An R Lecture from Practice: Part II

Fangda Fan

2016.5

Contents

Preparation

- Download the dataset "Rlecture_Diamonds.csv" and "Rlecture_loan.7z" (optional).

Our Goals

- Learn data operations simplified by data.table
- Learn how to summarize information of variables for cleaning
- Learn to design steps for data cleaning in data table
- Learn how to use function/for/if and other structures in R working procedure

Contents

Data Table: A Poweful Extension of Data Frame

- Install by install.packages("data.table")

```
library("data.table")
data = fread("Rlecture_Diamonds.csv")
str(data)

## Classes 'data.table' and 'data.frame':^^I53940 obs. of  11 variables:
##  $ n      : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ carat  : num  0.23 0.21 0.23 0.29 0.31 0.24 0.24 0.26 0.22 0.23 ...
##  $ cut    : chr  "Ideal" "Premium" "Good" "Premium" ...
##  $ color  : chr  "E" "E" "E" "I" ...
##  $ clarity: chr  "SI2" "SI1" "VS1" "VS2" ...
##  $ depth  : num  61.5 59.8 56.9 62.4 63.3 62.8 62.3 61.9 65.1 59.4 ...
##  $ table  : num  55 61 65 58 58 57 57 55 61 61 ...
##  $ price  : int  NA 326 NA 334 335 NA NA 337 337 338 ...
##  $ x      : num  3.95 3.89 4.05 4.2 4.34 3.94 3.95 4.07 3.87 4 ...
##  $ y      : num  3.98 3.84 4.07 4.23 4.35 3.96 3.98 4.11 3.78 4.05 ...
##  $ z      : num  2.43 2.31 2.31 2.63 2.75 2.48 2.47 2.53 2.49 2.39 ...
##  - attr(*, ".internal.selfref")=<externalptr>

summary(data)

##        n             carat           cut                color
##  Min.   :    1   Min.   :0.20   Length:53940       Length:53940
##  1st Qu.:13486   1st Qu.:0.40   Class :character   Class :character
##  Median :26970   Median :0.70   Mode  :character   Mode  :character
##  Mean   :26970   Mean   :0.80
##  3rd Qu.:40455   3rd Qu.:1.04
##  Max.   :53940   Max.   :5.01
##
##    clarity              depth          table           price
##  Length:53940       Min.   :43.0   Min.   :43.0   Min.   :  326
```

Take Subset from Data Table

- Take subset like data.frame, but using list to contain variables

```
data[2:4, list(n, carat, price)] # select row 2 to 4 and column (n, carat, x)

##    n carat price
## 1: 2  0.21   326
## 2: 3  0.23    NA
## 3: 4  0.29   334
```

- Use .N as the length of data, and operations in columns

```
data[5:.N, list(n, unit_price = price/carat)]

##            n unit_price
##     1:     5       1081
##     2:     6         NA
##     3:     7         NA
##     4:     8       1296
##     5:     9       1532
##    ---
## 53932: 53936       3829
## 53933: 53937       3829
## 53934: 53938         NA
## 53935: 53939       3206
## 53936: 53940         NA
```

Introduction of Data Table

- data.table has the form: data[i, j, by = ...] like SQL

```
data[i = price > 1000, j = list(count = .N, carat = mean(carat)), by = list(cut)]

##          cut count carat
## 1:      Fair  1114 1.081
## 2:     Ideal 10814 0.876
## 3: Very Good  6507 0.978
## 4:      Good  2795 0.997
## 5:   Premium  7794 1.061
```

R (data.table)	i	j	by
SQL	WHERE	SELECT	GROUP BY

- data.table is faster than R default, especially with big data

```
system.time(read.csv("Rlecture_Diamonds.csv")) # timing (second): read csv with R default

##    user  system elapsed
##    0.35    0.00    0.36

system.time(fread("Rlecture_Diamonds.csv")) # timing (second): read csv with data.table

##    user  system elapsed
##    0.21    0.00    0.23
```

Sort Data Table

- Sort data by setkey()

```
setkey(data, cut)
data[,list(n,cut)]

##             n       cut
##     1:      9      Fair
##     2:     92      Fair
##     3:     98      Fair
##     4:    124      Fair
##     5:    125      Fair
##    ---
## 53936: 53922 Very Good
## 53937: 53923 Very Good
## 53938: 53933 Very Good
## 53939: 53934 Very Good
## 53940: 53938 Very Good
```

