NBA 4920/6921 Lecture 15

Elastic Net Application

Murat Unal

10/21/2021

```r
rm(list=ls())
options(digits = 3, scipen = 999)
library(tidyverse)
library(ISLR)
library(ggplot2)
library(cowplot)
library(ggcorrplot)
library(stargazer)
library(corrr)
library(lmtest)
library(sandwich)
library(MASS)
library(car)
library(jtools)
library(caret)
library(leaps)
library(future.apply)
library(glmnet)
Hitters <- ISLR::Hitters
Hitters <- na.omit(Hitters)
```

- Previous test RMSEs:

```r
RMSE <- matrix(NA,ncol = 1, nrow = 8)
rownames(RMSE) <- c("rmse.ridge.lambdabig",
"rmse.ridge.lambda4","rmse.ridge.lambda0",
"rmse.ridge.lambdabest","rmse.lasso.lambda1se",
"rmse.lasso.lambdabest", "rmse.elnet.lambda1se",
"rmse.elnet.lambdabest")
RMSE[1:6,1] <- c(405,296,300,292,334,297)
RMSE
```

```
                      [,1]
rmse.ridge.lambdabig   405
rmse.ridge.lambda4     296
rmse.ridge.lambda0     300
rmse.ridge.lambdabest  292
rmse.lasso.lambda1se   334
rmse.lasso.lambdabest  297
rmse.elnet.lambda1se    NA
rmse.elnet.lambdabest   NA
```

Elastic net

- Now, there are two parameters to tune: λ and α.
- The **glmnet** package allows to tune λ via cross-validation for a fixed α, but it does not support α-tuning.

- Let's write our own loop that does the tuning
- First, we create a common `fold_id`, which just allows us to apply the same CV folds to each model.
- We then create a tuning grid that searches across a range of αs from 0-1, and empty columns where we'll dump our model results into.

```
# maintain the same folds across all models
fold_id <- sample(1:10, size = length(y[train]),
                                    replace=TRUE)


# search across a range of alphas
tuning_grid <- data.frame(
  alpha      = seq(0, 1, by = .1),
  mse_min    = NA,mse_1se    = NA,
  lambda_min = NA,lambda_1se = NA)
```

- Now we can iterate over each α value, apply a CV elastic net, and extract the minimum and one standard error MSE values and their respective λ values.

```
for(i in seq_along(tuning_grid$alpha)  ) {
  # fit CV model for each alpha value
  fit <- cv.glmnet(x[train,], y[train],
                   alpha = tuning_grid$alpha[i],
                           foldid = fold_id)


  # extract MSE and lambda values
  tuning_grid$mse_min[i]    <- fit$cvm[fit$lambda==
                                     fit$lambda.min]
  tuning_grid$mse_1se[i]    <- fit$cvm[fit$lambda==
                                     fit$lambda.1se]
  tuning_grid$lambda_min[i] <- fit$lambda.min
  tuning_grid$lambda_1se[i] <- fit$lambda.1se
}
```

```r
tuning_grid %>% arrange(mse_min)
```

alpha	mse_min	mse_1se	lambda_min	lambda_1se
0.0	129229	162541	24.31	4451
1.0	129734	161372	2.79	115
0.9	129775	161084	3.11	128
0.8	129932	160957	3.49	144
0.7	130133	161130	3.64	165
0.1	130311	164170	135.91	959
0.6	130320	161324	3.87	192
0.2	130418	162119	89.83	526
0.5	130565	161697	4.23	231
0.3	130710	163761	65.73	385
0.4	130749	162338	5.29	289

- Extract the optimum *alpha* and λ values

```
best.index <- which.min(tuning_grid$mse_min)
best.alpha <- tuning_grid[best.index ,"alpha"]
best.lambda <- tuning_grid[best.index ,"lambda_min"]
best.lambda.1se <- tuning_grid[best.index ,"lambda_1se"]

best.alpha
```

```
[1] 0
```

```
best.lambda
```

```
[1] 24.3
```

```
best.lambda.1se
```

```
[1] 4451
```

