caret Package
Cheat Sheet

Specifying the Model

Possible syntaxes for specifying the variables in the model:

```
train(y ~ x1 + x2, data = dat, ...)
train(x = predictor_df, y = outcome_vector, ...)
train(recipe_object, data = dat, ...)
```

- rfe, sbf, gafs, and safs only have the x/y interface.
- The train formula method will **always** create dummy variables.
- The x/y interface to train will not create dummy variables (but the underlying model function might).

Remember to:

- Have column names in your data.
- Use factors for a classification outcome (not 0/1 or integers).
- Have valid R names for class levels (not "0"/"1")
- Set the random number seed prior to calling train repeatedly to get the same resamples across calls.
- Use the train option na.action = na.pass if you will being imputing missing data. Also, use this option when predicting new data containing missing values.

To pass options to the underlying model function, you can pass them to train via *the ellipses*:

```
train(y ~ ., data = dat, method = "rf",
      # options to `randomForest`:
      importance = TRUE)
```

Parallel Processing

The foreach package is used to run models in parallel. The train code does not change but a "do" package must be called first.

```
# on MacOS or Linux          # on Windows
library(doMC)                 library(doParallel)
registerDoMC(cores=4)         cl <- makeCluster(2)
                              registerDoParallel(cl)
```

The function parallel::detectCores can help too.

Preprocessing

Transformations, filters, and other operations can be applied to the *predictors* with the preProc option.

```
train(, preProc = c("method1", "method2"), ...)
```

Methods include:

- "center", "scale", and "range" to normalize predictors.
- "BoxCox", "YeoJohnson", or "expoTrans" to transform predictors.
- "knnImpute", "bagImpute", or "medianImpute" to impute.
- "corr", "nzv", "zv", and "conditionalX" to filter.
- "pca", "ica", or "spatialSign" to transform groups.

train determines the order of operations; the order that the methods are declared does not matter.

The recipes package has a more extensive list of preprocessing operations.

Adding Options

Many train options can be specified using the trainControl function:

```
train(y ~ ., data = dat, method = "cubist",
      trControl = trainControl(<options>))
```

Resampling Options

trainControl is used to choose a resampling method:

```
trainControl(method = <method>, <options>)
```

Methods and options are:

- "cv" for K-fold cross-validation (number sets the # folds).
- "repeatedcv" for repeated cross-validation (repeats for # repeats).
- "boot" for bootstrap (number sets the iterations).
- "LGOCV" for leave-group-out (number and p are options).
- "LOO" for leave-one-out cross-validation.
- "oob" for out-of-bag resampling (only for some models).
- "timeslice" for time-series data (options are initialWindow, horizon, fixedWindow, and skip).

Performance Metrics

To choose how to summarize a model, the trainControl function is used again.

```
trainControl(summaryFunction = <R function>,
             classProbs = <logical>)
```

Custom R functions can be used but caret includes several: defaultSummary (for accuracy, RMSE, etc), twoClassSummary (for ROC curves), and prSummary (for information retrieval). For the last two functions, the option classProbs must be set to TRUE.

Grid Search

To let train determine the values of the tuning parameter(s), the tuneLength option controls how many values **per tuning** parameter to evaluate.

Alternatively, specific values of the tuning parameters can be declared using the tuneGrid argument:

```
grid <- expand.grid(alpha = c(0.1, 0.5, 0.9),
                    lambda = c(0.001, 0.01))

train(x = x, y = y, method = "glmnet",
      preProc = c("center", "scale"),
      tuneGrid = grid)
```

Random Search

For tuning, train can also generate random tuning parameter combinations over a wide range. tuneLength controls the total number of combinations to evaluate. To use random search:

```
trainControl(search = "random")
```

Subsampling

With a large class imbalance, train can subsample the data to balance the classes them prior to model fitting.

```
trainControl(sampling = "down")
```

Other values are "up", "smote", or "rose". The latter two may require additional package installs.