- Select rows by key value directly

```
data["Good", .N]

## [1] 4906

data["Good", mult = "first"]

##    n carat  cut color clarity depth table price    x    y    z
## 1: 3  0.23 Good     E     VS1  56.9    65    NA 4.05 4.07 2.31
```

Column Apply in Data Table

- Apply functions into each column with lapply and .SD

```
class(data[, price])

## [1] "integer"

data[,lapply(.SD, class)]

##          n   carat       cut     color   clarity   depth   table   price
## 1: integer numeric character character character numeric numeric integer
##          x       y       z
## 1: numeric numeric numeric
```

- Take a subset of columns with .SDcols

```
data[,lapply(.SD, class), .SDcols = -c(2:5)]

##          n   depth   table   price       x       y       z
## 1: integer numeric numeric integer numeric numeric numeric
```

Contents

Data Summarization

Goal: Summarize the characters of many variables of big data

- Amount of Information vs. Readability
- Various Variable Types vs. General Method/Representation
- Elaboration vs. Quickness/Easiness

Our Solution: Use data.table to convert each variable into key statistics, and create a new summarized data.table

Summarize Variables (1): Type and Size

- First identify the type of each variable

```
data[,lapply(.SD, class)]

##           n   carat       cut     color   clarity   depth   table   price
## 1: integer numeric character character character numeric numeric integer
##           x       y       z
## 1: numeric numeric numeric
```

- count NA values of each variable

```
x = data[,price]
nonNA = function(x){
  return(sum(!is.na(x)))
}
nonNA(data[,price])

## [1] 39708

data[,lapply(.SD, nonNA)] # count of non-NA values of variables

##          n carat   cut color clarity depth table price     x     y     z
## 1: 53940 53940 53940 53940   53940 53940 53940 39708 53940 53940 53940
```

Summarize Variables (2): Numeric Statistics

- Convert numeric variable into numeric statistics

```
summary_num = function(x){
  if(class(x) == "character")
    return(NA)
  else
    x_trans = c(mean(x, na.rm = T), sd(x, na.rm = T), quantile(x, na.rm = T))
  return(x_trans)
}
summary_num(data[,price])

##                0%   25%   50%   75%  100%
##  3928  3993   326   949  2397  5302 18818

data[,lapply(.SD, summary_num)]

##         n carat cut color clarity depth table price     x     y      z
## 1: 26971 0.798  NA    NA      NA 61.75 57.46  3928  5.73  5.73  3.539
## 2: 15571 0.474  NA    NA      NA  1.43  2.23  3993  1.12  1.14  0.706
## 3:     1 0.200  NA    NA      NA 43.00 43.00   326  0.00  0.00  0.000
## 4: 13486 0.400  NA    NA      NA 61.00 56.00   949  4.71  4.72  2.910
## 5: 26971 0.700  NA    NA      NA 61.80 57.00  2397  5.70  5.71  3.530
## 6: 40455 1.040  NA    NA      NA 62.50 59.00  5302  6.54  6.54  4.040
## 7: 53940 5.010  NA    NA      NA 79.00 95.00 18818 10.74 58.90 31.800
```

Summarize Variables (3): Frequency Table

- We create a function to get the most 5 frequent values

```
summary_value = function(x){
  freq = sort(table(x), decreasing = TRUE)
  return(names(freq)[1:5])
}
data[,lapply(.SD, summary_value)]

##     n carat       cut color clarity depth table price    x    y    z
## 1: 1   0.3     Ideal     G     SI1    62    56   605 4.37 4.34  2.7
## 2: 2  0.31   Premium     E     VS2  61.9    57   828 4.34 4.37 2.69
## 3: 3  1.01 Very Good     F     SI2  61.8    58   776 4.33 4.35 2.71
## 4: 4   0.7      Good     H     VS1  62.2    59   789 4.38 4.33 2.68
## 5: 5  0.32      Fair     D    VVS2  62.1    55   666 4.32 4.32 2.72
```

