Model Selection in practice

Timothy Daley

November 30, 2016

Dataset: Wage

- For this example I'm going to be looking at the Wage dataset in the ISLR package.

```r
library(ISLR)
library(pander)
data(Wage)
library(glmnet)
library(MASS)
```

Wage

```r
names(Wage)
```

```
##  [1] "year"       "age"        "sex"        "maritl"
##  [6] "education"  "region"     "jobclass"   "health"
## [11] "logwage"    "wage"
```

```r
levels(Wage$education)
```

```
## [1] "1. < HS Grad"       "2. HS Grad"          "3. Some C
## [4] "4. College Grad"    "5. Advanced Degree"
```

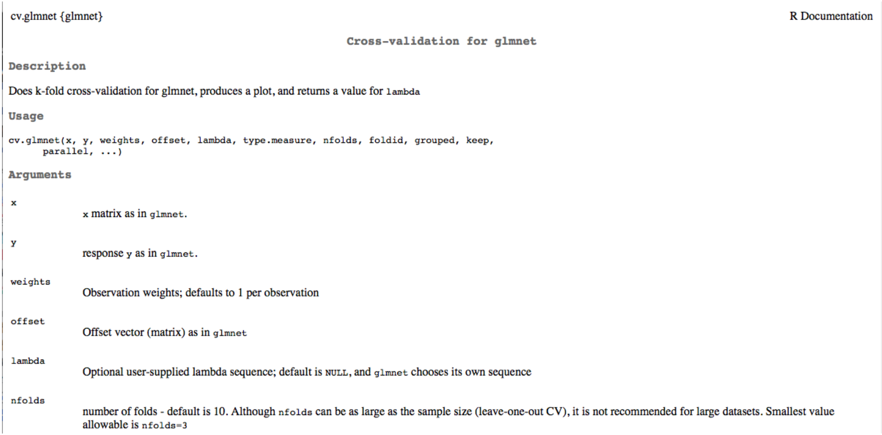
Applying the lasso to predict wage: prepping the data

```
xfactors = model.matrix(wage ~ sex + maritl + race +
                          education + region + jobclass +
                          health + health_ins,
                        data  = Wage)[,-1]
x = as.matrix(data.frame(year = Wage$year, age = Wage$age,
                         xfactors))
```

Using Cross-Validation to choose λ

- We use the function cv.glmnet

```
?cv.glmnet
```



cv.glmnet {glmnet} R Documentation

Cross-validation for glmnet

Description

Does k-fold cross-validation for glmnet, produces a plot, and returns a value for `lambda`

Usage

```
cv.glmnet(x, y, weights, offset, lambda, type.measure, nfolds, foldid, grouped, keep,
      parallel, ...)
```

Arguments

`x`	`x` matrix as in `glmnet`.
`y`	response `y` as in `glmnet`.
`weights`	Observation weights; defaults to 1 per observation
`offset`	Offset vector (matrix) as in `glmnet`
`lambda`	Optional user-supplied lambda sequence; default is `NULL`, and `glmnet` chooses its own sequence
`nfolds`	number of folds - default is 10. Although `nfolds` can be as large as the sample size (leave-one-out CV), it is not recommended for large datasets. Smallest value allowable is `nfolds=3`

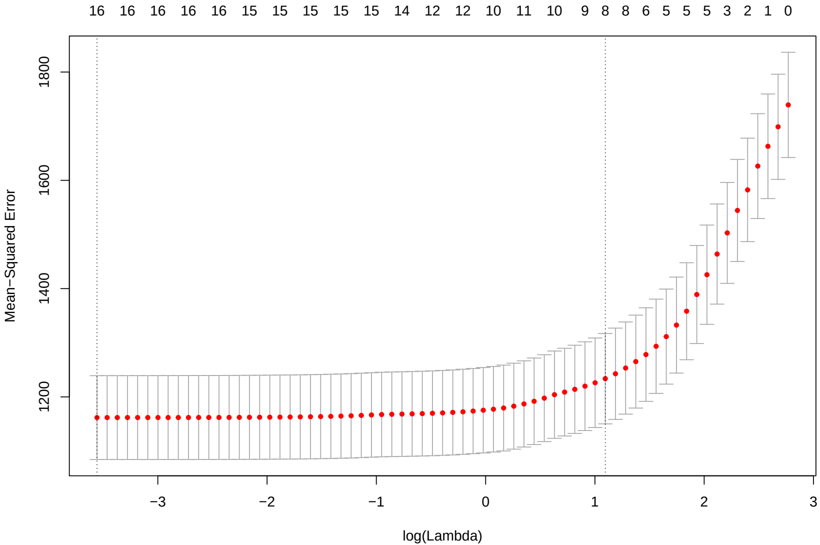
Using Cross-Validation to choose λ

```r
# lasso is alpha = 1
wage.cvfit = cv.glmnet(x, Wage$wage, alpha = 1)
wage.cvfit$lambda.min
```

```
## [1] 0.0285344
```

