The caret Package: A Unified Interface for Predictive Models

Max Kuhn

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

March 2, 2011

Motivation

Theorem (No Free Lunch)

In the absence of any knowledge about the prediction problem, no model can be said to be uniformly better than any other

Given this, it makes sense to use a variety of different models to find one that best fits the data

R has many packages for predictive modeling (aka machine learning)(aka pattern recognition) ...

Model Function Consistency

Since there are many modeling packages written by different people, there are some inconsistencies in how models are specified and predictions are made.

For example, many models have only one method of specifying the model (e.g. formula method only)

The table below shows the syntax to get probability estimates from several classification models:

`obj` **Class**	**Package**	`predict` **Function Syntax**
`lda`	`MASS`	`predict(obj)` (no options needed)
`glm`	`stats`	`predict(obj, type = "response")`
`gbm`	`gbm`	`predict(obj, type = "response", n.trees)`
`mda`	`mda`	`predict(obj, type = "posterior")`
`rpart`	`rpart`	`predict(obj, type = "prob")`
`Weka`	`RWeka`	`predict(obj, type = "probability")`
`LogitBoost`	`caTools`	`predict(obj, type = "raw", nIter)`

The caret Package

The caret package was developed to:

- create a unified interface for modeling and prediction
- streamline model tuning using resampling
- provide a variety of "helper" functions and classes for day–to–day model building tasks
- increase computational efficiency using parallel processing

First commits within Pfizer: 6/2005

First version on CRAN: 10/2007

Website: http://caret.r-forge.r-project.org

JSS Paper: www.jstatsoft.org/v28/i05/paper

4 package vignettes (82 pages total)

Example Data: TunedIT Music Challenge

http://tunedit.org/challenge/music-retrieval/genres

Using 191 descriptors, classify 12495 musical segments into one of 6 genres: Blues, Classical, Jazz, Metal, Pop, Rock.

Use these data to predict a large test set of music segments.

The predictors and class variables are contained in a data frame called `music`.

Data Splitting

`createDataPartition` conducts stratified random splits

```
> ## Create a test set with 25% of the data
> set.seed(1)
> inTrain <- createDataPartition(music$GENRE, p = .75, list = FALSE)
> str(inTrain)

 int [1:9373, 1] 2 7 14 20 22 47 48 51 64 80 ...
 - attr(*, "dimnames")=List of 2
  ..$ : NULL
  ..$ : chr "Resample1"

> trainDescr <- music[ inTrain, -ncol(music)]
> testDescr  <- music[-inTrain, -ncol(music)]
> trainClass <- music$GENRE[ inTrain]
> testClass  <- music$GENRE[-inTrain]

> prop.table(table(music$GENRE))

     Blues  Classical       Jazz      Metal        Pop       Rock
0.12773109 0.27563025 0.24033613 0.07394958 0.12605042 0.15630252

> prop.table(table(trainClass))

trainClass
     Blues  Classical       Jazz      Metal        Pop       Rock
0.12770724 0.27557879 0.24037128 0.07393577 0.12610690 0.15630001
```

Other functions: `createFolds`, `createMultiFolds`, `createResamples`

Data Pre–Processing Methods

preProcess calculates values that can be used to apply to any data set (e.g. training, set, unknowns).

Current methods: centering, scaling, spatial sign transformation, PCA or ICA "signal extraction", imputation (via bagging or k–nearest neighbors), Box–Cox transformations

```
> ## Determine means and sd's
> procValues <- preProcess(trainDescr, method = c("center", "scale"))
> procValues
```

```
> ## Use the predict methods to do the adjustments
> trainScaled <- predict(procValues, trainDescr)
> testScaled  <- predict(procValues, testDescr)
```

preProcess can also be called within other functions for each resampling iteration.