- Get the frequency of most 5 frequent values, tail and NA

```
summary_freq = function(x){
  freq = sort(table(x), decreasing = TRUE)
  return(c(freq[1:5], sum(freq[-(1:5)]), sum(is.na(x))))
}
data[,lapply(.SD, summary_freq)]

##         n carat   cut color clarity depth table price     x     y     z
## 1:      1  2604 21551 11292   13065  2239  9881   103   448   437   767
## 2:      1  2249 13791  9797   12258  2163  9724    96   437   435   748
## 3:      1  2242 12082  9542    9194  2077  8369    95   429   425   738
## 4:      1  1981  4906  8304    8171  2039  6572    91   428   421   730
## 5:      1  1840  1610  6775    5066  2020  6268    88   425   414   697
## 6: 53935 43024     0  8230    6186 43402 13126 39235 51773 51808 50260
## 7:      0     0     0     0       0     0     0 14232     0     0     0
```

Summarize Variables (4): Merge

- Use a list to collect these summaries, each reorganized by variable names as index

```
fL = list(class, nonNA, summary_num, summary_value, summary_freq)
summaryL = list()
for(i in 1:length(fL)){
  summaryL[[i]] = data[,lapply(.SD, fL[[i]])]
  summaryL[[i]] = data.table(t(summaryL[[i]]), keep.rownames=TRUE)
  setkey(summaryL[[i]], rn)
}
```

- Merge a list of data.tables into one data.table with Reduce()

```
summary_data = Reduce(function(X,Y){X[Y]}, summaryL)
colnames(summary_data) = c("variable", "type", "N",
                           "mean", "sd", "min", "Q1", "median", "Q3", "max",
                           paste("value", 1:5), paste("freq", c(1:5, "others", "NA")))
write.csv(summary_data, "Rlecture_Diamonds_summary.csv", row.names = FALSE, na = "")
```

Contents

Why Data Cleaning

- A key step to prepare big data for machine learning prediction
- Standardize raw data for statistical models to understand:
    - For programming generalization: convert data to numeric matrices for general-model use (beyond linear models in R)
    - Solve mathematical problems in data: sparsity, outliers, ...
- Difficulty: the size and complexity of variables in data

```
# data_big = fread("Rlecture_loan.csv")
# str(data_big)
# summary(data_big)
```

Variable Problem	Example	Model Effect	Test
Non-numeric	characters, time	Error, Info loss	Data Type
Sparsity	Identical values	Noise	Count Frequency
Collinearity	Two similar variables	Instability	Correlation Matrix
NA values	NA, NA notations	Error	Count NAs
Distribution Bias	exponential, outliers	Instability	Numeric statistics

Data Cleaning (1): Numeralization

Our Goal: Convert all non-numeric variables into numeric varibles

- Non-numeric variables with prior information (ordinal, time, ...) $\rightarrow$ map into numeric values with a key-value data.table

```
x = data[,cut]
mapDT = data.table(c("Fair", "Good", "Very Good", "Premium", "Ideal"), 1:5, key = "V1")
map = function(x, mapDT){
  op = data.table(x, key = "x")[mapDT]
  return(op[,V2])
}
data[, cut := map(cut, mapDT)]
```

- All other non-numeric variables $\rightarrow$ 0-1 dummy variables

```
options(na.action='na.pass')
data_clean = data.table(model.matrix( ~ . , data = data))
dim(data_clean)

## [1] 53940    23
```

Data Cleaning (2): Detect and Delete Sparse Variables

- Check the number of NAs and most frequent values of each variable, define it as sparity

```
# data_clean[,lapply(.SD, summary_freq)]
sparsity = function(x){
  op = sort(table(x), decreasing = TRUE)[1] + sum(is.na(x))
  return(op/length(x))
}
(sparse = data_clean[ , lapply(.SD, sparsity)])

##    (Intercept)        n  carat cut  colorE colorF colorG colorH colorI
## 1:           1 1.85e-05 0.0483 0.4   0.818  0.823  0.791  0.846  0.899
##    colorJ clarityIF claritySI1 claritySI2 clarityVS1 clarityVS2
## 1:  0.948     0.967      0.758       0.83      0.849      0.773
##    clarityVVS1 clarityVVS2  depth table price       x      y      z
## 1:       0.932       0.906 0.0415 0.183 0.266 0.00831 0.0081 0.0142
```

