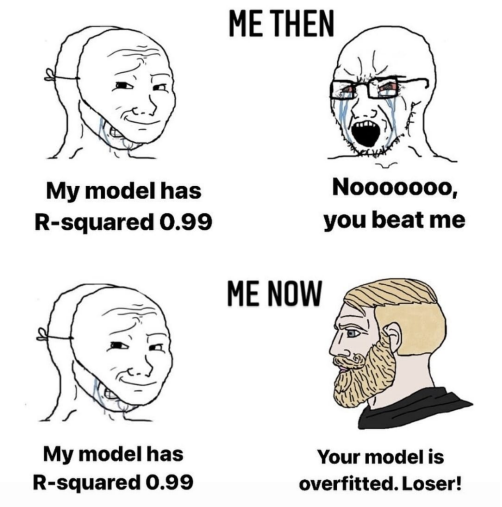
Regularização : Ridge Regression e Lasso

Anteriormente, estudamos o modelo de regressão linear, tanto o modelo simples como o múltiplo, na qual usamos várias variáveis independentes $(x_1, ..., x_n)$ para prever uma variável dependente y.

Quando estamos criando um modelo linear, vamos perceber que ao adicionar mais parâmetros, o R^2 sempre irá aumentar, dando-nos a falsa impressão de que quanto mais complexo for o modelo criado, melhor. No entanto, caso nosso modelo esteja desnecessariamente complexo, iremos cometer um erro chamado **overfitting**. Vamos falar mais detalhadamente sobre isso abaixo:

Overfitting e Underfitting



Comecemos por um exemplo: vamos utilizar o conjunto de dados mtcars, disponível no pacote base do R. Para mais informações acerca do conjunto, digite "?mtcars" no terminal do R.

