# caret Package for Machine Learning *useR*

Yi Han

Univeristy of FLorida

November 4, 2018

# Outline

# Outline

# Capabilities

- caret: Classification And Regression Training
- Website: http://topepo.github.io/caret/index.html
- Tools:
  - Data splitting – `createDataPartition`
  - Pre-processing – `preProcess`
  - Model building and tuning – `train`
  - Variable importance – `varImp`
  - Parallel processing – `caretNWS::trainNWS`

# Outline

# Data Splitting

- Simple Splitting
  - Function: `createDataPartition`
  - Function: `createResample`
  - Function: `createFolds`
- Maximum Dissimilarity Splitting
  - Function: `minDiss`
  - Function: `sumDiss`
- Times Series Splitting
  - Function: `createTimeSlices`

# Simple Splitting Examples

- `createDataPartition(y, p = 4/5, list = TRUE, times = 2)`
- `createResample(y, list = TRUE, times = 10)`
- `createFolds(y, k = 5, list = TRUE, returnTrain = FALSE)`

# Simple Splitting Examples

```
y=1:10
createDataPartition(y, p = 0.5, list = TRUE, times = 2)
```

```
## $Resample1
## [1]  1  2  5  6  9 10
##
## $Resample2
## [1] 1 3 4 7 8 9
```

```
createResample(y, list = TRUE, times = 2)
```

```
## $Resample1
##  [1] 2 4 5 6 6 6 7 7 8 8
##
## $Resample2
##  [1]  1  2  3  5  5  5  6  7  7 10
```

```
createFolds(y, k = 3, list = TRUE, returnTrain = FALSE)
```

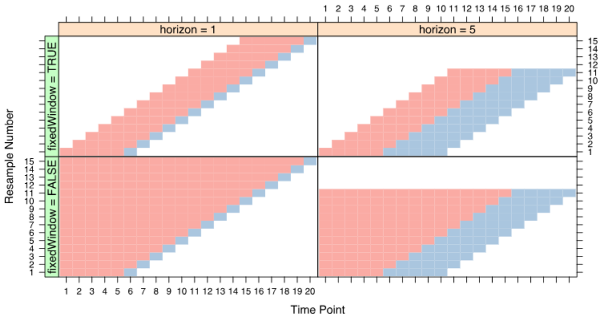
```
## $Fold1
## [1]  2  3  9 10
##
## $Fold2
## [1] 4 7 8
##
## $Fold3
## [1] 1 5 6
```

# Times Series Splitting

- `createTimeSlices(y, initialWindow=5, horizon = 1, fixedWindow = TRUE)`
  - `y`: vector of outcomes in chronological order.
  - `initialWindow`: the initial number of consecutive values in each training set sample.
  - `horizon`: The number of consecutive values in test set sample.
  - `fixedWindow`: A logical: training set size will vary over data splits or not.

# Times Series Splitting

- `createTimeSlices(y, initialWindow=5, horizon = 1, fixedWindow = TRUE)`



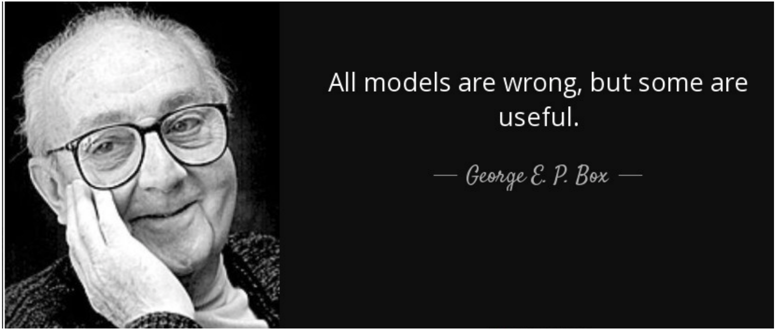
# Outline

# Pre-processing

- Transformation: Multicollinearity, interpretatability
  - `method = "pca", "BoxCox", "center", "scale"`
- Imputation: Missing data
  - KNN and Bagged Trees

# Outline

# Cliches



# Model Building

- Main function in caret package: `train`

  There are many different modeling functions in R. Some have different syntax for model training and/or prediction. The caret package started off as a way to provide a uniform interface the functions themselves, as well as a way to standardize common tasks (eliminate syntactical differences between many of the functions for building, predicting and find variable importance of models).