- Delete all variables with sparsity nearly 1

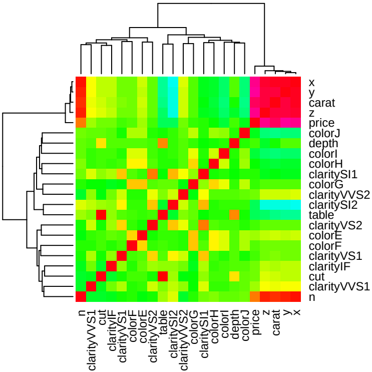
```
data_clean = data_clean[, data_clean[ ,sparse < 0.9999], with = FALSE]
dim(data_clean)

## [1] 53940    22
```

Data Cleaning (3a): Detect Collinear Variables

- Use correlation matrix to detect collinearity

```
cm = cor(data_clean, use = "pairwise.complete.obs")
heatmap(cm, col = rainbow(100), scale = "none")
write.csv(cm, "Rlecture_Diamonds_cm.csv")
## Then open "Rlecture_Diamonds_cm.csv" in Microsoft Excel (2010+):
## Ctrl+A -> tag: Home -> Conditional Formatting -> Color Scales
```



Data Cleaning (3b): Delete Collinear Variables

- Delete collinear variables with lower triangular of correlation matrix

```
(collinear = apply(lower.tri(cm) & (abs(cm) > 0.95), 1, sum))

##            n        carat          cut       colorE       colorF       colorG
##            0            0            0            0            0            0
##       colorH       colorI       colorJ    clarityIF   claritySI1   claritySI2
##            0            0            0            0            0            0
##   clarityVS1   clarityVS2  clarityVVS1  clarityVVS2        depth        table
##            0            0            0            0            0            0
##        price            x            y            z
##            0            1            2            3

data_clean = data_clean[, (names(collinear)[collinear == 0]), with = FALSE]
dim(data_clean)

## [1] 53940    19
```

Data Cleaning (4): Fill NAs

- Check NA values

```
data_clean[, lapply(.SD, function(x){sum(is.na(x))})]

##    n carat cut colorE colorF colorG colorH colorI colorJ clarityIF
## 1: 0     0   0      0      0      0      0      0      0         0
##    claritySI1 claritySI2 clarityVS1 clarityVS2 clarityVVS1 clarityVVS2
## 1:          0          0          0          0           0           0
##    depth table price
## 1:     0     0 14232
```

- Fill NA values with the median/mean/0/... of the column (Not necessary for predicted variable Y)

```
fillNA = function(x){
  a = median(x, na.rm = TRUE)
  x[is.na(x) == TRUE] = a
  return(x)
}
data_clean = data_clean[, lapply(.SD, fillNA)]
```

Data Cleaning (5): Distribution Normalization

- One method: Standardize all variables with mean 0 and standard deviation 1

```
data_clean = data_clean[, lapply(.SD, function(x){(x - mean(x))/sd(x)})]
```

- Another method: Standardize all variables to satisfy standard normal distributioon

```
data_clean = data_clean[, lapply(.SD, function(x){qnorm((frank(x)-0.5)/length(x))})]
```

Contents

Preparation

- Download the dataset “Rlecture_Diamonds.csv” and “Rlecture$_{D}iamonds_{p}redict_{t}rue.csv''$.

Our Goals

- Learn the basic idea of machine learning
- Learn to divide training/validation sets and cross-validation
- Learn general procedure to build models for prediction

Contents

A Machine Learning Question

- If we want to evaluate the unknown price of diamonds as precisely as possible
- We can use the information of the sample with known price

What is Machine Learning?