Model Tuning

train uses resampling to tune and/or evaluate candidate models.

```
> set.seed(1)
> rbfSVM <- train(x = trainDescr, y = trainClass,
+                 method = "svmRadial",
+                 ## center and scale
+                 preProc = c("center", "scale"),
+                 ## Length of default tuning parameter grid
+                 tuneLength = 8,
+                 ## Bootstrap resampling with custon performance metrics:
+                 ## sensitivity, specificity and ROC curve AUC
+                 trControl = trainControl(method = "repeatedcv",
+                                          repeats = 5),
+                 metric = "Kappa",
+                 ## Pass arguments to ksvm
+                 fit = FALSE)
Fitting: sigma=0.005184962, C=0.25
Fitting: sigma=0.005184962, C=0.5
Fitting: sigma=0.005184962, C=1
Fitting: sigma=0.005184962, C=2
Fitting: sigma=0.005184962, C=4
Fitting: sigma=0.005184962, C=8
Fitting: sigma=0.005184962, C=16
Fitting: sigma=0.005184962, C=32
Aggregating results
```

Model Tuning

```
> print(rbfSVM, printCall = FALSE)
9373 samples
 191 predictors

Pre-processing: centered, scaled
Resampling: Cross-Validation (10 fold, repeated 5 times)

Summary of sample sizes: 8437, 8435, 8434, 8435, 8437, 8436, ...

Resampling results across tuning parameters:

  C     Accuracy  Kappa  Accuracy SD  Kappa SD
  0.25  0.916     0.895  0.00953      0.0119
  0.5   0.938     0.923  0.00824      0.0103
  1     0.956     0.945  0.00641      0.008
  2     0.964     0.955  0.00614      0.00766
  4     0.968     0.961  0.0061       0.00761
  8     0.969     0.962  0.00623      0.00777
  16    0.969     0.962  0.00633      0.0079
  32    0.969     0.962  0.0063       0.00786

Tuning parameter 'sigma' was held constant at a value of 0.00518
Kappa was used to select the optimal model using  the largest value.
The final values used for the model were C = 16 and sigma = 0.00518.
```

Model Tuning

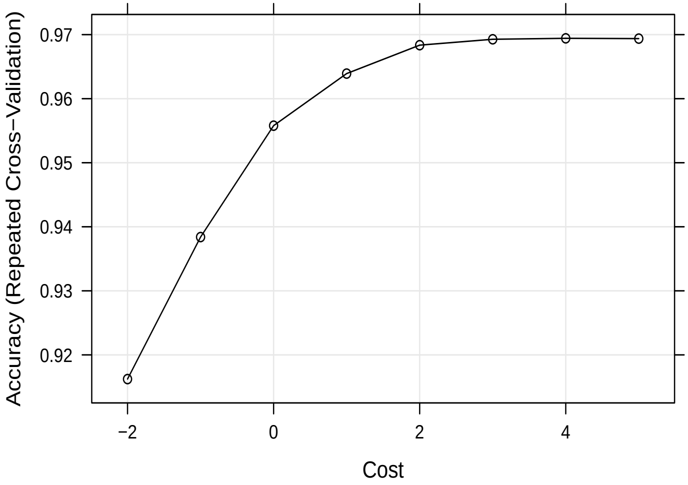
```
> class(rbfSVM)

[1] "train"

> class(rbfSVM$finalModel)

[1] "ksvm"
attr(,"package")
[1] "kernlab"
```

Model Tuning

- `train` uses as many "tricks" as possible to reduce the number of models fits (e.g. using sub–models). Here, it uses the `kernlab` function `sigest` to analytically estimate the RBF scale parameter.
- Currently, there are options for 108 models (see `?train` for a list)
- Allows user–defined search grid, performance metrics and selection rules
- Easily integrates with any parallel processing framework that can emulate `lapply`
- Formula and non–formula interfaces
- Methods: `predict`, `print`, `plot`, `varImp`, `resamples`, `xyplot`, `densityplot`, `histogram`, `stripplot`, . . .

