting we did last lesson, it's actually a lot easier.

```
Prediction <- predict(fit, test, type = "class")
submit <- data.frame(PassengerId = test$PassengerId, Survived = Prediction)
write.csv(submit, file = "myfirstdtree.csv", row.names = FALSE)
```

Here we have called rpart's predict function. Here we point the function to the model's fit object, which contains all of the decisions we see above, and tell it to work its magic on the test dataframe. No need to tell it which variables we originally used in the model-building phase, it automatically looks for them and will certainly let you know if something is wrong. Finally we tell it to again use the class method (for ones and zeros output) and as before write the output to a dataframe and submission file.

Let's send it in and see how our algorithm performed!

440

new

Total Recall

0.78469

4

Your Best Entry

You improved on your best score by 0.00478.

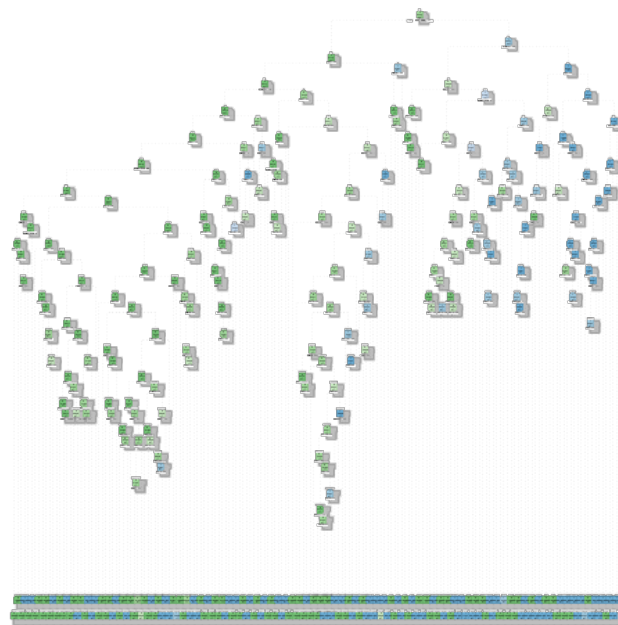
You just moved up 164 positions on the leaderboard.

Nice! We just jumped hundreds of spots with only an extra 0.5% increase in accuracy! Are you getting the picture here? The higher you climb in a Kaggle leaderboard, the more important these fractional percentage bumps become.

The rpart package automatically caps the depth that the tree grows by using a metric called complexity which stops the resulting model from getting too out of hand. But we already saw that a more complex model than what we made ourselves did a bit better, so why not go all out and override the defaults? Let's do it.

You can find the default limits by typing ?rpart.control. The first one we want to unleash is the cp parameter, this is the metric that stops splits that aren't deemed important enough. The other one we want to open up is minsplit which governs how many passengers must sit in a bucket before even looking for a split. Let's max both out and reduce cp to zero and minsplit to 2 (no split would obviously be possible for a single passenger in a bucket):

```
fit <- rpart(Survived ~ Pclass + Sex + Age + SibSp + Parch + Fare + Embarked, data=train,
             method="class", control=rpart.control(minsplit=2, cp=0))
fancyRpartPlot(fit)
```



Okay, I can't even see what's going on here, but with that much subsetting and mining for tiny nuggets of truth, how could we go wrong? Let's make a sub from this model and get to the top of the leaderboard!

440	new	Total Recall	0.78469	5
Your submission scored 0.74163 , which is not an improvement of your best score. Keep trying!				

Even our simple gender model did better! What went wrong? Welcome to overfitting.

Overfitting is technically defined as a model that performs better on a training set than another simpler model, but does worse on unseen data, as we saw here. We went too far and grew our decision tree out to encompass massively complex rules that may not generalize to unknown passengers. Perhaps that 34 year old female in third class who paid \$20.17 for a ticket from Southampton with a sister and mother aboard may have been a bit of a rare case after all.

The point of this exercise was that you must use caution with decision trees. While this particular tree may have been 100% accurate on the data that you trained it on, even a trivial tree with only one rule could beat it on unseen data. You just overfit big time!