- Machine learning explores the study and construction of algorithms that can learn from and make predictions on data. (Wikipedia)
- Model: $Y_{n \times q} = f(X_{n \times p}) + \varepsilon_{n \times q}$
- Objective: find best $\hat{f}$ to minimize $||Y, \hat{f}(X)||$
  - $||Y, \hat{f}(X)||$: the distance between Y and $\hat{f}(X)$
  - Y is also called label of data
- Example: for linear regression, $q = 1$, $f(X) = \beta_0 + X\beta$ and $||Y, f(X)|| = \sum_{i=1}^{n}(y_i - f(X)_i)^2$
- In our case,
  - $Y_{n \times 1}$: price of diamonds
  - $X_{n \times p}$: other variables
  - $f()$: the model we want to choose
  - $||Y, f(X)||$: the distance criteria we want to choose

How to Choose the Distance?

- Here are some examples of distance

Distance	Expression	Use
L^2 (Euclidean)	$\sum_{i=1}^{n}(y_i - f(X)_i)^2$	Regression
L^1 (Absolute)	$\sum_{i=1}^{n} \lvert y_i - f(X)_i \rvert$	Regression
L^0 (0-1)	$\sum_{i=1}^{n} 1(y_i - f(X)_i \neq 0)$	Classification
Cross Entropy	$\sum_{i=1}^{n} \log(f(X)_i{}^{y_i}(1 - f(X)_i)^{1-y_i})$	Classification

- In mathematics, L^p distance is denoted by $d^p = \|Z\|_p^p = \sum_{i=1}^{n} |z_i|^p$
- As a regression problem, we can choose L^2 distance (also called SSE) as our criteria of evaluating precision

Create a Distance Function in R

- We create an average L^p distance (use mean instead of sum)

```
Lp_dist = function(y, y_hat, p = 2){
  return(mean(abs(y - y_hat)^p))
}
```

- We examine it by a weather forecast example

```
rain_forecast = c(0.9, 0.2, 0.3, 0.6, 0.1) # the probability of rain forecasted
rain = c(1, 0, 1, 0, 0) # the true weather
Lp_dist(rain, rain_forecast) # default: L2 distance (MSE)

## [1] 0.182

Lp_dist(rain, rain_forecast, p = 1) # L1 distance (MAE)

## [1] 0.34

Lp_dist(rain, rain_forecast, p = 1e-10) # L0 distance (Error Rate)

## [1] 1

Lp_dist(rain, rain_forecast > 0.5, p = 1e-10) # L0 distance with discretizing classification

## [1] 0.4
```

Data Reading and Cleaining

- We construct X and Y with index "n". For the exponential distribution of Y, we use log(price) instead of price

```
library("data.table")
data = fread("Rlecture_Diamonds.csv")
Y = data[,list(n, price = log(price))]
```

- Clean X

```
# 1. Numeralization (0-1 Dummy Variable for Non-numeric, With Interaction Term)
options(na.action='na.pass')
X = data.table(model.matrix( ~ . - price - n, data = data))
# 2. Delete Sparsity
sparsity = function(x){return((sort(table(x), decreasing = TRUE)[1]+sum(is.na(x)))/length(x))}
icol = X[, lapply(.SD, sparsity) < 0.9999]
X = X[ , icol, with = FALSE]
# 3. Delete Collinearity
cm = cor(X, use = "pairwise.complete.obs")
icol = X[, !apply((lower.tri(cm) & abs(cm) > 0.95), 1, sum)]
X = X[ , icol, with = FALSE]
# 4. Fill NA
X = X[, lapply(.SD, function(x){x[is.na(x)] = median(x, na.rm = TRUE); return(x)})]
# 5. Standardization
X = X[, lapply(.SD, function(x){return((x - mean(x))/sd(x))})]
# Append index
X = cbind(data[,list(n)], X)
```

Contents

Training and Validation Set

- How do we know we catch the true effect $f(X)$ by model $\hat{f}(X)$?
    - Example: we can fit $Y_{n\times 1}$ perfectly by an n-degree polynomial
- Solution: divide labeled data (X, Y) into two parts (X_T, Y_T), (X_V, Y_V)
    - (X_T, Y_T): Training set, for training model
    - (X_V, Y_V): Validation set, for validating model