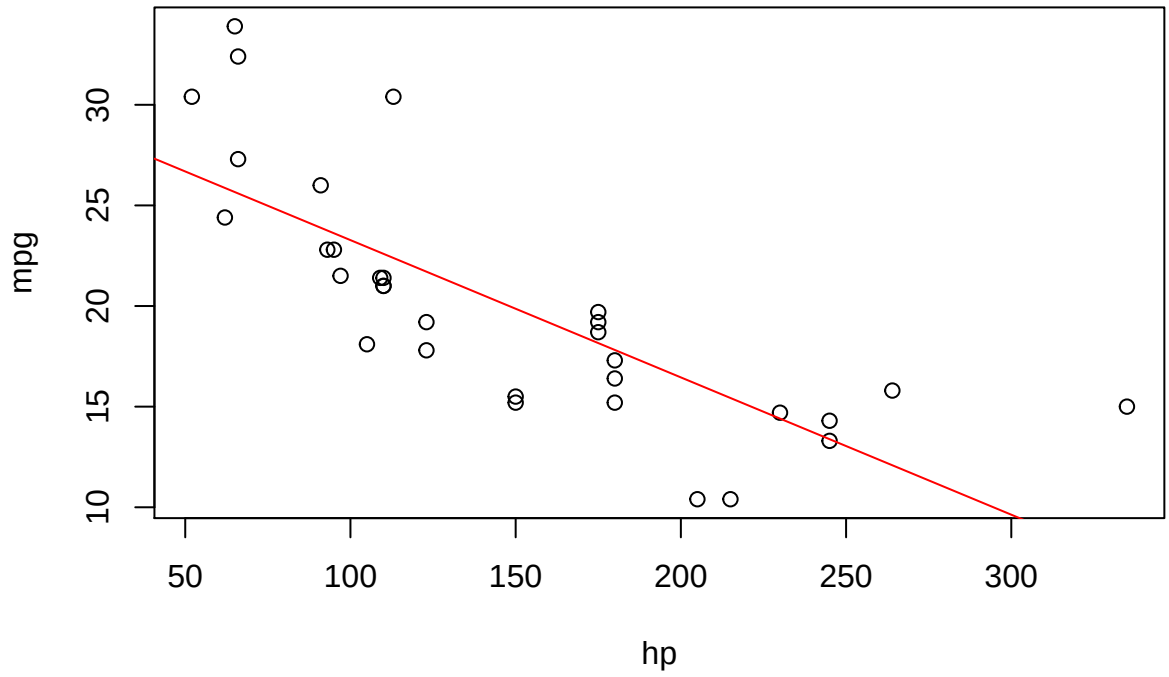
```
head(mtcars)
```

```
##                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

Esse conjunto de dados contém as informações de 11 variáveis de 32 carros. Estamos interessados em prever a quantidades de milhas rodadas por galão de combustível (variável mpg) com base nas outras variáveis.

Vamos criar um modelo de regressão linear simples que estima a variável mpg utilizando a potência, em cavalos, do carro (variável hp).

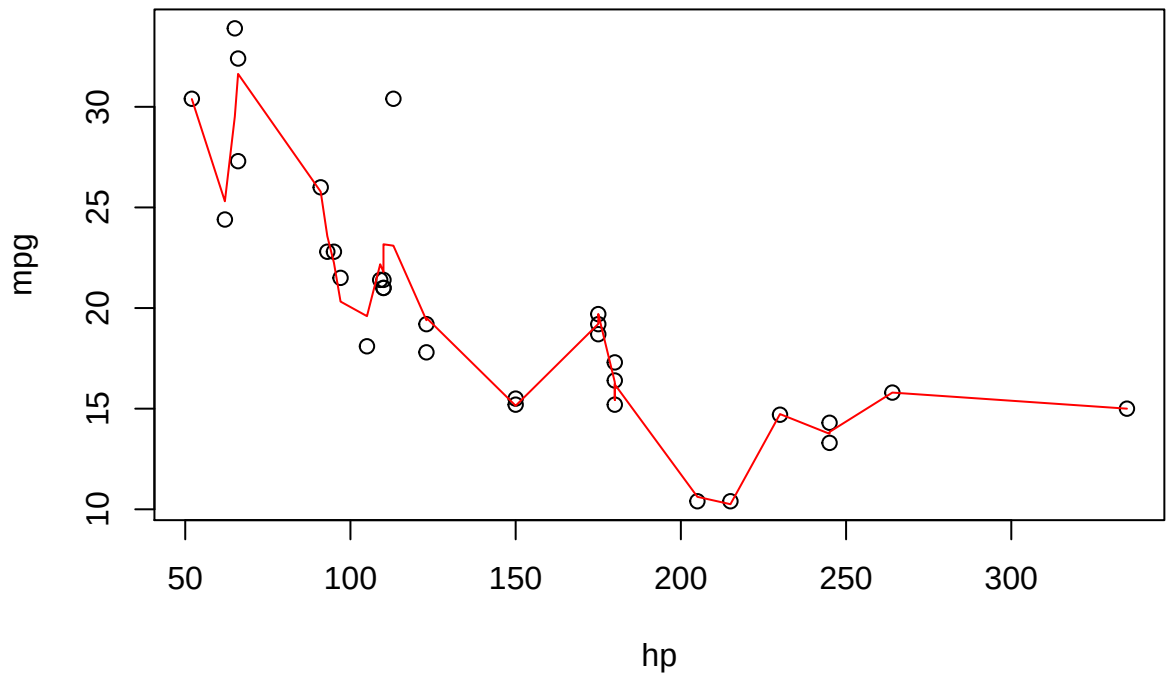
```
attach(mtcars)