Use caution with decision trees, and any other algorithm actually, or you can find yourself making rules from the noise you've mistaken for signal!

Before moving on, I encourage you to have a play with the various control parameters we saw in the rpart.control help file. Perhaps you can find a tree that does a little better by either growing it out further, or reigning it in. You can also manually trim trees in R with these commands:

```
fit <- rpart(Survived ~ Pclass + Sex + Age + SibSp + Parch + Fare + Embarked, data=train,
             method="class", control=rpart.control( your controls ))
new.fit <- prp(fit,snip=TRUE)$obj
fancyRpartPlot(new.fit)
```

An interactive version of the decision tree will appear in the plot tab where you simply click on the nodes that you want to kill. Once you're satisfied with the tree, hit 'quit' and it will be stored to the new.fit object. Try to look for overly complex decisions being made, and kill the nodes that appear to go to far.

Next lesson, we will push the envelope further by introducing some feature engineering concepts. Go there now!

All code from this tutorial is available on my Github repository.

Tweet

0

- ALSO ON TREVOR STEPHENS
- WHAT'S THIS?
- Titanic: Getting Started With R

10 comments
- Titanic: Getting Started With R - Part 1: Booting Up R

13 comments
- Who are these data scientists anyhow?

2 comments
- Titanic: Getting Started With R - Part 5: Random Forests

65 comments

19 Comments

Trevor Stephens

1

Lo...


Sort by Best

Share

Favorite



Join the discussion...



Ravindra

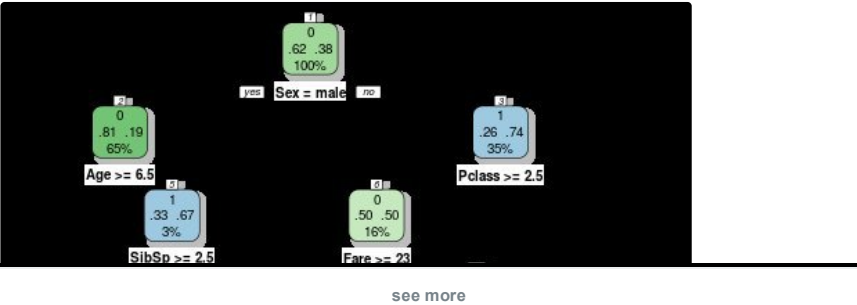
a month ago

Trevor: This tutorial is awesome. I loved the way u laid out the tutorial and even people who have basic sense of statistics can start working with R.

I have a question.

"And on the male side, the kids younger than 6 years old have a better chance of survival, even if there weren't too many aboard"


I am confused with this statement. I read the plot has kids younger than 6 years old have less chances of survival, as decision tree outputs '0' and also the numbers .83 and .17 tells only 17% survived. Please explain.



^ | v

• Reply

• Share



Trevor Stephens

Mod

Ravindra

a month ago

The logic after making the male/female decision asks 'is their age *greater than* 6'. So for the little kids we answer 'no' and move right where 67% survive.

^ | v


• Reply

• Share



Harris

2 months ago



Trevor Stephens

San Francisco, California.

Living, Learning & Loving Analysis:
MS Analytics Graduate from USF.

Looking for the Kaggle tutorial?

Find me on LinkedIn, Twitter and Kaggle.

Archive





Thanks for the awesome tutorial! Just a very basic question, I've got the decision tree displayed on my screen but having trouble with trying to zoom in, or changing the font size to be readable. Any hints on this?

^ | v • Reply • Share ›



Carol Hu • 2 months ago

Great instructions! However, I encountered this error when plotting fit... Error in plot.new() : figure margins too large

^ | v • Reply • Share ›



Trevor Stephens Mod → Carol Hu • 2 months ago

Could just be that your R studio session doesn't have room to plot, try maximizing the window and resizing the plot panel. If that fails try running the command in an R console and it should pop-up a separate plot window. Let me know if that doesn't work...

^ | v • Reply • Share ›



Dan Midwood • 4 months ago

I wanted to be able to record the changes I made when killing the nodes in the interactive tree. And then I found the function ``snip.rpart`` that replicates the remove-node functionality non-interactively, allowing reproducible scripts.

