June 14, 2015 (/machine-learning-in-practice/2015/6/12/r-caret-and-parameter-tuning-c50)

R, caret, and Parameter Tuning C5.0 (/machine-learning-in-practice/2015/6/12/r-caret-and-parameter-tuning-c50)

By John Alberg (https://www.linkedin.com/pub/john-alberg/3/9a7/8b0)

Boosted C5.0 classifiers are known to perform well when stacked up against other classifiers (see, for example, this paper (http://jmlr.org/papers/volume15/delgado14a/delgado14a.pdf)).

The c (http://topepo.github.io/caret/index.html)aret library for the R programming language is an exceptional environment for automatic parameter tuning and training of classifiers. However, caret does not allow for out-of-box tuning of C5.0 tree complexity. This post shows how you can customize caret to do just that.

Caret has built in capabilities for tuning the C5.0 meta parameters trials, model, and winnow. The C5.0 documentation (http://cran.r-project.org/web/packages/C50/C50.pdf)describes these parameters in detail. The following code illustrates the ease of tuning and training a C5.0 classifier with a custom tuning grid:

```
library(caret)
library(C50)
library(mlbench)

fitControl <- trainControl(method = "repeatedcv",
  number = 10,
  repeats = 10, returnResamp="all")

# Choose the features and classes
data(PimaIndiansDiabetes2)
x <- PimaIndiansDiabetes2[c("age","glucose","insulin","mass","pedigree","pregnant","pressure","triceps")]
y <- PimaIndiansDiabetes2$diabetes

grid <- expand.grid( .winnow = c(TRUE,FALSE), .trials=c(1,5,10,15,20), .model="tree" )

mdl<- train(x=x,y=y,tuneGrid=grid,trControl=fitControl,method="C5.0",verbose=FALSE)

mdl
```

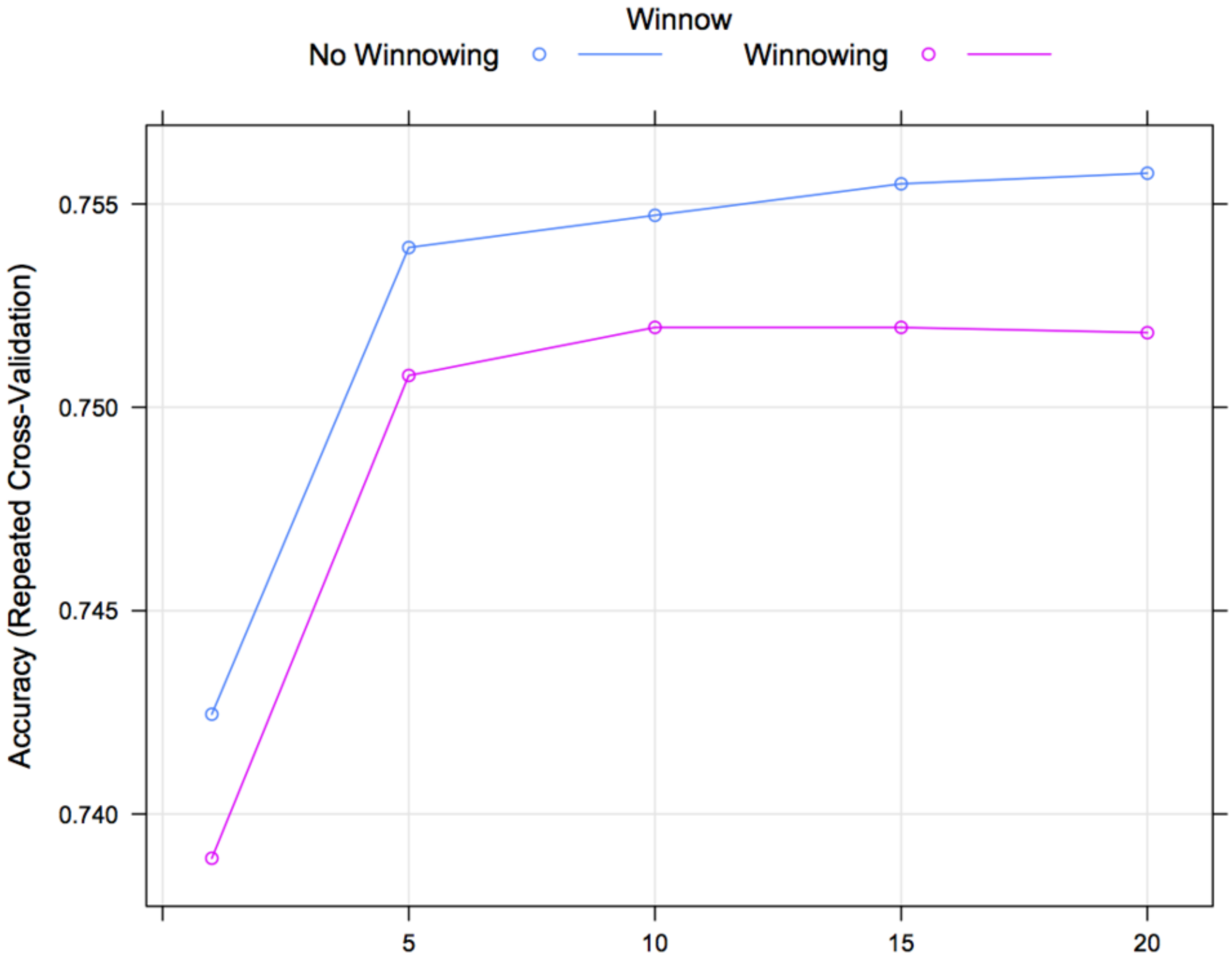
```

# visualize the resample distributions
xyplot(mdl,type = c("g", "p", "smooth"))
```

