Estatística Aplicada II

Aluno: Victor Lima

Data: 20/06/2023

Primeira Lista de Exercícios

Com a base de dados “imoveiscwbav” obter os seguintes resultados com o auxílio do “R”

1. Elaborar os modelos Ridge, Lasso e ElasticNet e estimar uma predição para cada um dos modelos (elabore seus próprios valores para a predição). Apresente os parâmetros estimados e os valores resultantes da predição.

1 - Carregar conjunto de dados e organizar para utilizar nos modelos

```
load("imoveiscwbav.RData")
imoveis <- imoveiscwbav
```

Carregando pacotes necessários

```
library(caret)
```

```
## Carregando pacotes exigidos: ggplot2
```

```
## Carregando pacotes exigidos: lattice
```

```
library(glmnet)
```

```
## Carregando pacotes exigidos: Matrix
```

```
## Loaded glmnet 4.1-7
```

Fixando randomização para tornar constante os resultados gerados

```
set.seed(999)
```

Separando os índices para treino e teste (80%)

```
index = sample(1:nrow(imoveis),0.8*nrow(imoveis))
train = imoveis[index,] # Dados de treino
test = imoveis[-index,] # Dados de teste
```

Selecionando as colunas que devem ser padronizadas

```
cols = c('price', 'age', 'parea', 'tarea', 'bath', 'ensuit', 'garag', 'plaz', 'park', 'tran
s', 'kidca', 'school', 'health', 'bike')
```

Padronizando os dados que não são binários

```
pre_proc_val <- preProcess(train[,cols], method = c("center", "scale"))
train[,cols] = predict(pre_proc_val, train[,cols])
test[,cols] = predict(pre_proc_val, test[,cols])
```

Criando um vetor com as variáveis em uso

```
cols_reg = c('price', 'age', 'parea', 'tarea', 'bath', 'ensuit', 'garag', 'plaz','park', 'tra
ns', 'kidca', 'school', 'health', 'bike', 'barb', 'balc', 'elev', 'fitg', 'party', 'categ')
```

Gerando dummies para organizar o conjunto de dados no objeto matriz

```
dummies <- dummyVars(price ~ age+parea+tarea+bath+ensuit+garag+plaz+park+trans+kidca+school+h
ealth+bike+
                       barb+balc+elev+fitg+party+categ,data = imoveis[,cols_reg])
train_dummies = predict(dummies, newdata = train[,cols_reg])
test_dummies = predict(dummies, newdata = test[,cols_reg])
```

O conjunto de dados da matriz para o modelo

```
x = as.matrix(train_dummies)
y_train = train$price
x_test = as.matrix(test_dummies)
y_test = test$price
```

2 - Organizando os dados que serão usados para testar os modelos

Como os valores do conjunto de dados são padronizados, também temos que padronizar os dados que queremos prever

```
age = (14-pre_proc_val[["mean"]][["age"]])/pre_proc_val[["std"]][["age"]] # age = 14
parea = (120-pre_proc_val[["mean"]][["parea"]])/pre_proc_val[["std"]][["parea"]] # parea = 12
0
tarea = (180-pre_proc_val[["mean"]][["tarea"]])/pre_proc_val[["std"]][["tarea"]] # tarea = 18
0
bath = (3-pre_proc_val[["mean"]][["bath"]])/pre_proc_val[["std"]][["bath"]] # bath = 3
ensuit = (3-pre_proc_val[["mean"]][["ensuit"]])/pre_proc_val[["std"]][["ensuit"]] # ensuit =
3
garag = (2-pre_proc_val[["mean"]][["garag"]])/pre_proc_val[["std"]][["garag"]] # garag = 2
plaz = (0-pre_proc_val[["mean"]][["plaz"]])/pre_proc_val[["std"]][["plaz"]] # plaz = 0
park = (2-pre_proc_val[["mean"]][["park"]])/pre_proc_val[["std"]][["park"]] # park = 2
trans = (2-pre_proc_val[["mean"]][["trans"]])/pre_proc_val[["std"]][["trans"]] # trans = 2
kidca = (1.5-pre_proc_val[["mean"]][["kidca"]])/pre_proc_val[["std"]][["kidca"]] # kidca = 1.
5
school = (1-pre_proc_val[["mean"]][["school"]])/pre_proc_val[["std"]][["school"]] # school =
0.5
health = (0.5-pre_proc_val[["mean"]][["health"]])/pre_proc_val[["std"]][["health"]] # health
= 0.5
bike = (0.5-pre_proc_val[["mean"]][["bike"]])/pre_proc_val[["std"]][["bike"]] # bike = 0.5
barb = 0 # barb = 0
balc = 0 # balc = 0
elev = 0 # elev = 0
fitg = 0 # fitg = 0
party = 1 # party = 1
categ = 1 # categ = 1
```

