The Analytics Edge (15.071x)

Monday, April 14, 2014 Knowledge • 1,685 teams Finished Monday, May 5, 2014

0.78 Private AUC: The Best Things I Threw Away

Ozzy Johnson

Posts **10**
Thanks **20**
Joined **11 Apr '14**
Email User

I'd never used R before this class and don't consider myself any sort of programmer, but I probably spent more time and had more fun trying to code tools during this competition than anything else.

Here's how it went in order of private scores generated…

(I've only included an overview of the functions I created, but if there's interest I don't mind going into more detail or trying to tame the sprawling mess of code and post an example.)

Basic GBM: 0.762XX

happyGBM = gbm(Happy ~., data=train,

n.trees=5000,
shrinkage=0.01,
verbose=FALSE,
cv.folds=5,
n.cores=4,
interaction.depth=1)

The only thing notable here is using a simple function I put together called f.getBestIter() to grab the best CV tree from the resulting model and use it when predicting. The real gem here is GBM itself which makes working with sparse training data a breeze. The n.cores parameter is notable as cross-validation is multi-threaded, but R never detects more than 2 cores for me.

Tuned GBM: 0.766XX

happyGBM = gbm(Happy ~., data=train,
n.trees=5000,
shrinkage=0.0056,
verbose=FALSE,
cv.folds=16,
n.cores=4

interaction.depth=1)

This uses parameters I discovered by writing my own caret-like function to iterate through various parameters and tune down to specific values when the model shows improvement. There's nothing magic about these values, but I had good results with them so I froze them while tuning other parts of the process. A different feature count will result in a wildly different set of optimal parameters.

Using my own function allowed me to tune on whatever parameter I chose, but once I discovered caret I did use it as a sort of sanity check for my own work.

Using *apply can speed this up quite a bit, but for whatever reason I could never get cross-validation to work properly when *applying my modeling functions .

Data Cleanup: 0.779XX

f.dataMutate(df) - A function I created which converts multi-level factors to numeric then iterates through different bin sizes looking for the bin size which produces the largest absolute correlation with the dependent variable.

After finding some success with this in a raw format I started investigating the variables by hand and assigning relative numeric values in line with my own intuition about how each might affect the dependent variable.

For factors with few levels it would have been feasible to automate this, but the problem space becomes enormous very quickly as you add levels making a brute force approach less realistic.

Though, I have some good ideas about how to simplify this a bit as I've started learning about matrices and the use of *apply functions in place of loops.

I expect a little tuning of this function could produce a huge score boost. One thing I'd like to investigate is the affect of re-binning on correlations with variables beyond the dependent. I expect optimal per variable correlation is the not the same as optimal correlation of all multi-level variables a group - reads like a knapsack problem, something I just learned about over in 6.002.

Re-tuning the GBM after this step probably wouldn't be a bad idea either.

Sadly, I didn't include any Data Mutation in my final submissions as I started going in another direction when they it failed to move my public scores about a week ago.

Imputation: 0.78XXX

f.miceMaker(df, m, i, s, cols) - A function I created for generating mids objects with various options. In this case, I use it to impute the most important variable from the model created in the step above. This step consistently puts the total over .78, but the amount varies. I expect an ideal set of parameters would be good for a bump of several thousandths to the score.

I've experimented with mice extensively, but never quite figured out how to tune it effectively despite writing my own functions to generate visit sequences and predictor matrices - perhaps if I actually understood the underlying math.

Again, I didn't use any imputation in my final submission for lack of evidence that it was useful in the public AUC.

Miscellaneous:

- A host of other utility functions like f.columnNegative() and f.makeFMLA() for generating formulas when iterating through test models were a help here. Others like f.makeVarMonotone() or f.miceMatrix() were fun to create, but didn't do much to help my scores.
- Use seeds, otherwise you're at the mercy of randomness. This is essential to consistent, repeatable results.
- Save everything. I learned the hard way how easy it is to get lost between parameter changes and find yourself unable to recreate a given model/score. I ended up creating f.saveWithDigest() to calculate an MD5 hash of an object using the digest package and

used that to generate a unique suffix for each file.

- Understand the structure of R objects. Figuring out how to grab bits of models or performance results directly and use them as variables are essential for automating refinements.

Thanked by **BillSChicago** , **Sushmitha** , **Plaice** , **Vinay K** , **Overfitters Anonymous** , and 8 others

#1 / Posted 14 days ago / Edited 14 days ago

MarkRijckenberg

Posts **12**
Thanks **4**
Joined **4 Apr '14**
Email User

Hi Ozzy,

Thanks a lot for the clear and interesting post above.

What are the CPU and memory requirements to run the following monster, where I added class.stratify.cv = TRUE ? ;-)

happyGBM = gbm(Happy ~., data=train,
n.trees=5000,
shrinkage=0.0056,
verbose=FALSE,
cv.folds=16,
n.cores=4,
interaction.depth=1, class.stratify.cv = TRUE)

I tried it and each time it causes my RStudio session to start using swap space like crazy after 5 to 10 minutes. Then R ends processing without giving a model object. I am running R v3.1.0 and RStudio on a laptop with 4 GB of RAM and a Core2Duo 64-bit CPU. I am running Lubuntu 13.10 64-bit.

It runs OK, if I limit n.trees to 2000, shrinkage to 0.02, n.cores to 1 and cv.folds to 9.

Would be great to get insights into this.

Did you get any better results by adding class.stratify.cv = TRUE to the gbm model?

#2 / Posted 14 days ago

Ozzy Johnson

Posts **10**
Thanks **20**

Joined **11 Apr '14**
Email User

On my MacBook Pro with a 3635QM and 8GB running from battery:

```
>system.time(f.timeGBM())

user system elapsed
36.103 1.019 223.848
```

I'm not sure how to best judge memory use in R, but I can see that the 4-8 R processes which fork out consume 130-160M each.

I had some flakiness with GBM myself, I forget the precise issues, but they generally centered around cross-validation.

I experimented with running some of my processes on AWS, but didn't get enough performance to justify the cost/effort. However, that was when I was running everything single-threaded with lots of for loops. Spinning up an instance to crunch a big model surely makes a lot more sense when mc*apply can be used.

Using...

```
class.stratify.cv = TRUE
```

Gave me just slightly worse results 0.78182 vs 0.78185 private score on the model I used for this post.

On a related note...

I did a lot of experimenting with varying levels of cv.folds and found 16 to be the clear sweet spot for my models, but I couldn't figure out why. Also, var monotone and weights produced

some promising results for improving GBM, but I didn't get around to tuning them.

I was curious and tried naively simplifying the same model to 1000 trees, 0.29 shrinkage and 8 folds. It runs in 1/10 the time while losing less than 0.002 in private score.

Thanked by **MarkRijckenberg** and **David Noon**

#3 / Posted 13 days ago / Edited 13 days ago

Reply	You must be logged in to reply to this topic. **Log in »**

Start Watching « Back to forum