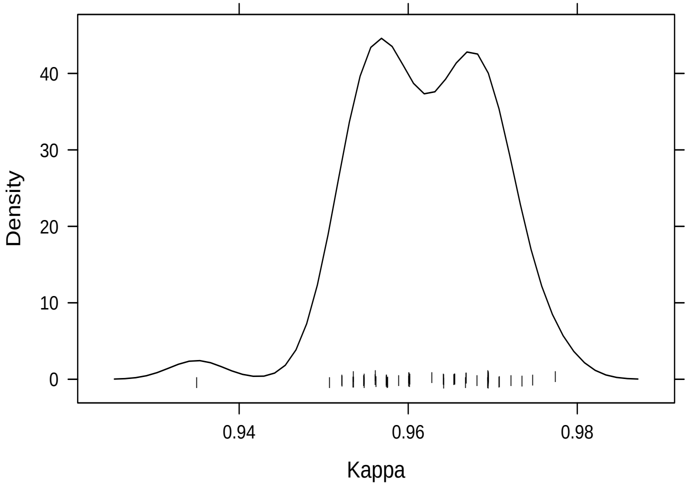
Plots

```
plot(rbfSVM, xTrans = function(x) log2(x))
```



Plots

```
densityplot(rbfSVM, metric = "Kappa", pch = "|")
```



Prediction and Performance Assessment

The `predict` method can be used to get results for other data sets:

```
> svmPred <- predict(rbfSVM, testDescr)
> str(svmPred)

 Factor w/ 6 levels "Blues","Classical",..: 3 2 6 3 5 6 5 1 2 6 ...

> svmProbs <- predict(rbfSVM, testDescr, type = "prob")
> str(svmProbs)

'data.frame': 3122 obs. of  6 variables:
 $ Blues    : num  0.031097 -0.000271 0.056312 0.093638 0.077027 ...
 $ Classical: num  0.5118 0.9907 0.0342 0.1547 0.0908 ...
 $ Jazz     : num  0.3153 0.0105 0.0743 0.3223 0.1635 ...
 $ Metal    : num  0.056453 -0.000301 0.141614 0.143288 0.171406 ...
 $ Pop      : num  0.02457 -0.000187 0.201617 0.143388 0.237104 ...
 $ Rock     : num  0.060765 -0.000447 0.491973 0.142607 0.260142 ...
```

Predction and Performance Assessment

```
> confusionMatrix(svmPred, testClass)

Confusion Matrix and Statistics

           Reference
Prediction  Blues Classical Jazz Metal Pop Rock
  Blues       395         0    0     3   1    1
  Classical     0       841   21     0   1    2
  Jazz          4        20  724     9   4    8
  Metal         0         0    0   214   2    0
  Pop           0         0    0     3 378    6
  Rock          0         0    5     2   7  471

Overall Statistics

               Accuracy : 0.9683
                 95% CI : (0.9615, 0.9742)
    No Information Rate : 0.2758
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9605
 Mcnemar's Test P-Value : NA
```

Predction and Performance Assessment

```
Statistics by Class:

                     Class: Blues Class: Classical Class: Jazz Class: Metal
Sensitivity                0.9900           0.9768      0.9653      0.92641
Specificity                0.9982           0.9894      0.9810      0.99931
Pos Pred Value             0.9875           0.9723      0.9415      0.99074
Neg Pred Value             0.9985           0.9911      0.9890      0.99415
Prevalence                 0.1278           0.2758      0.2402      0.07399
Detection Rate             0.1265           0.2694      0.2319      0.06855
Detection Prevalence       0.1281           0.2771      0.2463      0.06919
                     Class: Pop Class: Rock
Sensitivity              0.9618      0.9652
Specificity              0.9967      0.9947
Pos Pred Value           0.9767      0.9711
Neg Pred Value           0.9945      0.9936
Prevalence               0.1259      0.1563
Detection Rate           0.1211      0.1509
Detection Prevalence     0.1240      0.1553
```

Comparing Models

We can use the resampling results to make formal and informal comparisons between models.

