HW -5 574

Shrey

2022-11-04

1

The paper discusses LASSO as choice for variable selection and regularization. Lasso has properties of both ridge and variable selection. Its equivalent to solving OLS problem subject to $\Sigma|\beta| < t$

For p=2 the sign of lasso coefficients is the same as that of OLS but for p>3 the signs can change.

They further propose that lasso can be extended to generalized linear model like logistic regression and Tree based models.

2

(a)

When t = 0 all the coefficients except the intercept are set to zero

(b)

When t>t0 then we are removing the constraint from the LASSO problem and the objective function will be solved for ordinary least square method hence for t>t0 coefficients from lasso will be same as obtained in ordinary least squares

(c)

We have to prove that $min_\beta \Sigma_{i=1}^n (y_i - x_i^T \beta)^2 + \Sigma_{j=1}^p |\beta_j|$

and

$\Sigma_{j=1}^p (\beta_j^{ols} - \beta)^2 + \Sigma_{j=1}^p |\beta_j|$

will produce the same solution for beta

$\Sigma_{j=1}^p |\beta_j|$ is common in both

Writing both the equations in matrix form

$(Y - X\beta)^T (Y - X\beta)$ and $(\beta^{ols} - \beta)^T (\beta^{ols} - \beta)$

when X is orthogonal $\beta^{ols} = X^T Y$

second equation becomes $(X^T Y - \beta)^T (X^T Y - \beta)$

In the first equation pre- multiplying X^T gives us $(X^T Y - X^T X\beta)^T (X^T Y - X^T X\beta)$ $(X^T Y - \beta)^T (X^T Y - \beta)$

which is equal to the first equation

(d)

```
beta_ols = c(1.1,-0.8,0.3,-0.1)
l = c(1,0.4)

for (lambda in l)
{
  beta_ridge = beta_ols/(1+lambda)
  print(beta_ridge)
  beta_lasso = beta_ols
  for (i in seq(1,4))
  {
    if (beta_ols[i] > lambda/2)
    {
      beta_lasso[i] =beta_ols[i]-lambda/2
    }
    if (abs(beta_ols[i]) <= lambda/2)
    {
      beta_lasso[i] =0
    }
    if (beta_ols[i]< -1*lambda/2)
    {
      beta_lasso = beta_ols + lambda/2
    }
  }

  print(beta_lasso)
}
```

```
## [1]  0.55 -0.40  0.15 -0.05
## [1]  1.6 -0.3  0.0  0.0
## [1]  0.78571429 -0.57142857  0.21428571 -0.07142857
## [1]  1.3 -0.6  0.1  0.0
```

When lambda = 1 beta_ridge = 0.55 -0.40 0.15 -0.05 beta_lasso = 1.6 -0.3 0.0 0.0

When lambda = 0.4 beta_ridge = 0.78571429 -0.57142857 0.21428571 -0.07142857 beta_lasso = 1.3 -0.6 0.1 0.0

3

(a)

Adaptive Lasso can be transformed into an equivalent Lasso problem using the following algorithm:-

1. Divide column of centered X matrix with Wj, where $W_j = 1/\beta_j^{ols}$ 2. Solve the lasso problem with transformed X matrix

$\hat{\beta} = argmin_{\beta}||y - \Sigma x_j \beta_j||^2 + \lambda \Sigma_{i=1}^{p} |\beta_j|$

3. $\beta_j^{l\hat{asso}} = \hat{\beta}_j / W_j$

(b)

We can use the LARS package to fit adaptive lasso using the following way

1. Fit a linear model and get the coefficients and compute the weights, Wj = 1/bj
2. Divide the columns of centered X with W to and use cv.lars function from lasr package on the transformed X to get the parameter s.
3. Use the predict.lars() function with type = "coeff" to get the coefficients
4. Finally, divide the above coefficients with W to get the coefficents for adpative lasso

4

(a)

```
data = read.csv("/Users/shreyarora/Documents/Data sets/prostate_cancer.csv")
data = data[,2:11]
train_data = data[which(data["train"]==TRUE),][,1:9]
test_data = data[-which(data["train"]==TRUE),][,1:9]

linear_model = lm(lpsa~.,data = train_data)
#r_squared
print(summary(linear_model)$r.squared)
```

```
## [1] 0.6943712
```

```
# p_values
p_values = summary(linear_model)[4]$coefficients[,4]
print(p_values)
```

```
##  (Intercept)       lcavol      lweight          age         lbph          svi
## 7.833423e-01 1.469415e-06 7.917895e-03 1.680626e-01 4.430784e-02 1.650539e-02
##          lcp      gleason        pgg45
## 6.697085e-02 8.838923e-01 8.754628e-02
```

```
# significant_predictors
p_values[p_values<=0.05]
```

```
##       lcavol      lweight         lbph          svi
## 1.469415e-06 7.917895e-03 4.430784e-02 1.650539e-02
```

```
# train and test_errors
MSE_train = sum((linear_model$fitted.values - train_data[,9])^2)/67

MSE_test = sum((predict(linear_model, newdata = test_data)-test_data[,9])^2)/30

print(MSE_train)
```

```
## [1] 0.4391998
```

```r
print(MSE_test)
```

```
## [1] 0.521274
```