modelo1 <- lm(mpg ~ hp,
              data = mtcars)
plot(mpg~hp)
abline(modelo1, col ='red')
```



Vemos que o modelo linear simples não captura bem o relacionamento entre as duas variáveis, que parece ser mais complexo. Nesse caso, dizemos que ocorreu **underfitting**, isto é, o modelo é simples demais.

Vamos criar um segundo modelo, com 15 termos polinomiais.

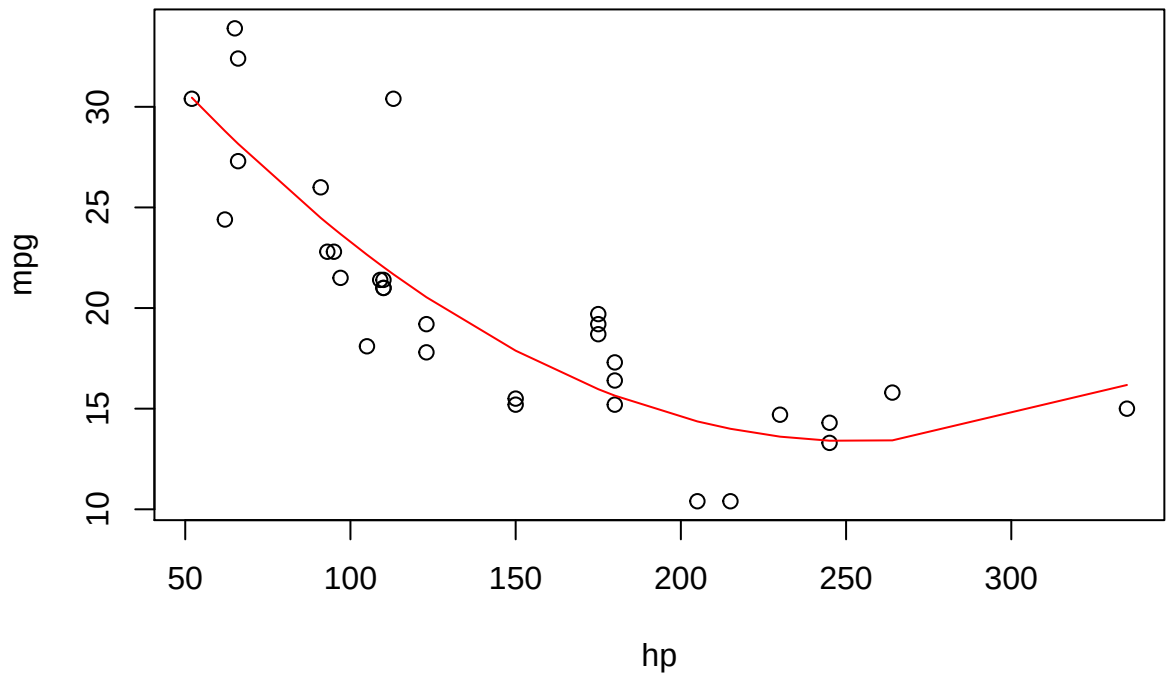
```
modelo2 <- lm(mpg ~ poly(hp,15),
              data = mtcars)
plot(mpg~hp)
lines(sort(hp),fitted(modelo2)[order(hp)], col ='red')
```



À primeira vista, esse novo modelo pode parecer ideal, afinal, nossas predições estão muito próximas dos valores observados da variável mpg. No entanto, esse modelo "se esforçou" demais para se aproximar dos dados de treino e não teria nenhuma utilidade para previsão de valores futuros. Dizemos que ocorreu **overfitting**, isto é, nosso modelo é complexo demais.

Vamos criar um terceiro modelo, que utiliza um polinômio de grau menor:

```
modelo3 <- lm(mpg ~ poly(hp,2),
              data = mtcars)
plot(mpg~hp)
lines(sort(hp),fitted(modelo3)[order(hp)], col ='red')
```



Esse modelo é complexo o suficiente para capturar a relação entre as variáveis, mas não complexo demais de modo que esteja viciado nos dados de teste. Nesse caso, dizemos que o modelo está **bem ajustado**.

Nesse exemplo, utilizamos um modelo linear simples para facilitar a visualização. De maneira geral, quando temos um modelo com várias variáveis, devemos ter cuidado para não adicionar variáveis demais e criar um modelo desnecessariamente compelxo. Ainda utilizando o mesmo conjunto de dados, um modelo que tenta prever a quantidade de milhas rodadas por galão de combustível com base em 7 variáveis não necessariamente será melhor que um que utiliza duas, por exemplo.

Para achar um equilíbrio entre a simplicidade e a complexidade do modelo, existem três alternativas:

- Selecionar manualmente quais variáveis se deseja incuir no modelo (como fizemos agora)
- Utilizar um algoritmo de seleção de variáveis (não falaremos sobre esses algoritmos agora)
- Utilizar métodos de regularização, sendo eles:
  - Lasso (também chamado regularização L1)
  - Ridge Regression (também chamado regularização L2)

Regularização e função custo

Em vez de reduzir o número de variáveis utilizadas na predição, os métodos de regularzação adicionam todas as variáveis disponíveis e, em seguida, "encolhem"(regularizam) os valores dos coeficientes associados a essas variáveis, reduzindo sua influência no modelo.

Vamos explicar esse conceito mais detalhadamente:

Suponha que tenhamos um conjunto de dados de com n observações e p variáveis.

Quando criamos um modelo de regressão linear simples, escolhemos os parâmetros $\beta_0, \beta_1, ..., \beta_p$ de modo a minimizar a função abaixo, que representa a soma do quadrado das distâncias entre os valores preditos para a variável de interesse y e o valor real delas.

$$L(\beta_0, \beta_1, ..., \beta_p) = \sum_{i=1}^{n} (y_i - (\beta_0 + \beta_1 x_1 + ... + \beta_n x_n))^2$$

Essa função é chamada **função custo** da regressão linear.

Quando realizamos uma regularização, tamém estamos interessados em reduzir uma função custo. No caso específico do Ridge Regression, a função custo é dada por:

$$L(\beta_0, \beta_1, ..., \beta_p) = \sum_{i=1}^{n} (y_i - (\beta_0 + \beta_1 x_1 + ... + \beta_n x_n))^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

Isto é, a função custo é similar a da regressão linear, no entanto, há um incremento do custo com o aumento dos valores dos coeficientes $\beta_1, ..., \beta_p$ (note que o coeficiente β_0 não é incluído na penalização).

Dessa maneira, nosso algoritmo irá diminuir a soma dos quadrados dos erros, como na regressão linear, mas ao mesmo tempo, irá ser penalizada por inflar os valores dos coeficientes $\beta_1, ..., \beta_p$.

O que determinará o grau de penalização pelo aumento dos valores dos coeficientes é a constante λ, chamada *tuning parameter*, ou parâmetro de sintonização. Esse valor deve ser maior que zero.

Caso λ seja muito grande, nosso modelo irá encolher todos os coeficientes do nosso modelo em direção a zero, tornando-o inútil. Por outro lado, caso λ seja muito próximo de 0, nosso modelo se tornará indistinguível de uma regressão linear.

Será necessário testar vários valores de λ e escolher o valor que resulta no melhor modelo. Falaremos disso com mais detalhes à frente.

Para o Lasso, a função custo é dada por:

$$L(\beta_0, \beta_1, ..., \beta_p) = \sum_{i=1}^{n} (y_i - (\beta_0 + \beta_1 x_1 + ... + \beta_n x_n))^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

O Lasso funciona de maneira semelhante ao Ridge Regression. A principal difereça entre os dois métodos é que, enquanto o Lasso pode reduzir o valor absoluto dos coeficientes envolvidos no modelo até zero, o algoritmo Ridge Regression pode reduzir o valor absoluto até um valor muito pequeno, mas nunca a zero, isto é, na regularização Ridge, nenhuma variável será removida do modelo, enquanto no Lasso, algumas variáveis podem ser retiradas.

Vale ressaltar que não existe um algoritmo de regularização universalmente melhor. A cada caso, deve-se avaliar qual dos dois oferece um melhor resultado.

Aplicação no R

Ridge Regression

Vamos utilizar novamente o conjunto de dados mtcars, sendo as variáveis indepententes e depentente as mesmas utilizadas anteriormente.

Vamos chamar a biblioteca **glmnet**, que performa a regularização, assim como a **dplyr**, para modificar nossos dados de maneira a adequá-los ao algoritmo de regularização.

