Classification and Regression Training in R package *caret*

Cheat Sheet

Made by: Neven Piculjan

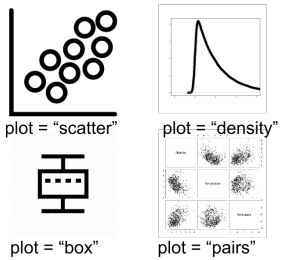
The *caret* package (short for *Classification And REgression Training*) is a set of functions that attempt to streamline the process for creating predictive models. The package contains tools for:

- data splitting
- pre-processing
- feature selection
- model tuning using resampling
- variable importance estimation

as well as other functionality.

Visualizations

`featurePlot (x, y, plot, labels, ...)`
A shortcut to produce lattice graphs



Pre-Processing

Creating Dummy Variables

`dummyVars (formula, data, sep, levelsOnly, fullRank, object, newdata, na.action, n, contrasts, sparse, x, drop2nd, ...)`
dummyVars creates a full set of dummy variables (i.e. less than full rank parameterization)

Zero- and Near Zero-Variance Predictors

`nearZeroVar (x, freqCut, uniqueCut, saveMetrics, names, y, index, foreach, allowParallel)`
nearZeroVar diagnoses predictors that have one unique value (i.e. are zero variance predictors) or predictors that are have both of the following characteristics: they have very few unique values relative to the number of samples and the ratio of the frequency of the most common value to the frequency of the second most common value is large. *checkConditionalX* looks at the distribution of the columns of x conditioned on the levels of y and identifies columns of x that are sparse within groups of y.

Identifying Correlated Predictors

`findCorrelation (x, cutoff, verbose, names, exact)`
This function searches through a correlation matrix and returns a vector of integers corresponding to columns to remove to reduce pair-wise correlations.

Linear Dependencies

`findLinearCombos (x)`
Enumerate and resolve the linear combinations in a numeric matrix.

The preProcess Function

`preprocess (x, method, thresh, pcaComp, na.remove, object, newdata, k, knnSummary, outcome, fudge, numUnique, verbose, ...)`
Pre-processing transformation (centering, scaling etc.) can be estimated from the training data and applied to any data set with the same variables.

Class Distance Calculations

`classDist (x, y, groups, pca, keep, object, newdata, trans, ...)`
This function computes the class centroids and covariance matrix for a training set for determining Mahalanobis distances of samples to each class centroid.

Data Splitting

Simple Splitting Based on the Outcome

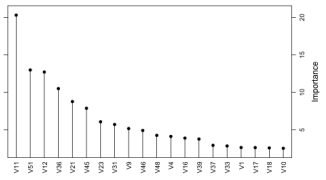
`createDataPartition (y, times, p, list, groups, k, returnTrain, initialWindow, horizon, fixedWindow, skip)`
A series of test/training partitions are created using *createDataPartition* while *createResample* creates one or more bootstrap samples. *createFolds* splits the data into k groups while *createTimeSlices* creates cross-validation sample information to be used with time series data.

Splitting Based on the Predictors

`maxDissim (a, b, n, obj, useNames, randomFrac, verbose, ..., u)`
Functions to create a sub-sample by maximizing the dissimilarity between new samples and the existing subset.

Variable Importance

`varImp (object, useModel, nonpara, scale, ..., numTrees, threshold, data, value, surrogates, competes, estimate, drop, cuts, weights, lambda)`



A generic method for calculating variable importance for objects produced by *train* and method specific methods.

Measuring Model Performance

Evaluating Test Sets

`postResample (pred, obs, data, lev, model, formula, na.rm, x)`
Given two numeric vectors of data, the mean squared error and R-squared are calculated. For two factors, the overall agreement rate and Kappa are determined

`sensitivity (data, reference, positive, negative, prevalence, na.rm, ...)`
These functions calculate the sensitivity, specificity or predictive values of a measurement system compared to a reference results (the truth or a gold standard). The measurement and ”truth” data must have the same two possible outcomes and one of the outcomes must be thought of as a ”positive” results. The sensitivity is defined as the proportion of positive results out of the number of samples which were actually positive. When there are no positive results, sensitivity is not defined and a value of NA is returned. Similarly, when there are no negative results, specificity is not defined and a value of NA is returned. Similar statements are true for predictive values. The positive predictive value is defined as the percent of predicted positives that are actually positive while the negative predictive value is defined as the percent of negative positives that are actually negative.

`confusionMatrix (data, reference, positive, dnn, prevalence, ...)`
Calculates a cross-tabulation of observed and predicted classes with associated statistics.

Formulas:
Suppose a 2x2 table with notation

Predicted \ Reference	Event	No Event
Event	A	B
No Event	C	D

The formulas used here are:

$Sensitivity = \frac{A}{A+C}$

$Specificity = \frac{D}{B+D}$

$Prevalence = \frac{A+C}{A+B+C+D}$

$PPV = \frac{Sensitivity \cdot Prevalence}{Sensitivity \cdot Prevalence + (1-Specificity)(1-Prevalence)}$

$NPV = \frac{Specificity \cdot (1-Prevalence)}{(1-Sensitivity) \cdot Prevalence + (Specificity) \cdot (1-Prevalence)}$

$DetectionRate = \frac{A}{A+B+C+D}$

$DetectionPrevalence = \frac{A+B}{A+B+C+D}$

$BalancedAccuracy = \frac{Sensitivity+Specificity}{2}$

Model Training and Tuning

`train (x, y, form, data, weights, subset, na.action, contrasts, method, ..., preProcess, metric, maximize, trControl, tuneGrid, tuneLength)`
This function sets up a grid of tuning parameters for a number of classification and regression routines, fits each model and calculates a resampling based performance measure.