c5fit.R view raw (https://gist.github.com/euclidjda/387a0384bb419ffdae94/raw/9ca329f6ebd103ffeb9e5c317d403334603fdf00/c5fit.R) (https://gist.github.com/euclidjda/387a0384bb419ffdae94#file-c5fit-r) hosted with ❤ by GitHub (https://github.com)

Then, typing the following

```
> plot(mdl)
```

from the R console will give you this nice chart showing the classifiers performance across the tuning parameters.



Boosting Iterations

Alas, there is an important C5.0 tuning parameter that is not baked into caret. This parameter is minCases. The minCases parameter specifies the minimum number of cases (training examples) that must be put in at least two of the splits. Essentially it controls the depth of the trees created by C5.0 (depth cannot be controlled directly) and hence it is intimately connected with the resulting tree complexity. The purpose of tuning meta parameters is to find the optimal trade-off between model complexity and the training set size and so minCases is an important parameter to tune. That said, tuning minCases is problematic under cross-validation because the number of cases in the training folds are different than the number of cases in the entire dataset so the optimal value of minCases found in cross validation will not be equal to the true optimum for the entire data set (which the final model will be trained on). To overcome this obstacle we can define minCases as a proportion of the data set size and tune the proportional parameter instead. If we define minCases as

```
minCases <- length(y)/splits
```