```
library(glmnet)
```

```
## Loading required package: Matrix

## Loaded glmnet 4.1-1
```

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

A função glmnet funciona de maneira diferente da função lm, utilizada para gerar modelos de regressão linear. Para o funcionamento correto da regularização, é necessário que:

- A variável de intersse esteja centrada (com esperança zero) e em formato de matriz
- Se especifique um valor para λ
- As variáveis indepententes estejam padronizadas (com esperança 0 e variância 1) e em formato de matriz

A função glmnet realiza tanto o Ridge Regression como o Lasso, para escolher qual algoritmo utilizar:

- defina o parâmetro alpha da função glmnet para 0 para Ridge Regression
- defina o parâmetro alpha da função glmnet para 1 para Lasso

As variáveis independentes podem ser padronizadas no momento da criação do modelo. Vamo inicialmente centralizar a variável y e transformá-la em matriz e transformar o data frame de variáveis indepententes em uma matriz.

Podemos fazer isso transformando a variável em $y - \bar{y}$. Note que:

$$E[y - \bar{y}] = E[y] - E[\bar{y}] = \bar{y} - \bar{y} = 0$$

Utilizando o pacote dplyr:

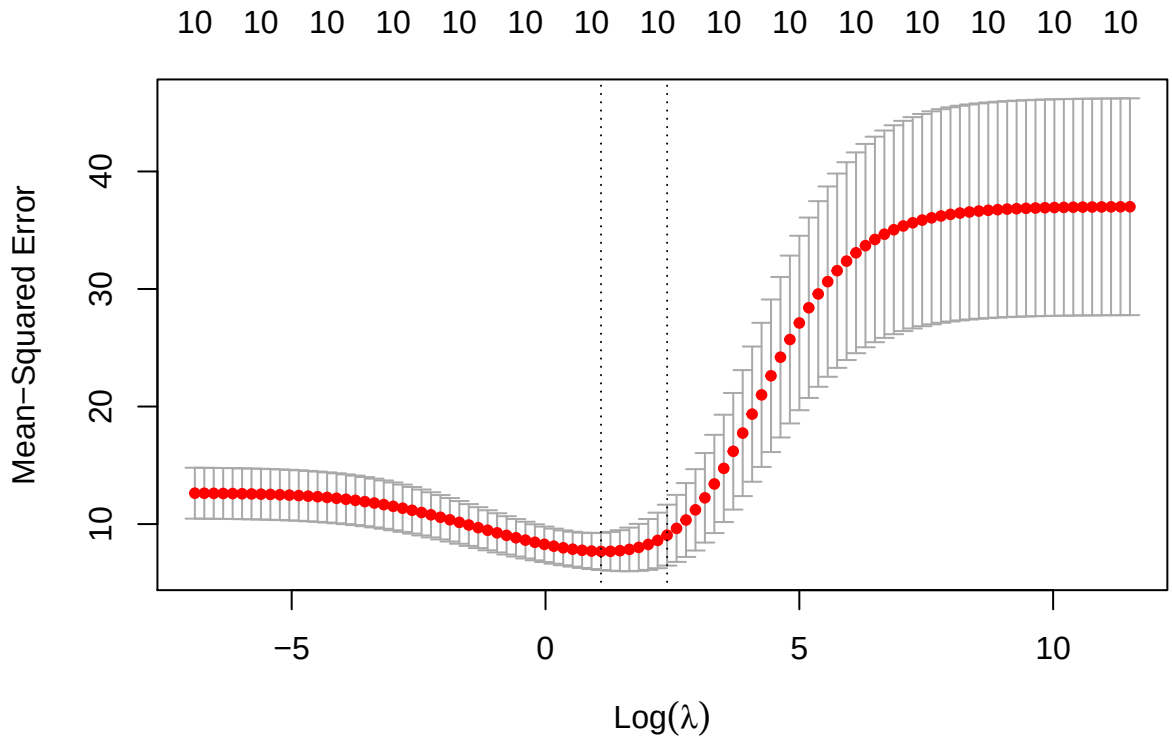
```
y <- mtcars %>%
  select(mpg) %>%
  scale(center = TRUE, scale = FALSE) %>%
  as.matrix()

X <- mtcars %>%
  select(-mpg) %>%
  as.matrix()
```

Vamos usar a função **cv.glmnet**, que faz uma validação cruzada, para escolher o valor de λ que gera o menor erro médio quadrático (MSE).

Testaremos valores de lambda de 10^{-3} a 10^{5}.

Note que aqui temos que colocar o parâmerto "standardize = TRUE" porque nossas variáveis $x_1, ..., x_n$ ainda não foram padronizadas.

