An Introduction to caret

Max Kuhn

max.kuhn@pfizer.com
Pfizer Global R&D
Nonclinical Statistics
Groton, CT

April 8, 2008

The caret Package

The `caret` package, short for Classification And REgression Training, contains numerous tools for developing predictive models using the rich set of models available in R. The package focuses on

- simplifying model training and tuning across a wide variety of modeling techniques
- pre–processing training data
- calculating variable importance
- model visualizations

The package is available at the Comprehensive R Archive Network (CRAN) at http://cran.r-project.org/. `caret` depends on over 25 other packages, although many of these are listed as "suggested" packages are are not automatically loaded when `caret` is started. Packages are loaded individually when a model is trained or predicted.

An Example

Kazius (2005) investigated using chemical structure to predict mutagenicity (the increase of mutations due to the damage to genetic material).

There were 4,337 compounds included in the data set with a mutagenicity rate of 55.3%. Using these compounds, the `DragonX` software (version 1.2.1) was used to generate a baseline set of 1,579 predictors, including constitutional, topological and connectivity descriptors, among others.

These variables consist of basic numeric variables (such as molecular weight) and counts variables (e.g. number of halogen atoms).

The descriptor data are contained in an `R` data frame names `descr` and the outcome data are in a factor vector called `mutagen` with levels `"mutagen"` and `"nonmutagen"`.

Test/Training Set Split

We decided to keep 75% of the data for training:

```
> library(caret)
> # initial data split
> set.seed(1)
> inTrain <- createDataPartition(mutagen, p = 3/4, list = FALSE)
> # this returns an index of which rows are in the sample
>
> trainDescr <- descr[inTrain,]
> testDescr <- descr[-inTrain,]
>
> trainClass <- mutagen[inTrain]
> testClass <- mutagen[-inTrain]
```

By default, `createDataPartition` does stratified random splits.

Filtering Predictors

There were three zero–variance predictors in the training data. We removed them. We also remove predictors to make sure that there are no between-predictor (absolute) correlations greater than 90%:

```
> ncol(trainDescr)
[1] 1576
> descrCorr <- cor(trainDescr)
> highCorr <- findCorrelation(descrCorr, 0.90)
> # returns an index of column numbers for removal
>
> trainDescr <- trainDescr[, -highCorr]
> testDescr <- testDescr[, -highCorr]
> ncol(trainDescr)
[1] 650
```

Transforming Predictors

The class preProcess can be used to center/scale the predictors, as well as apply other transformations. By default, centering and scaling is done:

```
> xTrans <- preProcess(trainDescr, method = c("center", "scale"))
> trainDescr <- predict(xTrans, trainDescr)
> testDescr <- predict(xTrans, testDescr)
```

To apply PCA to predictors in the training, test or other data, you can use:

```
> xTrans <- preProcess(trainDescr, method = "pca")
```

To apply a " spatial sign transformation" that projects the predictor onto a unit circle (i.e. $x = x/||x||$):

```
> xTrans <- preProcess(trainDescr, method = "spatialSign")
```

Tuning Models using Resampling

Resampling (i.e. the bootstrap, cross–validation) can be used to figure out the values of model tuning parameters (if any).

We come up with a set of candidate values for these parameters and fit a series of models for each tuning parameter combination.

For each combination, fit B models to the B resamples of the training data.

There are also B sets of samples that are not in the resamples. These are predicted for each model.

B sets of performance values is computed for each candidate variable(s).

Performance is estimated by averaging the B performance values.

Tuning Models using Resampling

As an example, a support vector machine with a radial basis function kernel:

$$K(a, b) = \exp(-\sigma ||a - b||^2)$$

has two tuning parameters: σ and the cost value `C`.

We use the method of Caputo et al. (2002) to analytically estimate the value of σ to be $\approx$ 0.0004.

We can train over 5 values of `C`: 10^{-1}, 1, 10, 100 and 1,000.