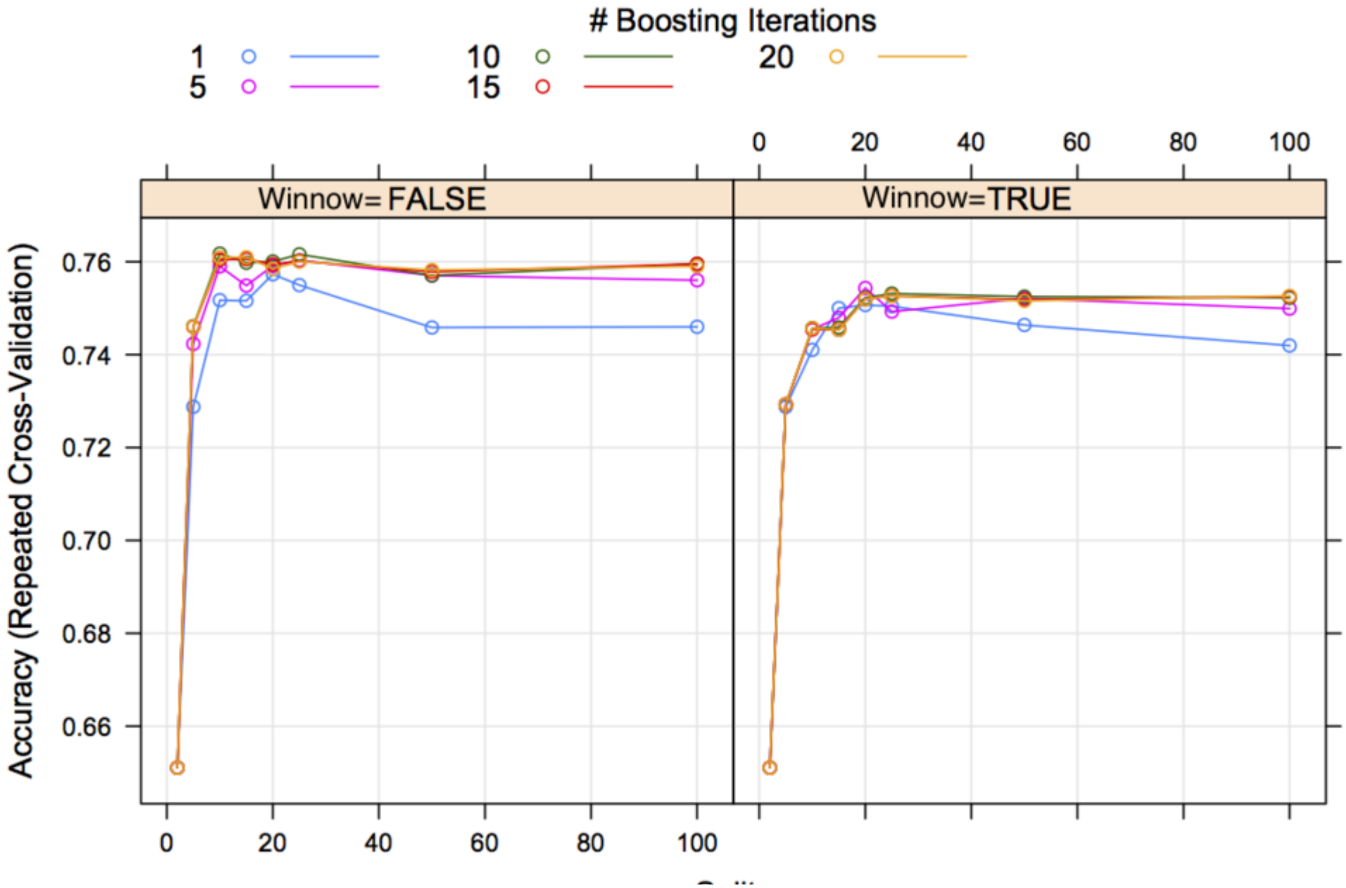
then as "splits" increases, so will the depth and complexity of the resulting trees. The code below customizes the standard caret functions to allow for the tuning of "splits" along with the other C5.0 meta parameters.

```
library(caret)
library(C50)
library(mlbench)

C5CustomSort <- function(x) {

  x$model <- factor(as.character(x$model), levels = c("rules","tree"))
  x[order(x$trials, x$model, x$splits, !x$winnow),]

}

C5CustomLoop <- function (grid)
{
    loop <- ddply(grid, c("model", "winnow","splits"), function(x) c(trials = max(x$trials)))
    submodels <- vector(mode = "list", length = nrow(loop))
    for (i in seq(along = loop$trials)) {
        index <- which(grid$model == loop$model[i] & grid$winnow ==
            loop$winnow[i] & grid$splits == loop$splits[i])
        trials <- grid[index, "trials"]
        submodels[[i]] <- data.frame(trials = trials[trials !=
            loop$trials[i]])
    }
    list(loop = loop, submodels = submodels)
}

C5CustomGrid <- function(x, y, len = NULL) {
  c5seq <- if(len == 1)  1 else  c(1, 10*((2:min(len, 11)) - 1))
  expand.grid(trials = c5seq, splits = c(2,10,20,50), winnow = c(TRUE, FALSE), model = c("tree","rules"))
}

C5CustomFit <- function(x, y, wts, param, lev, last, classProbs, ...) {
  # add the splits parameter to the fit function
  # minCases is a function of splits

  theDots <- list(...)

  splits   <- param$splits
  minCases <- floor( length(y)/splits ) - 1

  if(any(names(theDots) == "control"))
        {
    theDots$control$winnow        <- param$winnow
    theDots$control$minCases      <- minCases
    theDots$control$earlyStopping <- FALSE
  }
  else