Based on the work of

- Hothorn *et al*. "The design and analysis of benchmark experiments". *Journal of Computational and Graphical Statistics* (2005) vol. 14 (3) pp. 675-699
- Eugster *et al*. "Exploratory and inferential analysis of benchmark experiments". *Ludwigs-Maximilians-Universitat Munchen, Department of Statistics, Tech. Rep* (2008) vol. 30

Comparing Models

```
> set.seed(1)
> rfFit <- train(x = trainDescr, y = trainClass,
+                method = "rf", tuneLength = 5,
+                trControl = trainControl(method = "repeatedcv",
+                                         repeats = 5,
+                                         verboseIter = FALSE),
+                metric = "Kappa")
> set.seed(1)
> plsFit <- train(x = trainDescr, y = trainClass,
+                method = "pls", tuneLength = 20,
+                 preProc = c("center", "scale", "BoxCox"),
+                trControl = trainControl(method = "repeatedcv",
+                                         repeats = 5,
+                                         verboseIter = FALSE),
+                metric = "Kappa")
```

Comparing Models

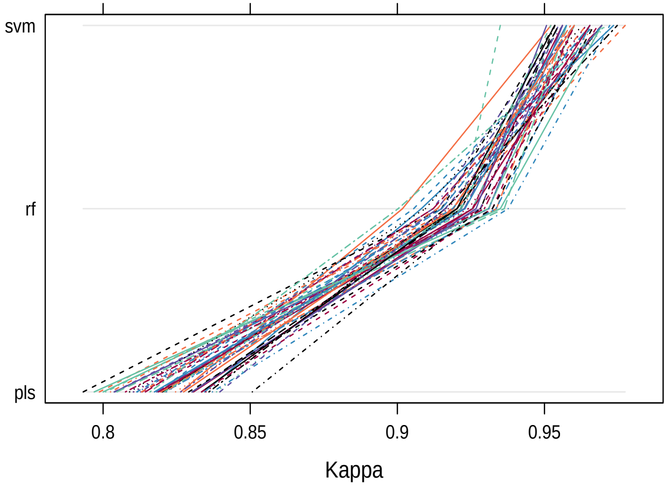
```
> resamps <- resamples(list(rf = rfFit, pls = plsFit, svm = rbfSVM))
> print(summary(resamps))

Call:
summary.resamples(object = resamps)

Models: rf, pls, svm
Number of resamples: 50

Accuracy
      Min. 1st Qu. Median   Mean 3rd Qu.   Max.
rf  0.9200  0.9328 0.9370 0.9370  0.9424 0.9499
pls 0.8348  0.8488 0.8554 0.8554  0.8631 0.8806
svm 0.9478  0.9648 0.9691 0.9694  0.9752 0.9819

Kappa
      Min. 1st Qu. Median   Mean 3rd Qu.   Max.
rf  0.9003  0.9162 0.9215 0.9215  0.9282 0.9376
pls 0.7932  0.8106 0.8192 0.8190  0.8286 0.8507
svm 0.9350  0.9561 0.9615 0.9619  0.9691 0.9774
```

Comparing Models

```
> diffs <- diff(resamps, metric = "Kappa")
> print(summary(diffs))

Call:
summary.diff.resamples(object = diffs)

p-value adjustment: bonferroni
Upper diagonal: estimates of the difference
Lower diagonal: p-value for H0: difference = 0

Kappa
    rf        pls       svm
rf             0.10245  -0.04043
pls < 2.2e-16           -0.14288
svm < 2.2e-16 < 2.2e-16
```

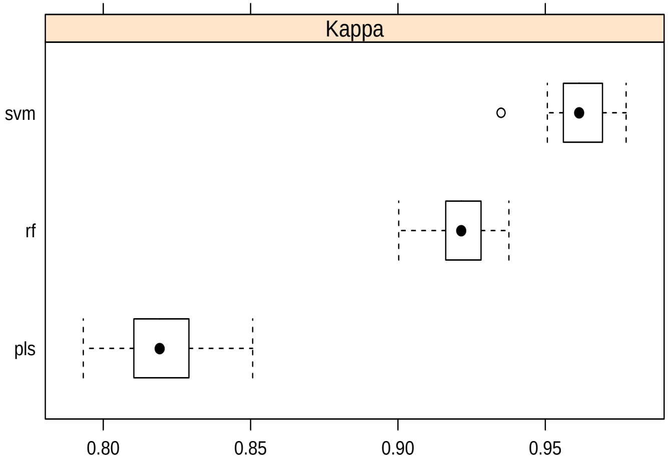
Parallel Coordinate Plots

```
parallel(resamps, metric = "Kappa")
```