Using Cross-Validation to choose λ

```r
plot(wage.cvfit)
```



Cross-Validated model

```r
head(coef(wage.cvfit, s = "lambda.min"), 13)
```

```
## 13 x 1 sparse Matrix of class "dgCMatrix"
##                                         1
## (Intercept)                 -2392.1122077
## year                            1.2260075
## age                             0.2710652
## sex2..Female                    .
## maritl2..Married               17.0423512
## maritl3..Widowed                1.5232654
## maritl4..Divorced               3.7478013
## maritl5..Separated             11.0884884
## race2..Black                   -5.0042586
## race3..Asian                   -2.6717244
## race4..Other                   -5.9340531
## education2..HS.Grad             7.1210700
## education3..Some.College       17.6820562
```

Cross-Validated model

```
tail(coef(wage.cvfit, s = "lambda.min"), 13)
```

```
## 13 x 1 sparse Matrix of class "dgCMatrix"
##                                       1
## education4..College.Grad     30.592752
## education5..Advanced.Degree  53.288490
## region2..Middle.Atlantic      .
## region3..East.North.Central   .
## region4..West.North.Central   .
## region5..South.Atlantic       .
## region6..East.South.Central   .
## region7..West.South.Central   .
## region8..Mountain             .
## region9..Pacific              .
## jobclass2..Information        3.556992
## health2....Very.Good          6.484683
## health_ins2..No             -17.533736
```

Why is sex not included?

```r
table(Wage$sex)
```

```
##
##    1. Male 2. Female
##       3000         0
```

Oh. There's no females in the data.

Predicting MLB salary

The hitters data set in ISLR has data from the 1986 and 1987 seasons. Let's see what individual statistics most contribute to salary

```
data(Hitters)
names(Hitters)
```

```
##  [1] "AtBat"     "Hits"      "HmRun"     "Runs"      "RB
##  [6] "Walks"     "Years"     "CAtBat"    "CHits"     "CH
## [11] "CRuns"     "CRBI"      "CWalks"    "League"    "Di
## [16] "PutOuts"   "Assists"   "Errors"    "Salary"    "Ne
```

```
Hitters = Hitters[which(!is.na(Hitters$Salary)), ]
```

Using the lasso to predict MLB salary: data prep

```
xfactors = model.matrix(Salary ~ League + Division +
                            interaction(League, Division),
                          data = Hitters)[,-1]
head(xfactors, 1)
```

```
##              LeagueN DivisionW interaction(League, Divis
## -Alan Ashby        1         1
##              interaction(League, Division)A.W
## -Alan Ashby                                 0
##              interaction(League, Division)N.W
## -Alan Ashby                                 1
```