  theDots$control <- C5.0Control(winnow = param$winnow, minCases = minCases, earlyStopping=FALSE )

  argList <- list(x = x, y = y, weights = wts, trials = param$trials, rules = param$model == "rules")

  argList <- c(argList, theDots)

  do.call("C5.0.default", argList)

}

GetC5Info <- function() {

  # get the default C5.0 model functions
  c5ModelInfo <- getModelInfo(model = "C5.0", regex = FALSE)[[1]]

  # modify the parameters data frame so that it includes splits
  c5ModelInfo$parameters$parameter <- factor(c5ModelInfo$parameters$parameter,levels=c(levels(c5ModelInfo$parameters$par
  c5ModelInfo$parameters$label <- factor(c5ModelInfo$parameters$label,levels=c(levels(c5ModelInfo$parameters$label),'Spl
  c5ModelInfo$parameters <- rbind(c5ModelInfo$parameters,c('splits','numeric','Splits'))

  # replace the default c5.0 functions with ones that are aware of the splits parameter
  c5ModelInfo$fit  <- C5CustomFit
  c5ModelInfo$loop <- C5CustomLoop
  c5ModelInfo$grid <- C5CustomGrid
  c5ModelInfo$sort <- C5CustomSort

  return (c5ModelInfo)

}

c5info <- GetC5Info()

# Define the structure of cross validation
fitControl <- trainControl(method = "repeatedcv", number = 10,  repeats = 10)
```

```

# create a custom cross validation grid
grid <- expand.grid( .winnow = c(TRUE,FALSE), .trials=c(1,5,10,15,20), .model=c("tree"), .splits=c(2,5,10,15,20,25,50,10

# Choose the features and classes
data(PimaIndiansDiabetes2)
x <- PimaIndiansDiabetes2[c("age","glucose","insulin","mass","pedigree","pregnant","pressure","triceps")]
y <- PimaIndiansDiabetes2$diabetes

# Tune and fit model
mdl<- train(x=x,y=y,tuneGrid=grid,trControl=fitControl,method=c5info,verbose=FALSE)

mdl
```

