Caret package

Dr. Saad

Caret Package (**C**lassification And **RE**gression **T**raining)

There are many algorithms to build **predictive models**. Unfortunately, they are distributed via different R packages, built by different authors, and often use different syntax. **Wouldn't it be nice if we have one package that serves many of these algorithms?** The answer is **caret**.

- Caret aims essentially to provide a uniforn interface for many different functions from different packages.
- Caret 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

[caret website:]http://topepo.github.io/caret/index.html

Caret Package Essential Functions

1. **createDataPartition()**

```r
args(createDataPartition)
```

```
# function (y, times = 1, p = 0.5, list = TRUE, groups = min(5,
#     length(y)))
# NULL
```

y: The outcome

times: default is 1, the number of partitions you want to generate (repeated splitting)

p: the percentage of training data

list: Whether you want the results as a list or not (TRUE or FALSE)

Note

Use a seed number to replicate the results

```r
set.seed(45678)
```

2. **Caret train** Function

- Caret package currently includes **238** different methods, they are summarized in this section (http://topepo.github.io/caret/available-models.html) in the **train** function.
- Caret does not include nor install the needed packages automatically, so to use a package through caret, we neet to install the required package.
- The required packages for each method are described by this list (http://topepo.github.io/caret/train-models-by-tag.html)
- Example of **train** function

```
train(y ~ ., method = "lm", data = train_set)
```

Training Models (continue)

`Train()` function has several arguments:

1. **formula**: the first argument can be a formula just like in `lm()` function

```
formula(y ~ x1 + x2 + ...)
```

2. **Method**: A **string** specifying which classification or regression model to use, such **lm**, **glm**, **knn**, **rf** (random forest), **nnet** (neural network) ...
3. **Data**: the training data set to train the model on.

```
data("Auto", package = 'ISLR')
train(mpg ~ weight + horsepower, method = "lm", data = Auto)
train(mpg ~ weight + horsepower, method = "rpart", data = Auto)
train(mpg ~ weight + horsepower, method = "gam", data = Auto)
train(mpg ~ weight + horsepower, method = "knn", data = Auto)
```

Predictions

3. Predict [**predict.train()**]

Prediction is easy using the output of **train** function. Just feed the trained model to **predict** function. (note, passing a train class object to prediction will automatically invoke predict.train() function). Also, note that we have `predict()` for every class and from different packages such as `predict.lm()`, `predict.glm()`, `predict.knn()` ...

```
args(predict.train)
```

```
function (object, newdata = NULL, type = "raw", na.action = na.omit,
    ...)
NULL
```

- Examples

```
predict(train_glm, newdata = testing, type = 'raw')
predict(train_knn, newdata = testing, type = 'raw')
```

Note

Pay attention to the type you are prediction.
for logistic regression, the type can be **log-odds** (default) or **probabilities** (type = "prob")

Othe Caret Package Functions

- Information about models or algorithms, like **hyperparameters**

```
getModelInfo('knn')
 # For parameters and hyperparameter for tuning
modelLookup('knn')
```

- Bootstrapping and Cross Validation Functions

```
createResample()      # Bootstrapping
createFolds()         # for k-fold cross-validation
createMultiFolds()    # for repeated cross-validation
createTimeSlices()    # CV for time Series
preProcess()          # For Preprocessing
trainControl()        # For hyper-parameter tuning (Extremely Important)
```

Note

We will use these functions throughout our courses.

k-fold Cross Validation

Before defining cross-validation, we should ask this question:

- **What are the drawbacks of splitting data into train and test set?**
  - The presence or absence of an outlier in the **test set** will highly affect the out-of-sample **RMSE**.
  - If we have a small data set, we will end up training the model on few observations which we will end up with a bad model.