Construindo uma matriz de dados para previsão

```
our_pred = as.matrix(data.frame(age=age, parea=parea, tarea=tarea, bath=bath, ensuit=ensuit,
garag=garag, plaz=plaz, park=park, trans=trans, kidca=kidca, school=school, health=health, bi
ke=bike, barb=barb, balc=balc, elev=elev, fitg=fitg, party=party, categ=categ))
```

3 - Rotina para calcular R² e RMSE a partir de valores verdadeiros e previstos

```
eval_results <- function(true, predicted, df) {
  SSE <- sum((predicted - true)^2)
  SST <- sum((true - mean(true))^2)
  R_square <- 1 - SSE / SST
  RMSE = sqrt(SSE/nrow(df))
  # Métricas de desempenho do modelo
  data.frame(RMSE = RMSE, Rsquare = R_square)
}
```

4 - Regressão Ridge

Cálculo do valor lambda ótimo

```
lambdas_ridge <- 10^seq(2, -3, by = -.1)
ridge_lamb <- cv.glmnet(x, y_train, alpha = 0, lambda = lambdas_ridge)
best_lambda_ridge <- ridge_lamb$lambda.min
best_lambda_ridge
```

```
## [1] 0.1
```

Treinamento do modelo

Modelo de estimativa

```
ridge_reg = glmnet(x, y_train, nlambda = 25, alpha = 0, family = 'gaussian', lambda = best_la
mbda_ridge)
summary(ridge_reg)
```

```
##           Length Class     Mode
## a0         1     -none-    numeric
## beta      19     dgCMatrix S4
## df         1     -none-    numeric
## dim        2     -none-    numeric
## lambda     1     -none-    numeric
## dev.ratio  1     -none-    numeric
## nulldev    1     -none-    numeric
## npasses    1     -none-    numeric
## jerr       1     -none-    numeric
## offset     1     -none-    logical
## call       7     -none-    call
## nobs       1     -none-    numeric
```

Os valores dos parâmetros

```
ridge_reg[["beta"]]
```

```
## 19 x 1 sparse Matrix of class "dgCMatrix"
##                  s0
## age    -0.175009203
## parea   0.160006840
## tarea   0.183490452
## bath    0.069478471
## ensuit  0.192235574
## garag   0.218999954
## plaz    0.028104135
## park   -0.063308411
## trans   0.028448215
## kidca  -0.005033790
## school  0.015631712
## health -0.003032506
## bike   -0.024204958
## barb   -0.042828040
## balc    0.129439904
## elev   -0.192286423
## fitg    0.250293752
## party   0.079681726
## categ   0.452327896
```

Fazendo previsões e avaliando o modelo

Previsão e avaliação de dados de treino

```
predictions_train_ridge <- predict(ridge_reg, s = best_lambda_ridge, newx = x)
```

Métricas de desempenho do modelo para os dados de treino

```
eval_results(y_train, predictions_train_ridge, train)
```

RMSE	Rsquare
<dbl>	<dbl>
0.4303306	0.8143859

1 row

Previsão e avaliação em dados de teste

```
predictions_test_ridge <- predict(ridge_reg, s = best_lambda_ridge, newx = x_test)
```

Métricas de desempenho do modelo para os dados de teste

```
eval_results(y_test, predictions_test_ridge, test)
```

RMSE	Rsquare
<dbl>	<dbl>
0.4229928	0.7672156

1 row

Fazendo a previsão

```
predict_our_ridge <- predict(ridge_reg, s = best_lambda_ridge, newx = our_pred)
predict_our_ridge
```

```
##             s1
## [1,] 0.2934091
```