tuneC5splits.R (https://gist.github.com/euclidjda/fc0f97b653471c55d15d#file-tunec5splits-r) **view raw (https://gist.github.com/euclidjda/fc0f97b653471c55d15d/raw/0f3346f97e317bf1ea424959f3824cce49e6faed/tuneC5splits.R)** hosted with ❤ by **GitHub (https://github.com)**

With this code we can generate cross validated results like those in the following chart:



Splits

Tagged: machinelearning (/machine-learning-in-practice/?tag=machinelearning), rstats (/machine-learning-in-practice/?tag=rstats), R (/machine-learning-in-practice/?tag=R), caret (/machine-learning-in-practice/?tag=caret), classification (/machine-learning-in-practice/?tag=classification)

♥ 0 Likes   < Share

COMMENTS (2)

Newest First   Subscribe via e-mail

Preview   POST COMMENT…

Justfor 3 months ago

Hello,

running the code ,when reaching mdl <- train line, gives 50 warnings as shown below.
Do you have any idea, why and how to fix it?

1: In predict.C5.0(modelFit, newdata, trial = submodels$trials[j]) :
'trials' should be <= 1 for this object. Predictions generated using 1 trials
.. Total of 50 similar lines; sometimes also trials <= 7 or trials <= 9.

John Alberg 3 months ago

C5.0 has an "earlyStopping" parameter that will cause the algorithm to cease adding trials (trees) if the additional trees give poor performance. I updated the gist so that early stopping is set to FALSE. The default is TRUE. Warnings should be gone now. Note, however, that for the purpose of this code the warnings are ok but I changed for cleanliness of execution.

Newer Post
Class distribution plots for machine learning in R and ggplot2 (/machine-learning-in-practice/2015/6/16/class-distribution-plots-for-machine-learning-in-r-and-ggplot2)

Follow our firm: (https://www.linkedin.com/company/euclidean-technologies)

Read our articles in Advisor Perspectives: Dec '13 (http://advisorperspectives.com/newsletters13/What_Matters_More_When_Investing.php) Jan '14 (http://advisorperspectives.com/newsletters14/Why_Are_There_Timeless_Lessons_That_Do_Not_Get_A '14 (http://www.advisorperspectives.com/newsletters14/Misunderstanding_Buffett.php)

THE INFORMATION ON THIS WEB SITE IS NOT AN OFFER TO SELL OR SOLICITATION OF AN OFFER TO BUY AN INTEREST IN ANY INVESTMENT FUND OR FOR THE PROVISION OF ANY INVESTMENT MANAGEMENT OR ADVISORY SERVICES. ANY SUCH OFFER OR SOLICITATION WILL BE MADE ONLY BY MEANS OF DELIVERY OF A CONFIDENTIAL PRIVATE OFFERING MEMORANDUM RELATING TO A PARTICULAR FUND OR INVESTMENT MANAGEMENT CONTRACT TO QUALIFIED INVESTORS IN THOSE JURISDICTIONS WHERE PERMITTED BY LAW. EUCLIDEAN TECHNOLOGIES MANAGEMENT, LLC IS A REGISTERED INVESTMENT ADVISER. MORE INFORMATION ABOUT THE FIRM IS AVAILABLE IN PART 2 OF ITS FORM ADV, WHICH CAN OBTAINED BY CONTACTING THE FIRM AT INFO@EUCLIDEAN.COM (mailto:info@euclidean.com).

PAST PERFORMANCE IS NOT NECESSARILY INDICATIVE OF FUTURE RETURNS. ANY INFORMATION YOU RECEIVE FROM EUCLIDEAN TECHNOLOGIES MANAGEMENT, LLC IS BELIEVED TO BE RELIABLE. NEVERTHELESS, NEITHER EUCLIDEAN TECHNOLOGIES MANAGEMENT, LLC, NOR ITS AGENTS ARE LIABLE FOR ANY DEFICIENCIES IN THE ACCURACY, COMPLETENESS, AVAILABILITY OR TIMELINESS OF SUCH INFORMATION. THE INFORMATION PROVIDED ON THIS WEB SITE DOES NOT NECESSARILY REFLECT THE MOST UP TO DATE OR CURRENT INFORMATION AVAILABLE ON THE PRODUCT OR SERVICE. THE INFORMATION CONTAINED HEREIN IS PROVIDED WITHOUT ANY WARRANTY OF ANY KIND.

USE OF THIS WEBSITE IS GOVERNED BY THE TERMS OF USE. (/terms)