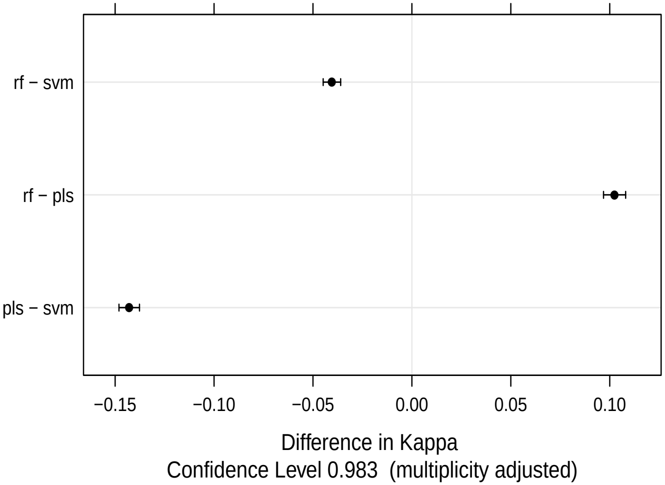
Box Plots

```
bwplot(resamps, metric = "Kappa")
```



Dot Plots of Average Differences

`dotplot(diffs)`



Other Functions and Classes

- `nearZeroVar`: a function to remove predictors that are sparse and highly unbalanced
- `findCorrelation`: a function to remove the optimal set of predictors to achieve low pair–wise correlations
- `predictors`: class for determining which predictors are included in the prediction equations (e.g. `rpart`, `earth`, `lars` models) (currently 57 methods)
- `confusionMatrix`, `sensitivity`, `specificity`, `posPredValue`, `negPredValue`: classes for assessing classifier performance
- `varImp`: classes for assessing the aggregate effect of a predictor on the model equations (currently 19 methods)

Other Functions and Classes

- `knnreg`: nearest–neighbor regression
- `plsda`, `splsda`: PLS discriminant analysis
- `icr`: independent component regression
- `pcaNNet`: `nnet::nnet` with automatic PCA pre–processing step
- `bagEarth`, `bagFDA`: bagging with MARS and FDA models
- `normalize2Reference`: RMA–like processing of Affy arrays using a training set
- `spatialSign`: class for transforming numeric data ($x' = x/||x||$)
- `maxDissim`: a function for maximum dissimilarity sampling
- `rfe`: a class/framework for recursive feature selection (RFE) with external resampling step
- `sbf`: a class/framework for applying univariate filters prior to predictive modeling with external resampling
- `featurePlot`: a wrapper for several `lattice` functions

Thanks

MA RUG Organizers, Kirk Mettler

R Core

Pfizer's Statistics leadership for providing the time and support to create R packages

caret contributors: Jed Wing, Steve Weston, Andre Williams, Chris Keefer and Allan Engelhardt

Session Info

- R version 2.11.1 Patched (2010-09-30 r53356), `x86_64-apple-darwin9.8.0`
- Base packages: base, datasets, graphics, grDevices, methods, splines, stats, tools, utils
- Other packages: caret 4.79, class 7.3-2, codetools 0.2-2, digest 0.4.2, e1071 1.5-24, gbm 1.6-3.1, kernlab 0.9-11, lattice 0.19-11, MASS 7.3-7, pls 2.1-0, plyr 1.2.1, randomForest 4.5-36, reshape 0.8.3, survival 2.35-8, weaver 1.14.0
- Loaded via a namespace (and not attached): grid 2.11.1

This presentation was created with a MacPro using LaTeXand `R`'s `Sweave` function at 09:33 on Saturday, Feb 26, 2011.