Create Training/Validation Sets in R

- Use labeled data for training/validation sets and leave unlabeled for prediction

```
set.seed(56789)
group = Y[!is.na(price), list(n, igroup = rank(runif(.N)) %% 10 + 1)]
Xp = as.matrix(X[is.na(Y$price)])[,-1]
select_set = function(X, Y, group, i){
  setkey(X, "n")
  setkey(Y, "n")
  id_valid = group[igroup == i, list(n)]
  id_train = group[igroup != i, list(n)]
  Xt = as.matrix(X[id_train])[,-1]
  Yt = as.matrix(Y[id_train])[,-1]
  Xv = as.matrix(X[id_valid])[,-1]
  Yv = as.matrix(Y[id_valid])[,-1]
  return(list(train = list(X = Xt, Y = Yt), valid = list(X = Xv, Y = Yv)))
}
XY = select_set(X, Y, group, 1)
```

- Try to train and validate with a linear model

```
model = lm(Y ~ X, data = XY$train)
Lp_dist(XY$train$Y, model$fitted.values) # training error

## [1] 0.113

yhat = predict(model, newdata = XY$valid)
Lp_dist(XY$valid$Y, yhat) # validation error

## [1] 0.12
```

Cross Validation

- Idea: Evaluate the prediction error of a model more precisely and estimate its range
- Divide same data into multiple pairs of training/validation sets
- K-folds cross validation:
  1. Cut the index of data randomly into K groups with equal size
  2. Take one group as validation set, and others as training set to train a model
  3. Do step 2 for K times, in each choose a different validation set, and get K models
  4. Use K models to predict the unlabeled data, take the average of prediction result

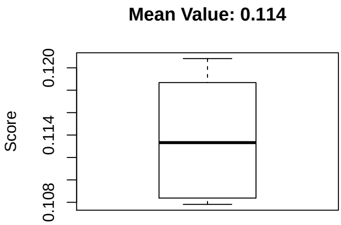
Do Cross Validation in R

- We use a list with a for-loop to build cross validation models

```
modelL = list()
score = c()
for(i in 1:max(group[,igroup])){
  XY = select_set(X, Y, group, i)
  modelL[[i]] = lm(Y ~ X, data = XY$train)
  yhat = predict(modelL[[i]], newdata = XY$valid)
  score = c(score, Lp_dist(XY$valid$Y, yhat))
}
score

##  [1] 0.120 0.111 0.108 0.113 0.108 0.121 0.119 0.118 0.108 0.114

boxplot(score, ylab = "Score", main = paste("Mean Value:", round(mean(score), 3)))
```



Contents

Over-Fitting

- Idea: $\hat{f}(X) = \hat{f}_f(X) + \hat{f}_\varepsilon(X) \to Y = f(X) + \varepsilon$
- Variance:
  - $Var(Y - \hat{f}(X)) = Var((f - \hat{f}_f)(X)) \downarrow + Var(\varepsilon) + Var(\hat{f}_\varepsilon(X)) \uparrow$
- Over-fitting: $Var(\hat{f}_\varepsilon(X)) \uparrow$ at a speed faster than $Var((f - \hat{f}_f)(X)) \downarrow$
  - The model spend too much complexity on fitting random errors
- To prevent over-fitting, we often use loss and penalty functions in model structures

Loss and Penalty Functions

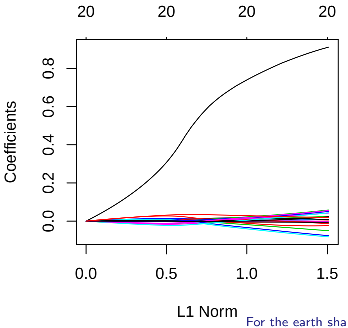
- Idea: Try to prevent over-fitting through controlling the complexity of model $\hat{f}$
- Loss function: $||Y, f(X)||_L$
- Penalty function: $||f||_P$
- Objective: find best $\hat{f}$ to minimize $||Y, \hat{f}(X)||_L + \lambda||\hat{f}||_P$
- $\lambda \geq 0$ is the penalty coefficient: the larger the λ become, the simpler the $\hat{f}$ will be