O resultado é um valor padronizado, vamos convertê-lo em valor nominal, consistente com o banco de dados original O valor predito é de R$ 1.112.961,00

```
price_pred_ridge=(predict_our_ridge*pre_proc_val[["std"]][["price"]])+pre_proc_val[["mean"]]
[["price"]]
price_pred_ridge
```

```
##           s1
## [1,] 1112961
```

Intervalos de confiança para nosso exemplo

```
n_ridge <- nrow(train)
m_ridge <- price_pred_ridge
s_ridge <- pre_proc_val[["std"]][["price"]]
dam_ridge <- s_ridge/sqrt(n_ridge)
CIlwr_ridge <- m_ridge + (qnorm(0.025))*dam_ridge
CIupr_ridge <- m_ridge - (qnorm(0.025))*dam_ridge
```

```
CIlwr_ridge
```

```
##           s1
## [1,] 1063056
```

Valor superior: R$ 1.063.056,00

```
CIupr_ridge
```

```
##           s1
## [1,] 1162867
```

Valor Inferior: R$ 1.162.867,00

5 - Regressão Lasso

Encontrando o melhor lambda e definindo alfa = 1 para implementar a regressão de Lasso

```
lambdasLasso <- 10^seq(2, -3, by = -.1)
lasso_lamb <- cv.glmnet(x, y_train, alpha = 1, lambda = lambdasLasso, standardize = TRUE, nfo
lds = 5)
```

Melhor lambda

```
best_lambda_lasso <- lasso_lamb$lambda.min
best_lambda_lasso
```

```
## [1] 0.003981072
```

Treinamento do modelo

Refazendo o modelo com o melhor lambda

```
lasso_model <- glmnet(x, y_train, alpha = 1, lambda = best_lambda_lasso, standardize = TRUE)
```

Como visualizar os parâmetros estimados

```
lasso_model[["beta"]]
```

```
## 19 x 1 sparse Matrix of class "dgCMatrix"
##                 s0
## age    -0.179545946
## parea   0.169869309
## tarea   0.189014919
## bath    0.039625919
## ensuit  0.213521091
## garag   0.233782044
## plaz    0.029286995
## park   -0.072069254
## trans   0.020457431
## kidca  -0.002468589
## school  0.010717577
## health  .
## bike   -0.018107430
## barb   -0.047803480
## balc    0.123747291
## elev   -0.201518163
## fitg    0.261917729
## party   0.066753658
## categ   0.496729293
```

Fazendo previsões e avaliando o modelo

Previsão e avaliação de dados de treino

```
predictions_train_Lasso <- predict(lasso_model, s = best_lambda_lasso, newx = x)
```

Métricas de desempenho do modelo para os dados de treino

```
eval_results(y_train, predictions_train_Lasso, train)
```

RMSE <dbl>	Rsquare <dbl>
0.4290149	0.8155192

1 row

Previsão e avaliação em dados de teste

```
predictions_test_Lasso <- predict(lasso_model, s = best_lambda_lasso, newx = x_test)
```

Métricas de desempenho do modelo para os dados de teste

```
eval_results(y_test, predictions_test_Lasso, test)
```

RMSE <dbl>	Rsquare <dbl>
0.4215564	0.768794

1 row

Previsão para o nosso exemplo, fazendo as previsões com base nos mesmos parâmetros da regressão de Lasso

```
predict_our_lasso <- predict(lasso_model, s = best_lambda_lasso, newx = our_pred)
predict_our_lasso
```

```
##             s1
## [1,] 0.3214761
```

Novamente, a informação que ele retorna é padronizada, temos que convertê-la para um valor compatível com o conjunto de dados original Valor predito: 1.127.815,00

```
price_pred_lasso=(predict_our_lasso*pre_proc_val[["std"]][["price"]])+pre_proc_val[["mean"]]
[["price"]]
price_pred_lasso
```

```
##           s1
## [1,] 1127815
```

Intervalos de confiança para o nosso exemplo

```
n_lasso <- nrow(train)
m_lasso <- price_pred_lasso
s_lasso <- pre_proc_val[["std"]][["price"]]
dam_lasso <- s_lasso/sqrt(n_lasso)
CIlwr_lasso <- m_lasso + (qnorm(0.025))*dam_lasso
CIupr_lasso <- m_lasso - (qnorm(0.025))*dam_lasso
```

```
CIlwr_lasso
```

```
##           s1
## [1,] 1077910
```