# Main function in caret package: train

```
control <- trainControl(method="repeatedcv",      # "cv" "boot" "LOOCV" "oob" "timeslice"
                        number=5,                 # 5-folds CV
                        repeats=2,                # 2 seperate 5-folds CV
                        classProbs = TRUE)        # whether class probabilities should be computed

C.grid <- data.frame(C = seq(0.2,0.8, length.out = 4), sigma = 0.004318767)

ML.Tune <- train(x = training.data,                 # a matrix or dataframe of predictors
                 y = as.factor(training.outcome), # vector of outcome
                 method = "svmRadial",            # "rf" "lda" "lasso" "ada" "nnet" "multinom"
                 metric="Accuracy",               # "Kappa" "RMSE" "Rsquared" select tuning par
                 preProcess="scale",              # Transformation
                 tuneGrid = C.grid,               # Specifc grid defined by outselves
                 # tuneLength = 5                 # Instead of set tunGrid, just give a length
                                                  # here C.grid = c(0.1, 1, 10, 100, 1000)
                 trControl=control)             #

ML.Tune$finalModel
```

# Prediction in Test Set

- `predict` can be used. However, it might need extra steps and `type=""` have different syntax for different methods.
  - Example: `predict(ML.Tune, newdata = test.data)`

| Function | `predict` Function Syntax |
|---:|---|
| `MASS::lda` | `predict(obj)` (no options needed) |
| `stats:::glm` | `predict(obj, type = "response")` |
| `gbm::gbm` | `predict(obj, type = "response", n.trees)` |
| `mda::mda` | `predict(obj, type = "posterior")` |
| `rpart::rpart` | `predict(obj, type = "prob")` |
| `RWeka::Weka` | `predict(obj, type = "probability")` |
| `caTools::LogitBoost` | `predict(obj, type = "raw", nIter)` |

- `predict.train` automatically handles all the details and `type=""` standardized to `"raw"` and `"prob"`

# Outline

# Variable Importance

- Generic Function: `varImp`
  - `varImp(your.SVM.model)`
  - `varImp(your.RF.model)`
  - `varImp(your.GBM.model)`
  - `varImp(your.LM.model)`
  - ...
- Plot: `plot(varImp(your.model))`

# Outline

# Parallel Processing

- Multiple processors available?
- Sister Package: `library(caretNWS)`
  Large degree of syntactical similarity: use `trainNWS` instead of `train`
- Example:

```
svmFit <- trainNWS(x=training.set,
y=train.outcome,
method="svmRadio",
tuneLength = 5,
trControl = trainNWSControl(),
scaled = FALSE)
```

# Outline

# An Example

```r
library(ISLR)
library(caret)
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```r
Weekly$Year <- as.factor(Weekly$Year)
head(Weekly)
```

```
##   Year   Lag1   Lag2   Lag3   Lag4   Lag5    Volume  Today Direction
## 1 1990  0.816  1.572 -3.936 -0.229 -3.484 0.1549760 -0.270      Down
## 2 1990 -0.270  0.816  1.572 -3.936 -0.229 0.1485740 -2.576      Down
## 3 1990 -2.576 -0.270  0.816  1.572 -3.936 0.1598375  3.514        Up
## 4 1990  3.514 -2.576 -0.270  0.816  1.572 0.1616300  0.712        Up
## 5 1990  0.712  3.514 -2.576 -0.270  0.816 0.1537280  1.178        Up
## 6 1990  1.178  0.712  3.514 -2.576 -0.270 0.1544440 -1.372      Down
```

# An Example

```
set.seed(1)
Folds <- createFolds(1:dim(Weekly)[1], k=5, returnTrain = FALSE )

Weekly.train <- Weekly[-Folds[[1]], ]
Weekly.test <- Weekly[Folds[[1]], ]


control1 <- trainControl(method = "repeatedcv",
                        number = 5,
                        repeats = 1,
                        classProbs = TRUE)

control2 <- trainControl(method = "oob")
```

# An Example

```
a <- Sys.time()
SML.model <- train(x = Weekly.train[,!names(Weekly) %in% c("Direction")],
                   y = Weekly.train[,"Direction"],
                   data = Weekly.train,
                   method = "rf",
                   ntree = 1000,
                   metric = "Accuracy",
                   verbose = FALSE,
                   tuneLength = 7,
                   trControl = control1 ,
                   importance = TRUE)
b <- Sys.time()
b-a          # control1: 14.91429 secs
```

```
## Time difference of 16.52156 secs
```

```
             # control2: 9.083024 secs
```

# An Example

```r
SML.model$finalModel
```

```
##
## Call:
##  randomForest(x = x, y = y, ntree = 1000, mtry = param$mtry, importance = TRUE,      da
##                Type of random forest: classification
##                      Number of trees: 1000
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 0.11%
## Confusion matrix:
##      Down  Up class.error
## Down  393   1 0.002538071
## Up      0 478 0.000000000
```

```r
SML.model$bestTune
```

```
##   mtry
## 1    2
```

```r
prd <- predict.train(SML.model, type = "raw", newdata = Weekly.test)
MER <- nrow(Weekly.test[prd  != Weekly.test$Direction,])/dim(Weekly.test)[1]; MER
```

```
## [1] 0
```

# An Example

```r
plot(varImp(SML.model))
```