$B = 25$ iterations of the bootstrap will be used as the resampling method. We use:

```
> svmFit <- train(
+                 x = trainDescr, y = trainClass,
+                 method = "svmradial",
+                 tuneLength = 5,
+                 scaled = FALSE)
```

The train Function

```
> svmFit

3252 samples
650 predictors

summary of bootstrap (25 reps) sample sizes:
    3252, 3252, 3252, 3252, 3252, 3252, ...

boot resampled training results across tuning parameters:

  sigma     C     Accuracy  Kappa  Accuracy SD  Kappa SD  Optimal
  0.000448  0.1   0.707     0.398  0.0102       0.0209
  0.000448  1     0.808     0.612  0.0117       0.0238
  0.000448  10    0.818     0.632  0.00885      0.0179    *
  0.000448  100   0.798     0.59   0.0113       0.0226
  0.000448  1000  0.78      0.555  0.0101       0.0204

Accuracy was used to select the optimal model
```

The Final Model

Resampling indicated that `C = 10` is the best value. It fits a final model with this value and saves it in the object:

```
> svmFit$finalModel
Support Vector Machine object of class "ksvm"

SV type: C-svc  (classification)
 parameter : cost C = 10

Gaussian Radial Basis kernel function.
 Hyperparameter : sigma =  0.000448258519236479

Number of Support Vectors : 1618

Objective Function Value : -9393.825
Training error : 0.080566
Probability model included.
```

Other Tuning Values

If you don't like the default candidate values, you can create your own.

For a boosted tree via gbm:

```
> gbmGrid <- expand.grid(
+                       .interaction.depth = (1:5) * 2,
+                       .n.trees = (1:10)*25,
+                       .shrinkage = .1)
>
> gbmFit <- train(
+                 trainDescr, trainClass,
+                 method = "gbm",
+                 verbose = FALSE,
+                 bag.fraction = 0.5,
+                 tuneGrid = gbmGrid)
Model 1: interaction.depth= 2, shrinkage=0.1, n.trees=250
 collapsing over other values of n.trees
Model 2: interaction.depth= 4, shrinkage=0.1, n.trees=250
 collapsing over other values of n.trees
Model 3: interaction.depth= 6, shrinkage=0.1, n.trees=250
 collapsing over other values of n.trees
Model 4: interaction.depth= 8, shrinkage=0.1, n.trees=250
 collapsing over other values of n.trees
Model 5: interaction.depth=10, shrinkage=0.1, n.trees=250
 collapsing over other values of n.trees
```

Shortcuts

Note that there are 50 different candidate values in `gbmGrid`, but only 5 models were fit.

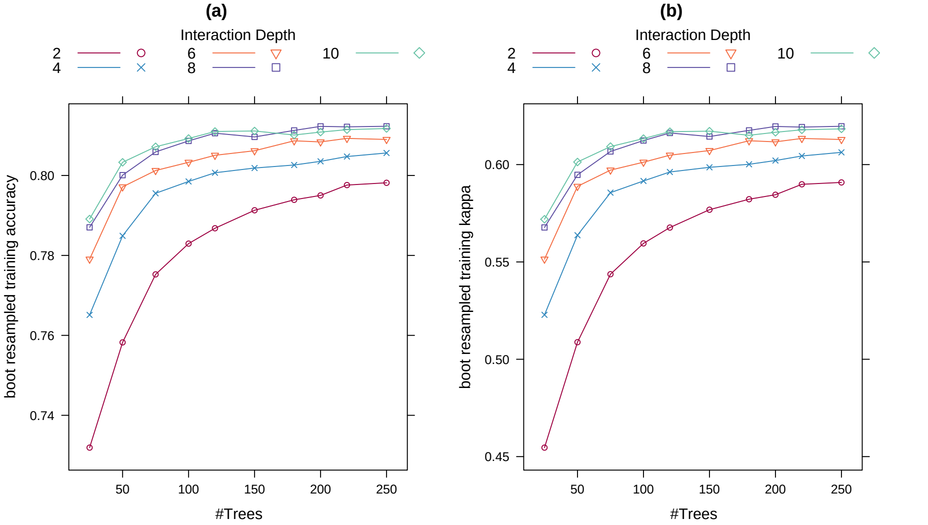
In many cases, `train` will derive model predictions without fitting a model.

In this case, for a specific tree depth, we evaluate 10 different values of `n.trees`.