Valor superior: R$ 1.077.910,00

```
CIupr_lasso
```

```
##           s1
## [1,] 1177721
```

Valor Inferior: R$ 1.177.721,00

6 - Regressão ElasticNet

Definir controle de treinamento

```
train_cont_elasticNet <- trainControl(method = "repeatedcv", number = 10, repeats = 5, search
= "random", verboseIter = FALSE, returnData = FALSE)
```

Treinamento do modelo

Não temos o parâmetro “alpha”, porque a Regressão ElasticNet vai encontrá-lo automaticamente, cujo valor estará entre 0 e 1. O parâmetro “lambda” também é escolhido por validação cruzada.

```
elastic_reg <- train(price ~ age+parea+tarea+bath+ensuit+garag+plaz+park+trans+kidca+school+h
ealth+bike+barb+balc+elev+fitg+party+categ, data = train, method = "glmnet", tuneLength = 10,
trControl = train_cont_elasticNet)
```

```
## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo,
## : There were missing values in resampled performance measures.
```

Melhor parâmetro de ajuste

```
elastic_reg$bestTune
```

	alpha <dbl>	lambda <dbl>
8	0.9606575	0.006726726

1 row

Fazendo previsões e avaliando o modelo

Previsão e avaliação de dados de treino

```
predictions_train_elasticNet <- predict(elastic_reg, x)
```

Métricas de desempenho do modelo para os dados de treino

```
eval_results(y_train, predictions_train_elasticNet, train)
```

RMSE <dbl>	Rsquare <dbl>
0.4294217	0.8151692

1 row

Previsão e avaliação em dados de teste

```
predictions_test_elasticNet <- predict(elastic_reg, x_test)
```

Métricas de desempenho do modelo para os dados de teste

```
eval_results(y_test, predictions_test_elasticNet, test)
```

RMSE <dbl>	Rsquare <dbl>
0.4219819	0.768327

1 row

Fazendo a previsão

```
predict_our_elastic <- predict(elastic_reg,our_pred)
predict_our_elastic
```

```
## [1] 0.3147937
```

O resultado é um valor padronizado, vamos convertê-lo em valor nominal, consistente com o banco de dados original

O valor predito é de R$ 1.124.279,00

```
price_pred_elastic=(predict_our_elastic*pre_proc_val[["std"]][["price"]])+pre_proc_val[["mea
n"]][["price"]]
price_pred_elastic
```

```
## [1] 1124279
```

Intervalos de confiança para o nosso exemplo

```
n_elasticnet <- nrow(train)
m_elasticnet <- price_pred_elastic
s_elasticnet <- pre_proc_val[["std"]][["price"]]
dam_elasticnet <- s_elasticnet/sqrt(n_elasticnet)
CIlwr_elasticnet <- m_elasticnet + (qnorm(0.025))*dam_elasticnet
CIupr_elasticnet <- m_elasticnet - (qnorm(0.025))*dam_elasticnet
```

```
CIlwr_elasticnet
```

```
## [1] 1074373
```

Valor superior: R$ 1.074.373,00

```
CIupr_elasticnet
```

```
## [1] 1174184
```

Valor Inferior: R$ 1.174.184,00

7 - Organizando os resultados finais

Exibindo em um dataframe todos os resultados para facilitar a avaliação dos modelos

```
results <- data.frame(model=c("ridge", "lasso", "elasticnet"), predict=c(price_pred_ridge, pr
ice_pred_lasso, price_pred_elastic), lowerConfidence=c(CIlwr_ridge, CIlwr_lasso, CIlwr_elasti
cnet), upperConfidence=c(CIupr_ridge, CIupr_lasso, CIupr_elasticnet), stringsAsFactors=FALSE)
results
```

model <chr>	predict <dbl>	lowerConfidence <dbl>	upperConfidence <dbl>
ridge	1112961	1063056	1162867
lasso	1127815	1077910	1177721
elasticnet	1124279	1074373	1174184

3 rows

Fim.