R-Caret

David Walling

Caret

Caret provides a commone interface for utilizing over 237 ML methods. It provides convenience functions for implementing most of the standard steps in the ML workflow including data imputation, splitting, cross validation and parameter tuning.

To top it all off, caret has excellent documentation:

https://topepo.github.io/caret/index.html

Data

For this excercise, we will be using a dataset from the UCI ML archive for classification of mollusk's as Male, Female or Infant.

http://archive.ics.uci.edu/ml/datasets/Abalone

Load Data

R can read data directly from a URL. There exists and entire family of standard 'read.foo' methods for handling various data types. The 'foreign' package provides additional methods for handling data from programs such as Excel, SAS and Stata.

By default, read.csv will treat all character data as factors, i.e. categorical data. This is generally what you'd want.

This file does not contain header information, but we can use the description on the website to assign column names.

```r
url="http://archive.ics.uci.edu/ml/machine-learning-databases/abalone/abalone.data"
data.all=read.csv(url, header = F, sep = ",")

colnames(data.all) = c('type', 'longestshell', 'diameter', 'height', 'wholeweight', 'shuckedweight', 'vi

summary(data.all)
```

```
##  type      longestshell      diameter          height
##  F:1307   Min.   :0.075   Min.   :0.0550   Min.   :0.0000
##  I:1342   1st Qu.:0.450   1st Qu.:0.3500   1st Qu.:0.1150
##  M:1528   Median :0.545   Median :0.4250   Median :0.1400
##           Mean   :0.524   Mean   :0.4079   Mean   :0.1395
##           3rd Qu.:0.615   3rd Qu.:0.4800   3rd Qu.:0.1650
##           Max.   :0.815   Max.   :0.6500   Max.   :1.1300
##   wholeweight     shuckedweight    visceraweight     shellweight
##  Min.   :0.0020   Min.   :0.0010   Min.   :0.0005   Min.   :0.0015
##  1st Qu.:0.4415   1st Qu.:0.1860   1st Qu.:0.0935   1st Qu.:0.1300
##  Median :0.7995   Median :0.3360   Median :0.1710   Median :0.2340
##  Mean   :0.8287   Mean   :0.3594   Mean   :0.1806   Mean   :0.2388
##  3rd Qu.:1.1530   3rd Qu.:0.5020   3rd Qu.:0.2530   3rd Qu.:0.3290
##  Max.   :2.8255   Max.   :1.4880   Max.   :0.7600   Max.   :1.0050
##      rings
##  Min.   : 1.000
##  1st Qu.: 8.000
##  Median : 9.000
##  Mean   : 9.934
##  3rd Qu.:11.000
##  Max.   :29.000
```

Reproducible Randomness

Most ML methods depend in some way on randomness in how they explore the solution space. With computers, there is no such thing as a truly random event, everything is deterministic by their nature. Instead, you have what is called 'psuedo' random number generators that produce data that appears random, but in fact is determined directly by the 'seed' value used. This is a good thing as it allows you to create solutions that are fully reproducible.

In R, use set.seed to ensure reproducible randomness

```
set.seed(1)
```

Messy Data

This dataset if pretty clean in that it has no missing values. So, we are going to randomly add some missing values in so we can excercise imputation techniques.

For missing continous data, we could either replace with mean/median or impute using carets preProcess function.

For categorical data, we could replace with the mode or build a classifier using the other variables.

However, use caution as the fact that the value is missing could itself indicate something about that sample.

```
data.all$diameter[sample(1:nrow(data.all), size=400, replace=F)] <- NA
```

Imputation

We can now utilize Caret's preProcess function to compute the missing values. Only continuous data can be imputed and only continous data can be used as input. If you have categorical variables, you would first need to one-hot encode those using the dummyVars method in Caret.

Here, all variables are continous. We will use the ensemble method 'bagImpute'.

```
imputer <- preProcess(data.all, method='bagImpute')
data.imputed <- predict(imputer, data.all)

data.all$diameter <- data.imputed$diameter
```

Train vs Test

Caret provides a function for splitting your data into train and test splits. It will implement stratified random sampling to ensure that rare categories are represented evenly between the sets.