A Better Solution for train/test split is Cross-Validation or CV for short

- **Definition of K-fold cross-validation**: it means randomly partition the data into **non-overlapping** (**every single points occurs only once**) **k** folds or sets of roughly equal size.
- If $k = 2$: we have a **2-fold CV** special case called **Validation Set Approach**
- If $k = 5$: then we have a **5-fold CV**
- If $k = 10$: then we have a **10-fold CV**

How CV works

Cross-validation is one of the most commonly approaches used today, but **how the folds are created?**

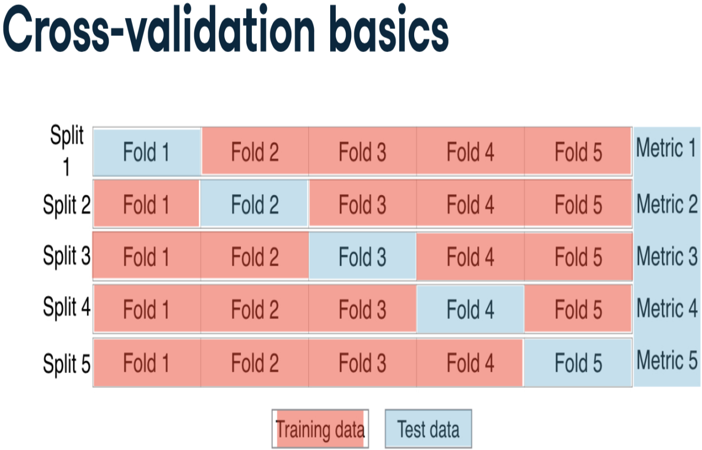
The folds are created in such a way that each point in the dataset occurs in **exactly one test set**. For a **10-fold CV** we will have **10 test sets**. In this way, we get a test set that is the same size as our training set, but is **composed of out-of-sample predictions** which gives us a more precise estimate of true out-of-sample error.

How it works:

suppose we have a 5-fold CV samples (**A, B, C, D, E**)

1. Set **A** as hold-out sample, then train the model on (B, C, D, E), Evaluate the model on **A** (RMSE for example)
2. Set **B** as hold-out sample, then train the model on (A, C, D, E), Evaluate the model on **B**
3. Repeat the process until all samples are used as a test set
4. Take the mean of all RMSEs. (This is the final out-of-sample test error)

Cross-Validation (continue)



Essential Note on Cross Validation

- Train function in caret uses **bootstrap** method by default. (It can do CV as well)
- Cross-validation is only used to estimate the out-of-sample error for our model.
- Re-fit model on the full training dataset, so as to fully exploit the information in that dataset.
- For example: Doing a 10-fold CV means fitting **11 models** (10 cross-validation models plus the final model). It takes 11 times longer than fitting one model.
- Cross validation is essential but the more folds we use, the more computationally expensive cross-validation becomes.

Boostrap Definition:

A bootstrap sample is a random sample of the data taken **with replacement**.

TrainControl

We have seen the **train** function, but we can **control** how we train our model through a the function `trainControl()` by passing it to the argument **trControl**.

Example: In this example we are going to train a **linear model** using a 10-fold cross-validation. the syntax is shown below

```
train_lm_cv <- train(mpg ~ wt + hp,
                     data = mtcars,
                     method = "lm",
                     trControl = trainControl(
                       method = "cv",
                       number = 10,
                       verboseIter = FALSE
                     ))
```

```r
names(train_lm_cv)
```

```
 [1] "method"       "modelInfo"    "modelType"    "results"      "pred"
 [6] "bestTune"     "call"         "dots"         "metric"       "control"
[11] "finalModel"   "preProcess"   "trainingData" "resample"     "resampledCM"
[16] "perfNames"    "maximize"     "yLimits"      "times"        "levels"
[21] "terms"        "coefnames"    "xlevels"
```

```r
print(train_lm_cv)
```

```
Linear Regression

32 samples
 2 predictor

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 29, 29, 29, 28, 29, 30, ...
Resampling results:

  RMSE      Rsquared   MAE
  2.738824  0.8527979  2.240456

Tuning parameter 'intercept' was held constant at a value of TRUE
```

Repeated Cross Validation

We can do more than just one iteration of cross-validation. Repeated cross-validation gives you a better estimate of the test-set error. You can also repeat the entire cross-validation procedure. This takes longer, but gives you many more out-of-sample datasets to look at and much more precise assessments of how well the model performs.

Here is the general formula

```
train_lm_repCV <- train(mpg ~ wt + hp,
                     data = mtcars,
                     method = "lm",
                     trControl = trainControl(
                       method = "reaptedcv",
                       number = 10,
                       repeats = 5,
                       verboseIter = FALSE
                     ))
```