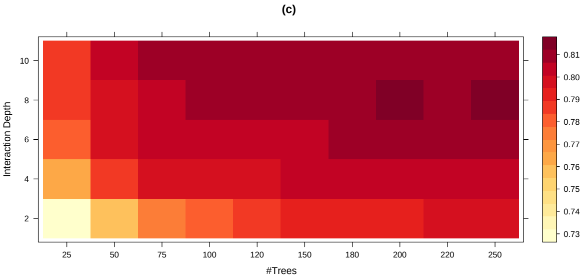
However, if we fit a boosted tree with 250 iterations, we can derive the predictions for all other models with `n.trees < 250` (for the same tree depth).

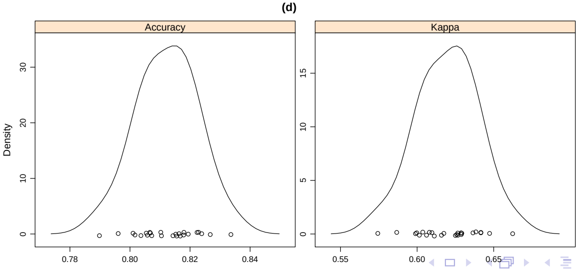
In many models, `train` exploits this to reduce training time.

(a) plot(gbmFit)
(b) plot(gbmFit, metric = "Kappa")



(c) plot(gbmFit, plotType="level")
(d) resampleHist(gbmFit)





Available Models

Model	method Value	Package	Tuning Parameters
Recursive partitioning	rpart	rpart	maxdepth
	ctree	party	mincriterion
Boosted trees	gbm	gbm	interaction.depth, n.trees, shrinkage
	blackboost	gbm	maxdepth, mstop
	ada	ada	maxdepth, iter, nu
Other boosted models	glmboost	mboost	mstop
	gamboost	mboost	mstop
Random forests	rf	randomForest	mtry
	cforest	party	mtry
Bagged trees	treebag	ipred	None
Neural networks	nnet	nnet	decay, size
Partial least squares	pls, plsda	pls, caret	ncomp
Support vector machines (RBF kernel)	svmradial	kernlab	sigma, C
Support vector machines (polynomial kernel)	svmpoly	kernlab	scale, degree, C
Linear least squares	lm	stats	None
Multivariate adaptive regression splines	earth, mars	earth	degree, nprune

Available Models

Model	method Value	Package	Tuning Parameters
Bagged MARS	`bagEarth`	caret, earth	`degree`, `nprune`
Elastic net	`enet`	elasticnet	`lambda`, `fraction`
The lasso	`lasso`	elasticnet	`fraction`
Linear discriminant analysis	`lda`	MASS	None
Logistic/multinomial regression	`multinom`	nnet	`decay`
Regularized discriminant analysis	`rda`	klaR	`lambda`, `gamma`
Flexible discriminant analysis (MARS basis)	`fda`	mda, earth	`degree`, `nprune`
Bagged FDA	`bagFDA`	caret, earth	`degree`, `nprune`
k nearest neighbors	`knn3`	caret	`k`
Nearest shrunken centroids	`pam`	pamr	`threshold`
Naive Bayes	`nb`	klaR	`usekernel`
Generalized partial least squares	`gpls`	gpls	`K.prov`
Learned vector quantization	`lvq`	class	`k`

Predictions

Since the output of `train` contains the final model object, you can use its `predict` methods as usual:

```
> gbmPred <- predict(
+                    gbmFit$finalModel,
+                    newdata = testDescr,
+                    n.trees = 250,
+                    type="link")
> gbmClass <- ifelse(gbmPred >= 0, "mutagen", "nonmutagen")
> gbmProb <-1/(1+exp(-gbmPred))
```

Instead of remembering these nuances, the `caret` functions `extractPrediction` and `extractProb` to handle all of the inconsistent syntax.

It can also handle multiple models at once.

Using extractPrediction to Get Class Predictions

```
> predValues <- extractPrediction(
+                                 list(
+                                      svmFit,
+                                      gbmFit),
+                                 testX = testDescr,
+                                 testY = testClass)
> testValues <- subset(
+                      predValues,
+                      dataType == "Test")
> str(testValues)
'data.frame': 2166 obs. of  4 variables:
 $ obs     : Factor w/ 2 levels "mutagen","nonmutagen": 1 2 1 2 1 1 2 2 2 2 ...
 $ pred    : Factor w/ 2 levels "mutagen","nonmutagen": 1 2 2 2 1 1 2 2 2 2 ...
 $ model   : Factor w/ 2 levels "gbm","svmradial": 2 2 2 2 2 2 2 2 2 2 ...
 $ dataType: Factor w/ 2 levels "Test","Training": 1 1 1 1 1 1 1 1 1 1 ...
> table(testValues$model)
      gbm svmradial
     1083      1083
> nrow(testDescr)
[1] 1083
```