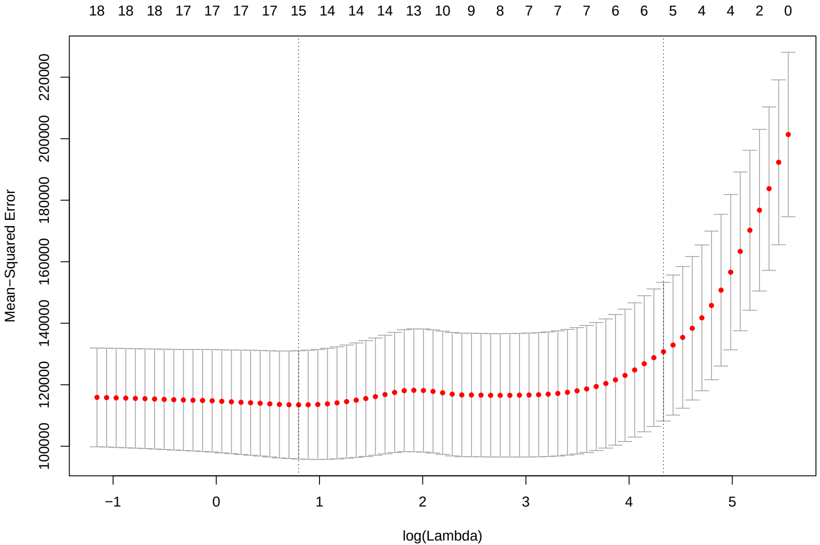
Using the lasso to predict MLB salary: fitting the lasso

```
x = as.matrix(data.frame(Hitters[ ,-c(14, 15, 19, 20)],
                                 xfactors))
MLBsalary.lasso.cvfit = cv.glmnet(x, Hitters$Salary,
                                    alpha = 1)
MLBsalary.lasso.cvfit$lambda.min
```

```
## [1] 2.220313
```

Using the lasso to predict MLB salary: optimal λ for lasso

```
plot(MLBsalary.lasso.cvfit)
```



Using the lasso to predict MLB salary: lasso coefficients

```r
head(coef(MLBsalary.lasso.cvfit), 12)
```

```
## 12 x 1 sparse Matrix of class "dgCMatrix"
##                        1
## (Intercept) 144.3797046
## AtBat          .
## Hits           1.3638038
## HmRun          .
## Runs           .
## RBI            .
## Walks          1.4973110
## Years          .
## CAtBat         .
## CHits          .
## CHmRun         .
## CRuns          0.1527517
```

Using the lasso to predict MLB salary: lasso coefficients

```r
tail(coef(MLBsalary.lasso.cvfit), 10)
```

```
## 10 x 1 sparse Matrix of class "dgCMatrix"
##                                           1
## CRBI                             0.32833941
## CWalks                           .
## PutOuts                          0.06625755
## Assists                          .
## Errors                           .
## LeagueN                          .
## DivisionW                        .
## interaction.League..Division.N.E .
## interaction.League..Division.A.W .
## interaction.League..Division.N.W .
```

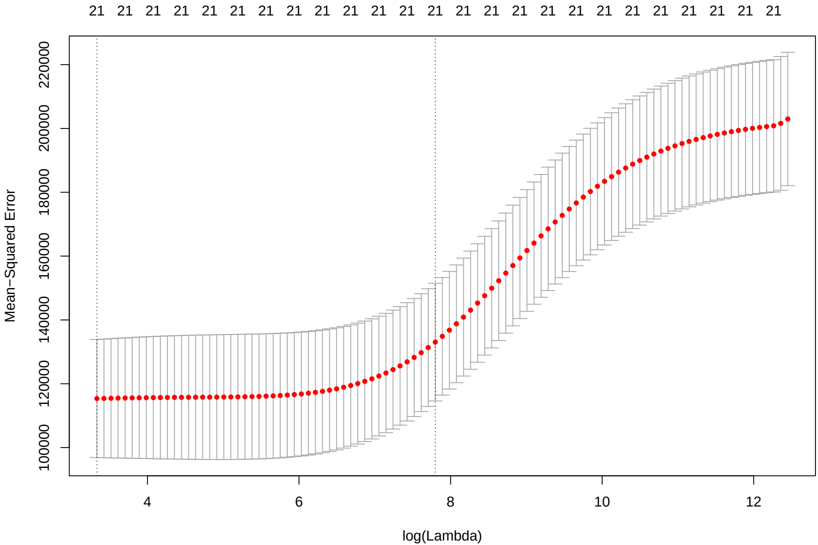
Using ridge regression to predict MLB salary: fitting

```
MLBsalary.rr.cvfit = cv.glmnet(x, Hitters$Salary,
                               alpha = 0)
MLBsalary.rr.cvfit$lambda.min
```

```
## [1] 28.01718
```

Using ridge regression to predict MLB salary: optimal λ

```
plot(MLBsalary.rr.cvfit)
```