```
lambdas_candidatos <- 10^seq(-3,5,length.out = 100)

ridge_cv <- cv.glmnet(X,
                      y,
                      alpha = 0,
                      lambda = lambdas_candidatos,
                      standardize = TRUE,
                      nfolds = 10)

plot(ridge_cv)
```



Escolheremos o lambda que resulta no menor MSE. Podemos obtê-lo com o comando abaixo

```
lambda_escolhido <- ridge_cv$lambda.min
```

Finalmente, geramos o modelo regularizado com a função **glmnet**. Para especificar que queremos a regularização Ridge, optamos pelo parâmetro alpha = 0:

```
mod_rr <- glmnet(X,
                 y,
                 alpha = 0,
                 lambda = lambda_escolhido,
                 standardize = TRUE)
```

Para ver os coeficientes do modelo, usamos o código:

```
mod_rr$beta
```

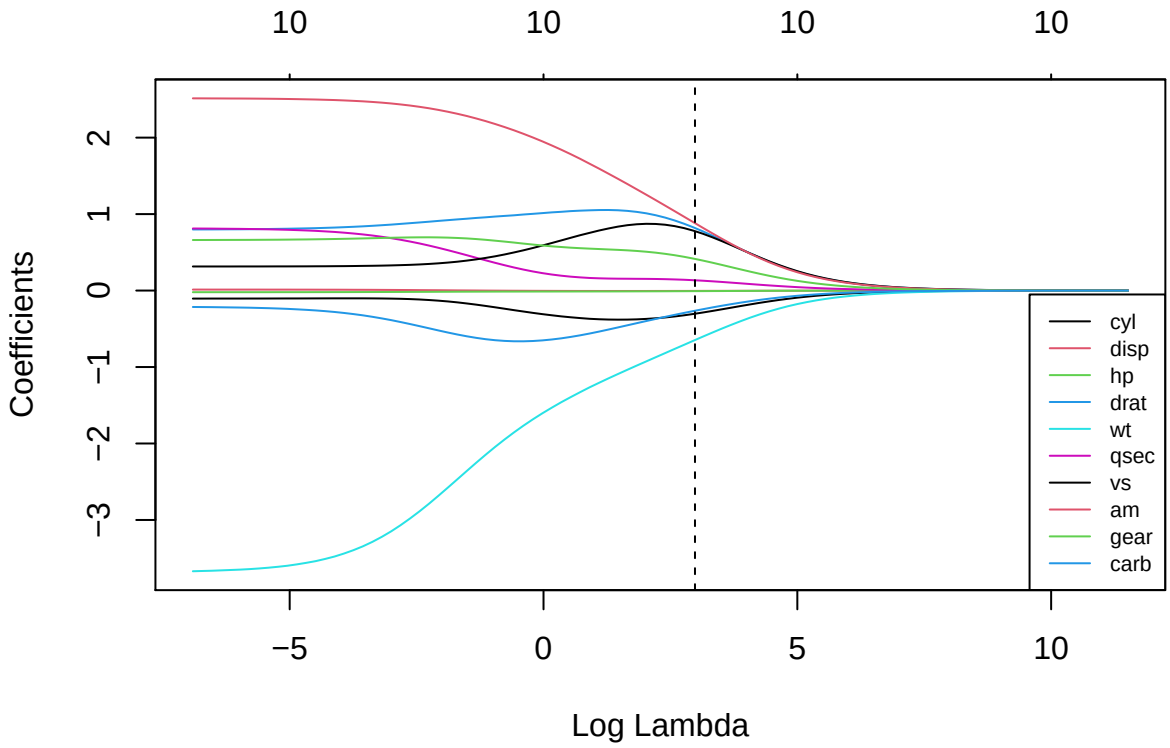
```
## 10 x 1 sparse Matrix of class "dgCMatrix"
##                s0
## cyl  -0.374250969
## disp -0.005317757
## hp   -0.011519902
## drat  1.055565066
## wt   -1.207393974
## qsec  0.160397178
## vs    0.784538697
## am    1.594311112
## gear  0.541902172
## carb -0.536015354
```

Podemos prever os valores da variável de interesse com a função **predict**:

```
y_pred_rr <- predict(mod_rr,X)
```

Podemos visualizar o efeito do valor de λ no valor dos coeficientes do modelo:

```
modelos <- glmnet(X,
                  y,
                  alpha = 0,
                  lambda = lambdas_candidatos,
                  standardize = TRUE)

plot(modelos,xvar = 'lambda')
legend('bottomright',lwd = 1, col = 1:6, legend = colnames(X), cex = .7)
abline(v = lambda_escolhido, lty = 2)
```



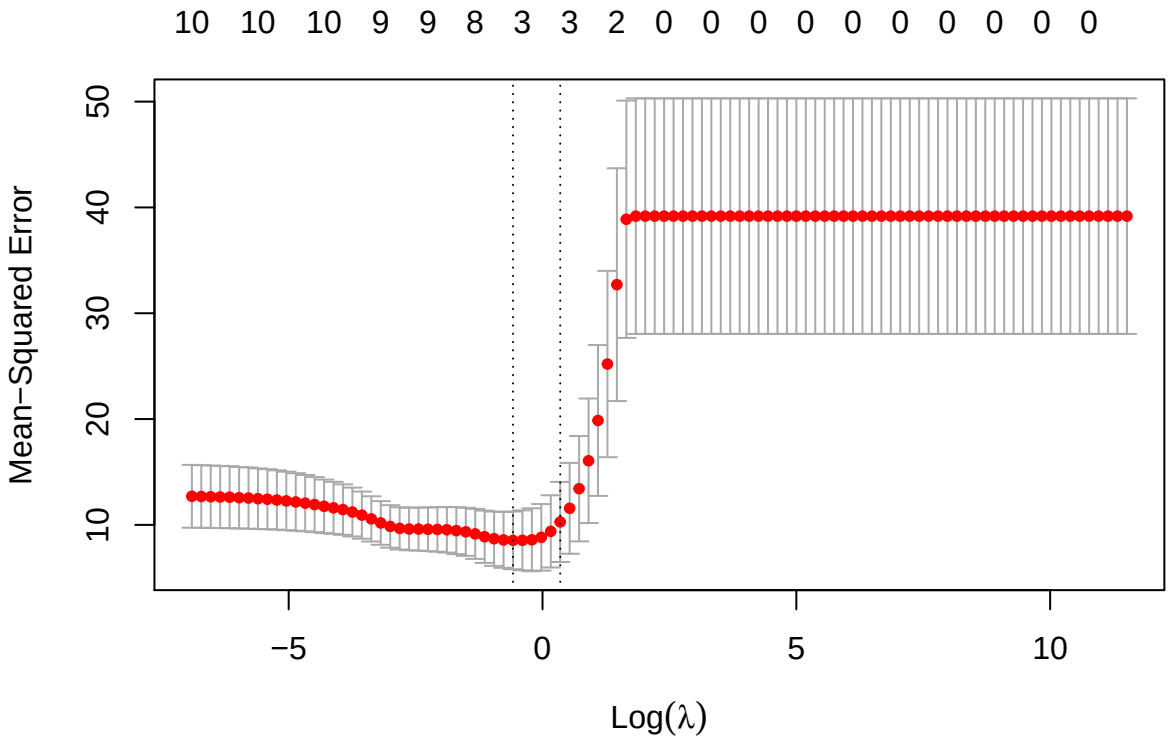
Como podemos ver, quanto maior o valor de lambda, mais próximos os coeficientes ficam de zero. O valor de lambda escolhido está representado pela linha vertical pontilhada.

Lasso

Para realizar a regularização Lasso, repetimos o processo anterio. Mudamos somente o parâmetro alpha = 1 nas funções cv.glmnet e glmnet.

Validação cruzada e escolha do lambda:

```
lambdas_candidatos_lasso <- 10^seq(-3,5,length.out = 100)

lasso_cv <- cv.glmnet(X,
                      y,
                      alpha = 1,
                      lambda = lambdas_candidatos_lasso,
                      standardize = TRUE,
                      nfolds = 10)

plot(lasso_cv)
```



Criação do modelo:

```
lambda_lasso_escolhido <- lasso_cv$lambda.min

mod_lasso <- glmnet(X,
                    y,
                    alpha = 1,
                    lambda = lambda_lasso_escolhido,
                    standardize = TRUE)
```

Previsão dos valores:

```
y_pred_lasso <- predict(mod_lasso,X)
```

Vamos visualizar os valores dos coeficientes:

```
mod_lasso$beta
```

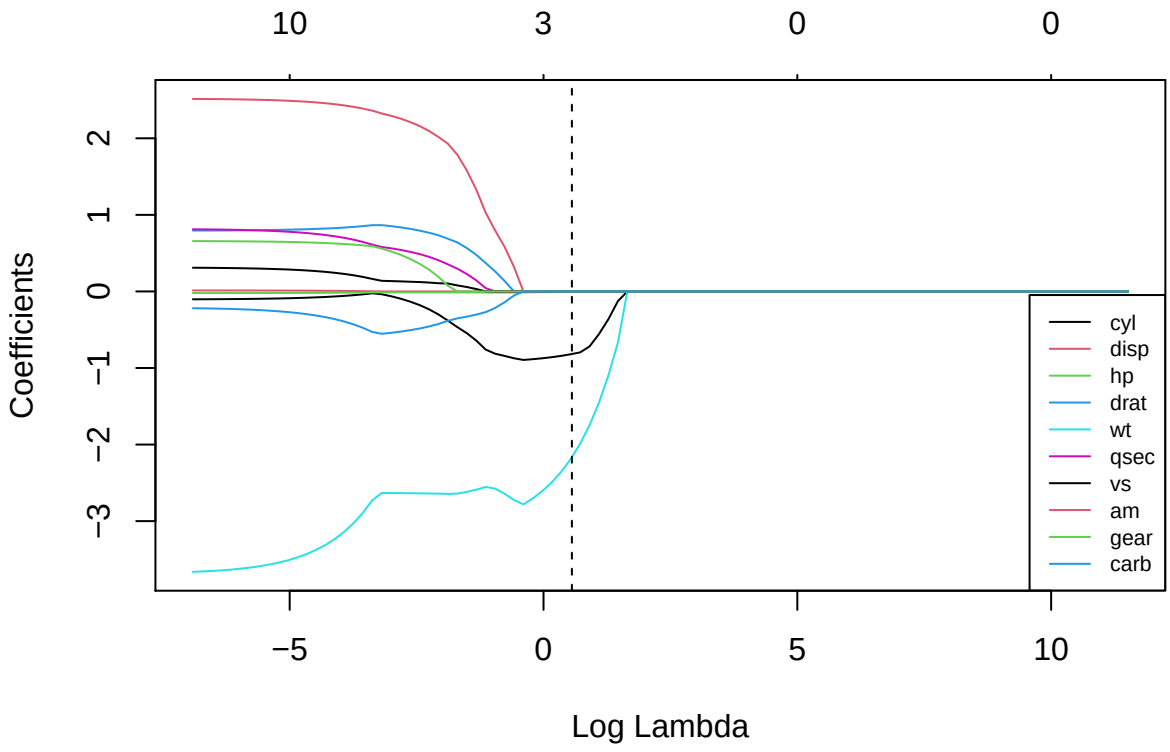
```
## 10 x 1 sparse Matrix of class "dgCMatrix"
##                s0
## cyl  -0.87120283
## disp  .
## hp   -0.01395311
## drat  .
## wt   -2.72506271
```

```
## qsec  .
## vs    .
## am    0.32012360
## gear  .
## carb -0.05209847
```

Como podemos ver, alguns coeficientes estão com valor zero, isto é, as variáveis correspondentes a esses coeficientes foram removidas do modelo. Isso é possível utilizando o algoritmo do Lasso.

Finalmente, vamos visualizar os valores dos coeficientes em relação a lambda.

```
modelos_lasso <- glmnet(X,
                        y,
                        alpha = 1,
                        lambda = lambdas_candidatos_lasso,
                        standardize = TRUE)

plot(modelos_lasso,xvar = 'lambda')
legend('bottomright',lwd = 1, col = 1:6, legend = colnames(X), cex = .7)
abline(v = lambda_lasso_escolhido, lty = 2)
```



Comparando resultados : Lasso e Ridge Regression

Vamos calcular o R^2 associado a cada um dos modelos regularizados.

O R^2 pode ser obtido calculando-se o quadrado da correlação entre os valores preditos da variável de interesse e a variável de interesse.

```
rsq_rr <- cor(y,y_pred_rr)^2
rsq_lasso <- cor(y,y_pred_lasso)^2

valores <- cbind("R2" = c(rsq_rr,rsq_lasso))
rownames(valores) <- c("Ridge Regression","Lasso")
valores
```

```
##                          R2
## Ridge Regression 0.8524086
## Lasso            0.8463680
```

Nesse caso, vemos que os dois modelos apresentam uma performance muito parecida, com uma leve vantagem da regularização Ridge.