Using extractProb to Get Class Probabilities

```
> probValues <- extractProb(
+                           list(svmFit, gbmFit),
+                           testX = testDescr,
+                           testY = testClass)
>
> testProbs <- subset(
+                     probValues,
+                     dataType == "Test")
> str(testProbs)
'data.frame': 2166 obs. of  6 variables:
 $ mutagen   : num  0.6332 0.2899 0.1662 0.0179 0.9346 ...
 $ nonmutagen: num  0.3668 0.7101 0.8338 0.9821 0.0654 ...
 $ obs       : Factor w/ 2 levels "mutagen","nonmutagen": 1 2 1 2 1 1 2 2 2 2 ...
 $ pred      : Factor w/ 2 levels "mutagen","nonmutagen": 1 2 2 2 1 1 2 2 2 2 ...
 $ model     : chr  "svmradial" "svmradial" "svmradial" "svmradial" ...
 $ dataType  : chr  "Test" "Test" "Test" "Test" ...
```

Evaluating Performance

For classification models, there are functions to compute the confusion matrix and associated statistics. There are also functions for two–class problems: `sensitivity`, `specificity` and so on.

The function `confusionMatrix` calculates statistics for a data set. The no–information rate (NIR) is estimated as the largest class proportion in the data set. A one–sided statistical test is done to see if the observed accuracy is greater than the NIR.

Confusion Matrices and Statistics

```
> svmPred <- subset(testValues, model == "svmradial")
> confusionMatrix(svmPred$pred, svmPred$obs)

Confusion Matrix and Statistics

            Reference
Prediction   mutagen nonmutagen
  mutagen        528         99
  nonmutagen      72        384

            Accuracy : 0.8421
              95% CI : (0.819, 0.8633)
 No Information Rate : 0.554
 P-Value [Acc > NIR] : 8.082e-91

               Kappa : 0.6787

         Sensitivity : 0.88
         Specificity : 0.795
      Pos Pred Value : 0.8421
      Neg Pred Value : 0.8421
```

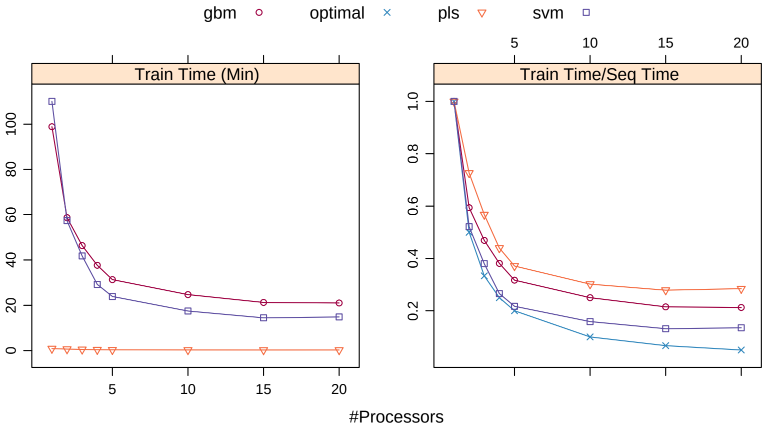
Other Fucntions

caret contains other functions

- an alternate k–nearest neighbor classifier (knn3)
- a function for partial least squares disciminant analysis (plsda)
- maximum dissimilairty sampling (maxDissim)
- a class for variable importance estimates across different models (varImp)
- ROC curves (roc, aucRoc)
- and a few other functions

Parallel Processing

caret has a few sister packages that can be used to parallelize train. One verison, caretNWS uses the NetWorkSpaces framework. The systax is almost identical to train. Benchmarks show a good speedup when compared to sequential processing:



Thanks

Thanks to

- Benevolent Overlords David Potter and Ed Kadyszewski
- Kjell Johnson, Dirk Eddelbuettel, Steve Milborrow, Steve Weston for feedback
- Martin for the invitation