In our case, the data is already well balanced as evident by the counts of each class factor level.

```r
trainIdx = createDataPartition(data.all$type, p=0.8, list=F, times=1)

train.data = data.all[trainIdx,]
test.data = data.all[-trainIdx,]
```

Multinom

We will use the multinom package for classification.

Multinom is part of the nnet package and implements a simple neural network for the classification tasks.

Train Control

Caret makes it very simply to implement cross validation in your ML workflow. Cross validation involves building your model repeatedly against a subset of the data and testing it against a hold out set. The primary purpose is to ensure the resulting model is not heavily influced by outliers.

For the purposes of this demo, we will run 3-fold CV. The standard is to use 10-fold.

```r
trctrl <- trainControl(method = "cv", number = 3)
```

Parameter Tuning

Most ML methods have 1 or more tunable parameters and the performance of the particular method can very greatly based on those settings. Thus, one needs to repeat the model fitting process multiple times to test the effects of those parameters on your result.

Caret will automatically run multiple instances of your model with various values of the method parameters in order to find the best combination for you tasks. This can be controlled with the 'tuneLenght' parameter to the fit method.

You can also explictly setup a 'grid' of values, can caret will repeat the model fitting for each combination of the parameters. For our case, we have only 1 parameter and will let caret automaticall select 3 values for testing.

Parallel Processing

Caret can leverage multiple cores or even multiple nodes seemlessly. Both parameter tuning and cross validation are embarrisingly parallel workflows, meaning one iteration does not depend on another.

Caret utilizes the 'foreach' package to train each model seperately, so all one must do is register a 'backend' for this package. In our case, will utilize the 'multicore' package to use all cores on a single node. Note that multicore uses forking and thus only works on linux and mac.

Also be advised that some methods that caret wraps are themselves already multithreaded, for example the xgboost package. When using those methods, one must be careful not to spaw too many processes as this may slow down the overall training as the processes compete for compute cycles. Use 'top' on linux/mac to view the actual processor usage as the training takes place.

```r
library(doMC)
registerDoMC(4)
```

Model Fitting

We are now ready to fit our model. With CV = 3 and tuneLength = 3, a total of 9 models will be fit.

```
fit <- train(type ~.,
             data = train.data,
             method = "multinom",
             trControl=trctrl,
             tuneLength = 3
             )
```

```
## # weights:  30 (18 variable)
## initial  value 3672.660881
## iter  10 value 2912.109381
## final  value 2906.463416
## converged
```

Model Fitting

We can view the selected model as follows:

```
print(fit)
```

```
## Penalized Multinomial Regression
##
## 3343 samples
##    8 predictor
##    3 classes: 'F', 'I', 'M'
##
## No pre-processing
## Resampling: Cross-Validated (3 fold)
## Summary of sample sizes: 2229, 2228, 2229
## Resampling results across tuning parameters:
##
##   decay  Accuracy   Kappa
##   0e+00  0.5510046  0.3235156
##   1e-04  0.5510046  0.3235156
##   1e-01  0.5536965  0.3258745
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was decay = 0.1.
```

Performance

After we have trained our model, it is very easy to then evaluate the performance against our hold out test data. Caret will automatically select relevant metrics for the given ML method used.

```
truth = test.data$type
predictions = predict(fit, newdata=test.data)

confusionMatrix(predictions, truth)
```

```
## Confusion Matrix and Statistics
##
```

```
##           Reference
## Prediction   F   I   M
##          F  84   9  76
##          I  40 223  66
##          M 137  36 163
##
## Overall Statistics
##
##                Accuracy : 0.5635
##                  95% CI : (0.5291, 0.5975)
##     No Information Rate : 0.3657
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.3412
##
##  Mcnemar's Test P-Value : 5.941e-10
##
## Statistics by Class:
##
##                      Class: F Class: I Class: M
## Sensitivity            0.3218   0.8321   0.5344
## Specificity            0.8517   0.8127   0.6730
## Pos Pred Value         0.4970   0.6778   0.4851
## Neg Pred Value         0.7338   0.9109   0.7149
## Prevalence             0.3129   0.3213   0.3657
## Detection Rate         0.1007   0.2674   0.1954
## Detection Prevalence   0.2026   0.3945   0.4029
## Balanced Accuracy      0.5867   0.8224   0.6037
```