Using ridge regression to predict MLB salary: coefficients

```r
head(coef(MLBsalary.rr.cvfit), 12)
```

```
## 12 x 1 sparse Matrix of class "dgCMatrix"
##                        1
## (Intercept) 2.051021e+02
## AtBat       9.371095e-02
## Hits        3.896188e-01
## HmRun       1.222685e+00
## Runs        6.227768e-01
## RBI         6.184842e-01
## Walks       8.085052e-01
## Years       2.539892e+00
## CAtBat      7.891144e-03
## CHits       3.052107e-02
## CHmRun      2.268930e-01
## CRuns       6.116106e-02
```

Using ridge regression to predict MLB salary: coefficients

```r
tail(coef(MLBsalary.rr.cvfit), 10)
```

```
## 10 x 1 sparse Matrix of class "dgCMatrix"
##                                         1
## CRBI                          0.063385593
## CWalks                        0.060508479
## PutOuts                       0.056023095
## Assists                       0.007589146
## Errors                       -0.173441672
## LeagueN                       2.334635145
## DivisionW                   -20.678801778
## interaction.League..Division.N.E   9.804002150
## interaction.League..Division.A.W -20.311680354
## interaction.League..Division.N.W  -6.393999590
```

Making sense of ridge regression coefficients

```
sapply(c("AtBat", "Hits", "HmRun", "Runs", "RBI",
         "Walks", "Years", "CAtBat", "CHits", "CHmRun",
         "CRuns", "CRBI", "CWalks", "PutOuts", "Assists",
         "Errors"),
       function(x) mean(Hitters[ ,x])*
         coef(MLBsalary.rr.cvfit)[x,])
```

```
##      AtBat       Hits      HmRun       Runs        RBI
## 37.8257316 42.0121640 14.2073188 34.0940699 31.8437034 3
##      Years     CAtBat      CHits     CHmRun      CRuns
## 18.5711494 20.9710590 22.0419025 15.7099711 22.0926294 2
##     CWalks    PutOuts    Assists     Errors
## 15.7483095 16.2865315  0.9012905 -1.4904113
```

Standard regression

First let's look at the full regression

```
x.full = data.frame(x, Salary = Hitters$Salary)
MLBsalary.lm = lm(Salary ~ ., data = x.full)
```

Standard regression

Table 1: Fitting linear model: Hitters$Salary ~ .

	Estimate	Std. Error	t value	Pr($>\lvert t\rvert$)
AtBat	-1.998	0.6353	-3.145	0.001866
Hits	7.551	2.381	3.172	0.001709
HmRun	4.412	6.208	0.7108	0.4779
Runs	-2.382	2.984	-0.7981	0.4256
RBI	-1.054	2.604	-0.4047	0.686
Walks	6.211	1.828	3.398	0.0007929
Years	-3.453	12.55	-0.2752	0.7834
CAtBat	-0.1689	0.1355	-1.246	0.2139
CHits	0.1093	0.6719	0.1626	0.8709
CHmRun	-0.2066	1.615	-0.128	0.8983
CRuns	1.478	0.747	1.978	0.04903
CRBI	0.8245	0.6917	1.192	0.2344
CWalks	-0.8133	0.3281	-2.479	0.01387
PutOuts	0.2819	0.07746	3.64	0.0003335
Assists	0.3721	0.2213	1.682	0.09395

Forward stepwise regression

```
MLBsalary.null = lm(Salary ~ 1, data = x.full)
MLBsalary.forward =
  stepAIC(MLBsalary.null, scope =
            list(upper = MLBsalary.lm,
                 lower = MLBsalary.null),
          direction = "forward")
```

```
## Start:  AIC=3215.77
## Salary ~ 1
##
##                                    Df Sum of Sq      RSS
## + CRBI                              1  17139434 36179679
## + CRuns                             1  16881162 36437951
## + CHits                             1  16065140 37253973
## + CAtBat                            1  14759710 38559403
## + CHmRun                            1  14692193 38626920
## + CWalks                            1  12792622 40526491
## + RBI                               1  10771083 42548030
```

Forward stepwise regression

```
pander(MLBsalary.forward, style = "simple")
```

Table 2: Fitting linear model: Salary ~ CRBI + Hits + PutOuts + DivisionW + AtBat + Walks + CWalks + CRuns + CAtBat + Assists

