# 1 The caret Package

- The caret package (short for Classification And REgression Training) was created to streamline the process for building and evaluating predictive models. Using the package, a practitioner can quickly evaluate many different types of models to find the more appropriate tool for their data.

- The beauty of R is that it provides a large and diverse set of modeling packages. However, since these packages are created by many different people over time, there are a minimal set of conventions that are common to each model. For example, Table B.1 shows the syntax for calculating class probabilities for several different types of classification models.

- Remembering the syntactical variations can be difficult and this discourages users from evaluating a variety of models. One method to reduce this complexity is to provide a unified interface to functions for model building and prediction. caret provides such an interface for across a wide vary of models (over 140). The package also provides many options for data pre-processing and resampling-based parameter tuning techniques (Chaps. 3 and 4).

- In this text, resampling is the primary approach for optimizing predictive models with tuning parameters. To do this, many alternate versions of the training set are used to train the model and predict a holdout set. This process is repeated many times to get performance estimates that generalize to new data sets.

- Each of the resampled data sets is independent of the others, so there is no formal requirement that the models must be run sequentially. If a computer with multiple processors or cores is available, the computations could be spread across these workers to increase the computational efficiency. caret leverages one of the parallel processing frameworks in R to do just this.

- The foreach package allows R code to be run either sequentially or in parallel using several different technologies, such as the multicore or Rmpi packages (see *Schmidberger et al.*

*(2009)* for summaries and descriptions of the available options). There are several R packages that work with foreach to implement these techniques, such as **doMC** (for multicore) or doMPI (for Rmpi).

- To tune a predictive model using multiple workers, the syntax in the caret package functions (e.g., train, rfe or sbf) does not change. A separate function is used to register the parallel processing technique and specify the number of workers to use. For example, to use the multicore package (not available on Windows) with five cores on the same machine, the package is loaded and then registered:

```
library(doMC)
registerDoMC(cores = 5)
## All subsequent models are then run in parallel
model <- train(y ~ ., data = training, method = "rf")
```

- The syntax for other packages associated with foreach is very similar. Note that as the number of workers increases, the memory required also increases. For example, using five workers would keep a total of six versions of the data in memory. If the data are large or the computational model is demanding, performance can be affected if the amount of required memory exceeds the physical amount available.
- Does this help reduce the time to fit models? The job scheduling data (Chap. 17) was modeled multiple times with different number of workers for several models. Random forest was used with 2,000 trees and tuned over 10 values of mtry. Variable importance calculations were also conducted during each model fit. Linear discriminant analysis was also run, as was a costsensitive radial basis function support vector machine (tuned over 15 cost values).
- All models were tuned using five repeats of 10-fold cross-validation. The results are shown in Fig.B.1. The y-axis corresponds to the total execution time (encompassing model tuning

and the final model fit) versus the number of workers.

- Random forest clearly took the longest to train and the LDA models were very computationally efficient. The total time (in minutes) decreased as the number of workers increase but stabilized around seven workers. The data for this plot were generated in a randomized fashion so that there should be no bias in the run order.

- The bottom right panel shows the speedup which is the sequential time divided by the parallel time. For example, a speedup of three indicates that the parallel version was three times faster than the sequential version.

- At best, parallelization can achieve linear speedups; that is, for M workers, the parallel time is 1/M. For these models, the speedup is close to linear until four or five workers are used. After this, there is a small improvement in performance. Since LDA is already computationally efficient, the speed-up levels off more rapidly than the other models. While not linear, the decrease in execution time is helpfula nearly 10 h model fit was decreased to about 90 min.

- Note that some models, especially those using the RWeka package, may not be able to be run in parallel due to the underlying code structure.

- One additional trick that train exploits to increase computational efficiency is to use sub-models; a single model fit can produce predictions formultiple tuning parameters. For example, in most implementations of boosted models, a model trained on B boosting iterations can produce predictions for models for iterations less than B. For the grant data, a gbm model was fit that evaluated 200 distinct combinations of the three tuning parameters (see Fig. 14.10).

- In reality, train only created objects for 40 models and derived the other predictions from these objects. More detail on the caret package can be found in Kuhn (2008) or the four extended manuals (called vignettes) on the package web site (Kuhn 2010).