10 Random Hyperparameter Search

The default method for optimizing tuning parameters in `train` is to use a grid search. This approach is usually effective but, in cases when there are many tuning parameters, it can be inefficient. An alternative is to use a combination of grid search and racing. Another is to use a random selection of tuning parameter combinations to cover the parameter space to a lesser extent.

There are a number of models where this can be beneficial in finding reasonable values of the tuning parameters in a relatively short time. However, there are some models where the efficiency in a small search field can cancel out other optimizations. For example, a number of models in caret utilize the "sub-model trick" where *M* tuning parameter combinations are evaluated, potentially far fewer than M model fits are required. This approach is best leveraged when a simple grid search is used. For this reason, it may be inefficient to use random search for the following model codes: `ada`, `AdaBag`, `AdaBoost.M1`, `bagEarth`, `blackboost`, `blasso`, `BstLm`, `bstSm`, `bstTree`, `C5.0`, `C5.0Cost`, `cubist`, `earth`, `enet`, `foba`, `gamboost`, `gbm`, `glmboost`, `glmnet`, `kernelpls`, `lars`, `lars2`, `lasso`, `lda2`, `leapBackward`, `leapForward`, `leapSeq`, `LogitBoost`, `pam`, `partDSA`, `pcr`, `PenalizedLDA`, `pls`, `relaxo`, `rfRules`, `rotationForest`, `rotationForestCp`, `rpart`, `rpart2`, `rpartCost`, `simpls`, `spikeslab`, `superpc`, `widekernelpls`, `xgbDART`, `xgbTree`.

Finally, many of the models wrapped by `train` have a small number of parameters. The average number of parameters is 2.

To use random search, another option is available in `trainControl` called `search`. Possible values of this argument are `"grid"` and `"random"`. The built-in models contained in caret contain code to generate random tuning parameter combinations. The total number of unique combinations is specified by the `tuneLength` option to `train`.

Again, we will use the sonar data from the previous training page to demonstrate the method with a regularized discriminant analysis by looking at a total of 30 tuning parameter combinations:

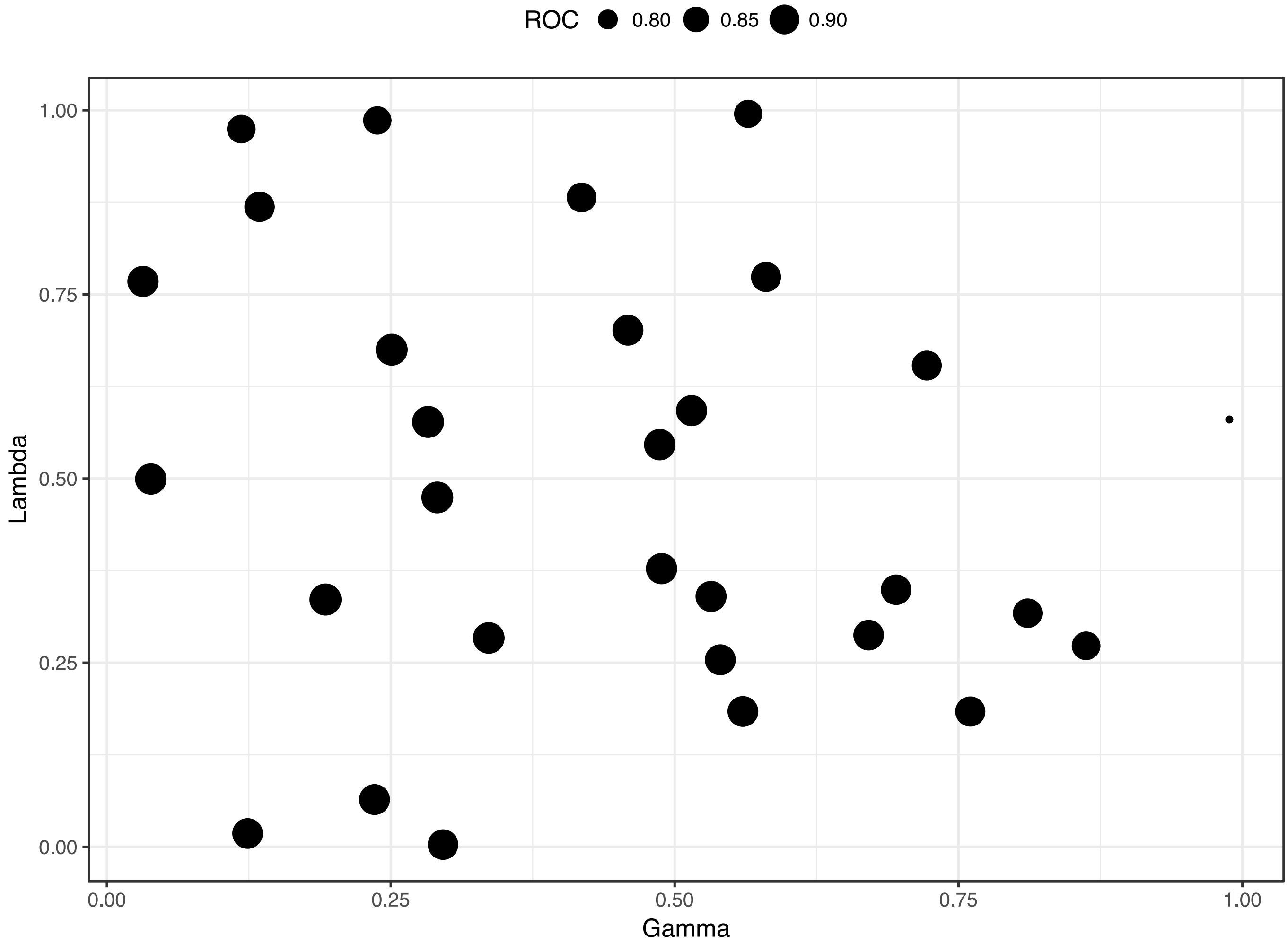
```
library(mlbench)
data(Sonar)

library(caret)
set.seed(998)
inTraining <- createDataPartition(Sonar$Class, p = .75, list = FA
training <- Sonar[ inTraining,]
testing  <- Sonar[-inTraining,]

fitControl <- trainControl(method = "repeatedcv",
                           number = 10,
                           repeats = 10,
                           classProbs = TRUE,
                           summaryFunction = twoClassSummary,
                           search = "random")

set.seed(825)
rda_fit <- train(Class ~ ., data = training,
                 method = "rda",
                 metric = "ROC",
                 tuneLength = 30,
                 trControl = fitControl)
rda_fit
```

```
## Regularized Discriminant Analysis
##
## 157 samples
##  60 predictor
##   2 classes: 'M', 'R'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 10 times)
## Summary of sample sizes: 141, 141, 142, 141, 141, 142, ...
## Resampling results across tuning parameters:
##
##   gamma       lambda       ROC        Sens       Spec
##   0.03177874  0.767664044  0.9168502  0.8998611  0.8182143
##   0.03868192  0.499283304  0.9199752  0.9001389  0.8287500
##   0.11834801  0.974493793  0.8831200  0.8469444  0.7630357
##   0.12391186  0.018063038  0.9090377  0.8851389  0.7975000
##   0.13442487  0.868918547  0.9053943  0.9012500  0.7755357
##   0.19249104  0.335761243  0.9290451  0.9184722  0.8151786
##   0.23568481  0.064135040  0.9126414  0.8923611  0.7782143
##   0.23814584  0.986270274  0.8805159  0.8522222  0.7723214
##   0.25082994  0.674919744  0.9274182  0.9337500  0.7996429
##   0.28285931  0.576888058  0.9275099  0.9225000  0.7969643
##   0.29099029  0.474277013  0.9261954  0.9237500  0.8051786
##   0.29601805  0.002963208  0.9075967  0.8850000  0.7626786
##   0.33633553  0.283586169  0.9232465  0.9187500  0.7855357
##   0.41798776  0.881581948  0.8971677  0.8883333  0.7778571
##   0.45885413  0.701431940  0.9130208  0.9191667  0.7678571
```

```
##    0.48684373  0.545997273  0.9199380  0.9177778  0.7635714
##    0.48845661  0.377704420  0.9178175  0.9105556  0.7633929
##    0.51491517  0.592224877  0.9155010  0.9140278  0.7666071
##    0.53206420  0.339941226  0.9154291  0.9056944  0.7623214
##    0.54020648  0.253930177  0.9131448  0.9043056  0.7626786
##    0.56009903  0.183772303  0.9113790  0.8958333  0.7671429
##    0.56472058  0.995162379  0.8784102  0.8244444  0.8008929
##    0.58045730  0.773613530  0.9015104  0.8868056  0.7694643
##    0.67085142  0.287354882  0.9088269  0.9031944  0.7541071
##    0.69503284  0.348973440  0.9077133  0.9105556  0.7607143
##    0.72206263  0.653406920  0.9003894  0.8908333  0.7676786
##    0.76035804  0.183676074  0.9026513  0.9018056  0.7414286
##    0.81091174  0.317173641  0.8953100  0.9022222  0.7308929
##    0.86234436  0.272931617  0.8841691  0.8976389  0.7196429
##    0.98847635  0.580160726  0.7588616  0.7179167  0.6787500
##
## ROC was used to select the optimal model using the largest val
## The final values used for the model were gamma = 0.192491 and
##  = 0.3357612.
```

There is currently only a `ggplot` method (instead of a basic `plot` method). The results of this function with random searching depends on the number and type of tuning parameters. In this case, it produces a scatter plot of the continuous parameters.

```
ggplot(rda_fit) + theme(legend.position = "top")
```



Ch10
default: train(search="grid")

train(search="random")减少parameter提高部分model的效率，但并非适用于所有model