	Estimate	Std. Error	t value	Pr(>|t|)
CRBI	0.7743	0.2096	3.694	0.0002706
Hits	6.918	1.647	4.201	3.686e-05
PutOuts	0.2974	0.07444	3.995	8.504e-05
DivisionW	-112.4	39.21	-2.866	0.004511
AtBat	-2.169	0.5363	-4.044	6.996e-05
Walks	5.773	1.585	3.643	0.0003274
CWalks	-0.8308	0.2636	-3.152	0.001818
CRuns	1.408	0.3904	3.607	0.0003731
CAtBat	-0.1301	0.0555	-2.344	0.01986
Assists	0.2832	0.1577	1.796	0.07367
(Intercept)	162.5	66.91	2.429	0.01583

Backward stepwise regression

```
MLBsalary.backward =
    stepAIC(MLBsalary.lm, scope =
              list(upper = MLBsalary.lm,
                   lower = MLBsalary.null),
            direction = "backward")
```

```
## Start:  AIC=3046.13
## Salary ~ AtBat + Hits + HmRun + Runs + RBI + Walks + Yea
##     CAtBat + CHits + CHmRun + CRuns + CRBI + CWalks + Pu
##     Assists + Errors + LeagueN + DivisionW + interactio
##     interaction.League..Division.A.W + interaction.Leagu
##
##
## Step:  AIC=3046.13
## Salary ~ AtBat + Hits + HmRun + Runs + RBI + Walks + Yea
##     CAtBat + CHits + CHmRun + CRuns + CRBI + CWalks + Pu
##     Assists + Errors + LeagueN + DivisionW + interactio
##     interaction.League..Division.A.W
```

Backward stepwise regression

```r
pander(MLBsalary.backward, style = "simple")
```

Table 3: Fitting linear model: Salary ~ AtBat + Hits + Walks + CAtBat + CRuns + CRBI + CWalks + PutOuts + Assists + DivisionW

	Estimate	Std. Error	t value	Pr(>|t|)
AtBat	-2.169	0.5363	-4.044	6.996e-05
Hits	6.918	1.647	4.201	3.686e-05
Walks	5.773	1.585	3.643	0.0003274
CAtBat	-0.1301	0.0555	-2.344	0.01986
CRuns	1.408	0.3904	3.607	0.0003731
CRBI	0.7743	0.2096	3.694	0.0002706
CWalks	-0.8308	0.2636	-3.152	0.001818
PutOuts	0.2974	0.07444	3.995	8.504e-05
Assists	0.2832	0.1577	1.796	0.07367
DivisionW	-112.4	39.21	-2.866	0.004511
(Intercept)	162.5	66.91	2.429	0.01583

Looks like the same model

```r
length(names(MLBsalary.backward$coefficients))
```

```
## [1] 11
```

```r
length(intersect(names(MLBsalary.backward$coefficients),
          names(MLBsalary.forward$coefficients)))
```

```
## [1] 11
```

Comparing forward and backward regression via LOOCV

```
forward_error = 0
for(i in 1:dim(x.full)[1]){
  full_model = lm(Salary ~ ., data = x.full[-i, ])
  null_model = lm(Salary ~ 1, data = x.full[-i, ])
  forward_model =
    step(null_model, scope = list(upper = full_model,
                                  lower = null_model),
         direction = "forward", trace = 0)
  forward_error = forward_error +
    (x.full$Salary[i] -
     predict(forward_model, newdata = data.frame(x.full[i,]
             interval = "none"))^2
}
```

Comparing forward and backward regression via LOOCV

```
backward_error = 0
for(i in 1:dim(x.full)[1]){
  full_model = lm(Salary ~ ., data = x.full[-i, ])
  null_model = lm(Salary ~ 1, data = x.full[-i, ])
  forward_model =
    step(full_model, scope = list(upper = full_model,
                                  lower = null_model),
         direction = "backward", trace = 0)
  backward_error = backward_error +
    (x.full$Salary[i] -
     predict(forward_model, newdata = data.frame(x.full[i,]
             interval = "none"))^2
}
```

Comparing forward and backward regression via LOOCV

```
forward_error
```

```
## -Alan Ashby
##    30060779
```

```
backward_error
```

```
## -Alan Ashby
##    28922347
```