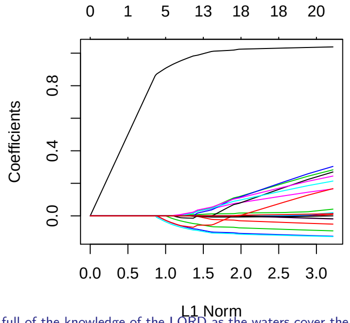
Models (GLM)	Loss	Penalty
Linear Regression	$\|\|Y - f(X)\|\|_2^2$	None
Stepwise Selection (AIC)	$\|\|Y - f(X)\|\|_2$	$\|\|\beta\|\|_0^0$
Ridge Regression	$\|\|Y - f(X)\|\|_2^2$	$\|\|\beta\|\|_2^2$
Lasso Regression	$\|\|Y - f(X)\|\|_2^2$	$\|\|\beta\|\|_1$
Quantile Regression	$\|\|Y - f(X)\|\|_1$	None

Penalized Linear Models: Ridge and Lasso

- Install by install.packages("glmnet")
- ElasticNet: $||f||_P = \frac{1-\alpha}{2}||\beta||_2^2 + \alpha||\beta||_2$

```
library("glmnet")
model_ridge = glmnet(XY$train$X, XY$train$Y, alpha = 0) # Ridge Regression
plot(model_ridge)
model_lasso = glmnet(XY$train$X, XY$train$Y, alpha = 1) # Lasso Regression
plot(model_lasso)
```

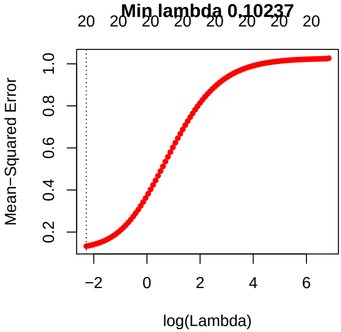


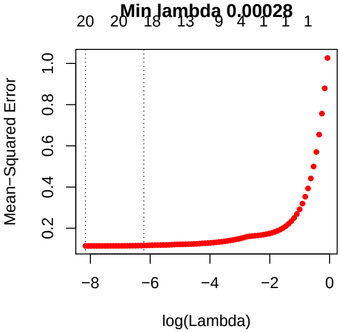


Select Best Parameters for Models

- We use cross validation inside the glmnet package to select best lambda

```
modelcv_ridge = cv.glmnet(XY$train$X, XY$train$Y, alpha = 0) # Ridge Regressuib
plot(modelcv_ridge, main = paste("Min lambda", round(modelcv_ridge$lambda.min, 5)))
modelcv_lasso = cv.glmnet(XY$train$X, XY$train$Y, alpha = 1) # Lasso Regression
plot(modelcv_lasso, main = paste("Min lambda", round(modelcv_lasso$lambda.min, 5)))
```





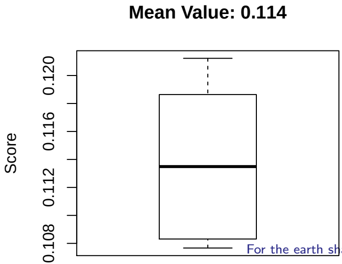
Train Model with the Best Parameter(s)

- Cross validate model with selected lambda

```
modelL = list()
score = c()
for(i in 1:max(group[,igroup])){
  XY = select_set(X, Y, group, i)
  modelL[[i]] = glmnet(XY$train$X, XY$train$Y, alpha = 1, lambda = modelcv_lasso$lambda.min)
  yhat = predict(modelL[[i]], newx = XY$valid$X)
  score = c(score, Lp_dist(XY$valid$Y, yhat))
}
score

##  [1] 0.120 0.111 0.108 0.113 0.108 0.121 0.119 0.118 0.108 0.114

boxplot(score, ylab = "Score", main = paste("Mean Value:", round(mean(score), 3)))
```



Cross Model Prediction

- Predict Y with average of each cross model

```
Yp_cross = sapply(modelL, function(model){return(predict(model, newx = Xp))})
Yp = apply(Yp_cross, 1, mean)
data_predict = fread("Rlecture_Diamonds_predict_true.csv")
Yp_true = data_predict[, log(price)]
score_predict = Lp_dist(Yp_true, Yp)
```

```
score_predict_cross = apply(Yp_cross, 2, function(x){return(Lp_dist(Yp_true, x))})
boxplot(score_predict_cross, main = paste("Predicted Score:", round(score_predict, 3)))
abline(h = mean(score_predict_cross))
abline(h = score_predict, col = "red")
```