Find the interactive output:

```
`Delete node 33 SibSp >= 2`  
and then put it in your script with:  
`fit <- snip.rpart(fit, toss = 33)`
```

^ | v • Reply • Share ›



Lyon Anderson • 5 months ago

I never would have guess that my mind could be so blown by decision trees. Great tutorial! Looking forward to the next ones.

^ | v • Reply • Share ›



passenger • 5 months ago

Great tutorial... I am a bit confused about the use of control arguments and snipping. Can you point to some resource for a deeper understanding ?

^ | v • Reply • Share ›



Trevor Stephens Mod → passenger • 5 months ago

Do you have any specific questions? The wiki on decision trees is a pretty good place to start: <http://en.wikipedia.org/wiki/D...> . But there's tons of good articles out there on the web.

Documentation on all the control arguments can be found by typing `'?rpart.control'` at the console. But I think the best way to understand these parameters better is to experiment. Change the numbers and then plot the trees to see what they do, if anything.

Snipping/trimming/pruning the trees is just you manually overriding some of these control parameters to make the trees more conservative by removing some of the decisions.

^ | v • Reply • Share ›



passenger → Trevor Stephens • 5 months ago

I was a bit confused about why you chose `cp=0` and `minsplit` to 2. But now I got a hang of it. The parameter `cp` removes any validation on the next split to be effective than the previous and `minsplit` is what you already mentioned. Thanks. One more thing I wanted to ask is that do you have any experience using C5.0 decision tree? I came to know about it from book called 'Machine Learning with R' but I'm facing troubles applying it to this dataset.

^ | v • Reply • Share ›



Trevor Stephens Mod → passenger • 5 months ago

Ah, yes, `cp` is a post trimming metric that automatically reduces the size of your tree. If you want to have more control over stopping the splits at a more arbitrary level you should set it to zero first and then implement your own controls. You can also adjust the `cp` only and leave the others at their defaults, but it is hard to control the growth of the tree that way in my experience.

C5.0 appears to be available in SKLearn (Python) but `rpart` in R uses a variation of CART. You could try out the C50 package, <http://cran.r-project.org/web/...>, but I have not tried it. Max Khun, who co-authored this

package, is pretty legit so I'd expect it to be a very reliable package.

^ | v • Reply • Share ›



passenger → Trevor Stephens • 5 months ago

Actually I was able to use C5.0 decision tree. The trick of combining the train and test data initially, so that both have consistent factor levels, worked (I was able to figure this out after reading your next post. Thanks :)). Infact, it gave a higher score than the default implementation using rpart.

^ | v • Reply • Share ›



Trevor Stephens Mod → passenger • 5 months ago

Awesome, which package?

^ | v • Reply • Share ›



passenger → Trevor Stephens • 5 months ago

C50 package only :) Here is the question I asked on stackoverflow and was able to resolve the problem after reading your next post.

(<http://stackoverflow.com/quest...>

1 ^ | v • Reply • Share ›



Trevor Stephens Mod → passenger • 5 months ago

Thanks for sharing!

^ | v • Reply • Share ›



J. twin • 7 months ago

I'm having an issue when trying to snip the tree. I keep getting this error when I finish:

Error in while ((inode <- identify(node.xy\$x, node.xy\$y, n = 1, plot = FALSE)) && :
missing value where TRUE/FALSE needed

Even if I don't kill a node, this issue occurs. Got any suggestions.

Thanks for the tut by the way. It has been very helpful learning R

^ | v • Reply • Share ›



Trevor Stephens Mod → J. twin • 7 months ago

Have not seen this error before sorry. Anyone else out there encountered this?

^ | v • Reply • Share ›



tkachi • 8 months ago

I had problems in RStudio with install.packages from the console, but fared much better with Install Packages from the Tools menu.

^ | v • Reply • Share ›



Trevor Stephens Mod → tkachi • 8 months ago

I've had issues on some machines in the past with that too, I think it can be a challenge on some machines if you don't have write access to some folders. Good tip for those who see the same thing. Thanks.

^ | v • Reply • Share ›

✉ Subscribe

🔗 Add Disqus to your site

« prev

next »