- Now that have identified the preferred model, we retrain the model and simply use `predict` to predict the same model on a new data set.

```
elnet.mod <- glmnet(x[train,], y[train],alpha=best.alpha)

elnet.pred.lambdabest <- predict(elnet.mod,
                    s=best.lambda,newx=x[-train,])

elnet.pred.lambda1se <- predict(elnet.mod,
                    s=best.lambda.1se,newx=x[-train,])

rmse.elnet.lambdabest<-sqrt(mean((y[-train] -
                              elnet.pred.lambdabest)^2))
rmse.elnet.lambda1se<-sqrt(mean((y[-train] -
                              elnet.pred.lambda1se)^2))
```

```r
RMSE[7:8,1]<-c(rmse.elnet.lambda1se,
               rmse.elnet.lambdabest)
RMSE
```

```
                      [,1]
rmse.ridge.lambdabig   405
rmse.ridge.lambda4     296
rmse.ridge.lambda0     300
rmse.ridge.lambdabest  292
rmse.lasso.lambda1se   334
rmse.lasso.lambdabest  297
rmse.elnet.lambda1se   329
rmse.elnet.lambdabest  295
```

- We could also use the **caret** package to do cross-validation for both α and λ
- The package has the `train()` meta engine (aggregator) that allows us to apply almost any direct engine with `method()`

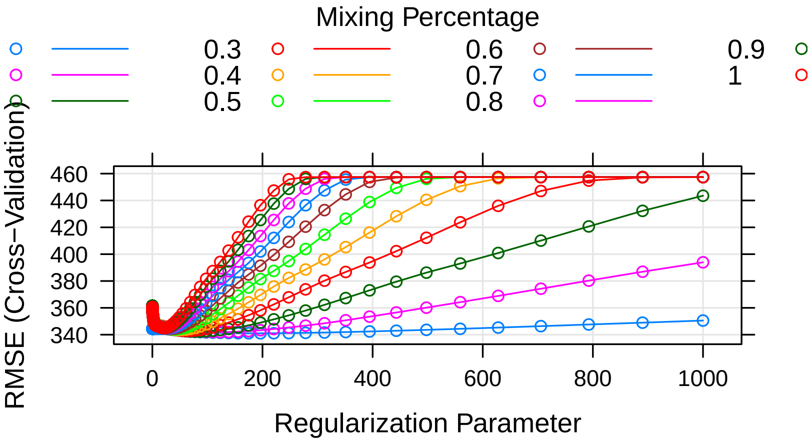
```r
cv_10 = trainControl(method = "cv", number = 10)

grid =  expand.grid(alpha = seq(0,1,by=0.1),
                    lambda = 10^seq(3,-2,length=100))
elnet = train(
  Salary ~ .,
  data = Hitters[train,],
  method = "glmnet",
  trControl = cv_10,
  preProcess = c("center", "scale"),
  tuneGrid = grid)
```

```r
elnet$bestTune
```

	alpha	lambda
86	0	196

```r
alpha.best <- unlist(unname(elnet$bestTune[1]))
lambda.best <- unlist(unname(elnet$bestTune[2]))
```

```r
plot(elnet, xvar = "lambda")
```



- Final model with cross-validated parameters

```
elnet.final <- glmnet(x[train,],y[train],alpha=alpha.best)

elnet.final.lambdabest <- predict(elnet.final,
                        s=lambda.best,newx=x[-train,])

elnet.lambdabest.caret <- sqrt(mean((y[-train]-
                            elnet.final.lambdabest)^2 ))
elnet.lambdabest.caret
```

```
[1] 293
```

```r
predict(elnet.final,s=lambda.best,type = "coefficients")[1
```

```
 (Intercept)        AtBat         Hits        HmRun         Runs
     7.62142     -0.05566      1.02454      1.29116      1.24767
       Walks        Years       CAtBat        CHits       CHmRun
     2.31336     -0.31149      0.00929      0.06329      0.43651
        CRBI       CWalks      LeagueN    DivisionW      PutOuts
     0.11970      0.04249     17.35836   -135.45306      0.21112
      Errors   NewLeagueN
    -1.94017      5.37944
```