Pseudocode:

Define sets of model parameter values to evaluate
for *each parameter set* **do**
  for *each resampling iteration* **do**
    Hold-out specific samples
    [Optional] Pre-process the data
    Fit the model on the remainder
    Predict the hold-out samples
  end for
  Calculate the average performance across hold-out predictions
end for
Determine the optimal parameter set
Fit the final model to all the training data using the optimal parameter set

`trainControl (method, number, repeats, verboseIter, returnData, returnResamp, savePredictions, p, search, initialWindow, classProbs, summaryFunction, selectionFunction, preProcOptions, sampling, index, indexOut, timingSamps, predictionBounds, seeds, adaptive, trim, allowParallel)`
Control the computational nuances of the train function. The function generates parameters that further control how models are created, with possible values:

- method: The resampling method: "boot", "cv", "LOOCV", "LGOCV", "repeatedcv", "timeslice", "none" and "oob". The last value, out-of-bag estimates, can only be used by random forest, bagged trees, bagged earth, bagged flexible discriminant analysis, or conditional tree forest models. GBM models are not included (the gbm package maintainer has indicated that it would not be a good idea to choose tuning parameter values based on the model OOB error estimates with boosted trees). Also, for leave-one-out cross-validation, no uncertainty estimates are given for the resampled performance measures.
- number and repeats: number controls with the number of folds in K-fold cross-validation or number of resampling iterations for bootstrapping and leave-group-out cross validation. repeats applied only to repeated K-fold cross-validation. Suppose that method = "repeatedcv", number = 10 and repeats = 3, then three separate 10-fold cross-validations are used as the resampling scheme.
- verboseIter: A logical for printing a training log.
- returnData: A logical for saving the data into a slot called trainingData.
- p: For leave-group out cross-validation: the training percentage
- For method = "timeslice", trainControl has options initialWindow, horizon and fixedWindow that govern how cross-validation can be used for time series data.
- classProbs: a logical value determining whether class probabilities should be computed for held-out samples during resample.
- index and indexOut: optional lists with elements for each resampling iteration. Each list element is the sample rows used for training at that iteration or should be held out. When these values are not specified, train will generate them.
- summaryFunction: a function to compute alternate performance summaries.
- selectionFunction: a function to choose the optimal tuning parameters. and examples. PCAthresh, ICAcomp and k: these are all options to pass to the preProcess function (when used).
- returnResamp: a character string containing one of the following values: "all", "final" or "none". This specifies how much of the resampled performance measures to save.
- allowParallel: a logical that governs whether train should use parallel processing (if available).

Feature Selection

Feature Selection Methods

Wrapper methods evaluate multiple models using procedures that add and/or remove predictors to find the optimal combination that maximizes model performance. In essence, wrapper methods are search algorithms that treat the predictors as the inputs and utilize model performance as the output to be optimized. caret has wrapper methods based on

- recursive feature elimination,
- genetic algorithms, and
- simulated annealing.

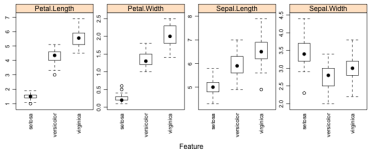
Filter methods evaluate the relevance of the predictors outside of the predictive models and subsequently model only the predictors that pass some criterion. For example, for classification problems, each predictor could be individually evaluated to check if there is a plausible relationship between it and the observed classes. Only predictors with important relationships would then be included in a classification model. Saeys, Inza, and Larranaga (2007) surveys filter methods. *caret* has a general framework for using univariate filters.

Examples

Visualizations

The iris data is used for box plot illustration.

```
featurePlot(x = iris[, 1:4],
                  y = iris$Species,
                  plot = "box",
                  ## Pass in options to bwplot()
                  scales = list(y = list(relation="free"),
                                x = list(rot = 90)),
                  layout = c(4,1 ),
                  auto.key = list(columns = 2))
```



Pre-Processing

In the example below, the half of the MDRR data are used to estimate the location and scale of the predictors. The function preProcess doesn't actually pre-process the data. predict.preProcess is used to pre-process this and other data sets.

```
set.seed(96)
inTrain <- sample(seq(along = mdrrClass), length(mdrrClass)/2)

training <- filteredDescr[inTrain,]
test <- filteredDescr[-inTrain,]
trainMDRR <- mdrrClass[inTrain]
testMDRR <- mdrrClass[-inTrain]

preProcValues <- preProcess(
     training, method = c("center", "scale")
)

trainTransformed <- predict(preProcValues, training)
testTransformed <- predict(preProcValues, test)
```

The preProcess option "ranges" scales the data to the interval [0, 1].

Variable Importance

On the model training web, several models were fit to the example data. The boosted tree model has a built-in variable importance score but neither the support vector machine or the regularized discriminant analysis model do.

```
gbmImp <- varImp(gbmFit3, scale = FALSE)
```

Final project for the course: Advances in Data Mining
Advisor: Przemyslaw Biecek, MINI PW, 2016
http://topepo.github.io/caret/index.html