R-squared = 0.6943712 p values = (Intercept) lcavol lweight age lbph 7.833423e-01 1.469415e-06 7.917895e-03 1.680626e-01 4.430784e-02 svi lcp gleason pgg45 1.650539e-02 6.697085e-02 8.838923e-01 8.754628e-02

significant coefficients = lcavol lweight lbph svi

Train error = 0.4391998 Test error = 0.521274

(b)

```r
library(leaps)
forward_selection = regsubsets(train_data[,-9],train_data[,9],method = "forward")
summary(forward_selection)
```

```
## Subset selection object
## 8 Variables  (and intercept)
##         Forced in Forced out
## lcavol      FALSE      FALSE
## lweight     FALSE      FALSE
## age         FALSE      FALSE
## lbph        FALSE      FALSE
## svi         FALSE      FALSE
## lcp         FALSE      FALSE
## gleason     FALSE      FALSE
## pgg45       FALSE      FALSE
## 1 subsets of each size up to 8
## Selection Algorithm: forward
##          lcavol lweight age lbph svi lcp gleason pgg45
## 1  ( 1 ) "*"    " "     " " " "  " " " " " "     " "
## 2  ( 1 ) "*"    "*"     " " " "  " " " " " "     " "
## 3  ( 1 ) "*"    "*"     " " " "  "*" " " " "     " "
## 4  ( 1 ) "*"    "*"     " " "*"  "*" " " " "     " "
## 5  ( 1 ) "*"    "*"     " " "*"  "*" " " " "     "*"
## 6  ( 1 ) "*"    "*"     " " "*"  "*" "*" " "     "*"
## 7  ( 1 ) "*"    "*"     "*" "*"  "*" "*" " "     "*"
## 8  ( 1 ) "*"    "*"     "*" "*"  "*" "*" "*"     "*"
```

```r
columns = c(1,2,5,4,8,6,3,7)

train_errors = c()
BIC = c()

for (i in seq(1,8))
{
  x_train = train_data[,columns[1:i]]
  lpsa = train_data[,9]
  df = data.frame(x_train,lpsa)
  m = lm(lpsa~.,data = df)
```

```r
  print(m$coefficients)
  m.train_error = sum((m$fitted.values - lpsa)^2)/67
  m.bic = 67*log(m.train_error)+log(67)*length(m$coefficients)
  train_errors = append(train_errors,m.train_error)
  BIC =append(BIC,m.bic)
  print(m.train_error)
  print(m.bic)
}
```

```
## (Intercept)     x_train
##   1.5163048   0.7126351
## [1] 0.6646057
## [1] -18.96422
## (Intercept)      lcavol     lweight
##  -1.0494396   0.6276074   0.7383751
## [1] 0.5536096
## [1] -27.00272
## (Intercept)      lcavol     lweight         svi
##  -1.0227780   0.5199861   0.7367954   0.5379032
## [1] 0.5210112
## [1] -26.86414
## (Intercept)      lcavol     lweight         svi        lbph
##  -0.3259212   0.5055209   0.5388292   0.6718487   0.1400111
## [1] 0.489776
## [1] -26.80161
##  (Intercept)       lcavol      lweight          svi         lbph        pgg45
## -0.465877591  0.472278483  0.563935476  0.578163005  0.137116261  0.004330753
## [1] 0.4786485
## [1] -24.1367
##  (Intercept)       lcavol      lweight          svi         lbph        pgg45
## -0.728972257  0.549778034  0.563105747  0.756354835  0.125978836  0.007541236
##          lcp
## -0.190824719
## [1] 0.4558176
## [1] -23.20654
##  (Intercept)       lcavol      lweight          svi         lbph        pgg45
##  0.259061747  0.573930391  0.619208833  0.741781258  0.144426474  0.008944996
##          lcp          age
## -0.205416986 -0.019479879
## [1] 0.4393627
## [1] -21.46527
##  (Intercept)       lcavol      lweight          svi         lbph        pgg45
##  0.429170133  0.576543185  0.614020004  0.737208645  0.144848082  0.009465162
##          lcp          age      gleason
## -0.206324227 -0.019001022 -0.029502884
## [1] 0.4391998
## [1] -17.28543
```

```r
## min BIC
print(which.min(BIC))
```

```
## [1] 2
```

```r
##Final Model with min BIC
x_train = train_data[,columns[1:which.min(BIC)]]
lpsa = train_data[,9]
df = data.frame(x_train,lpsa)
m = lm(lpsa~.,data = df)
print(m)
```

```
##
## Call:
## lm(formula = lpsa ~ ., data = df)
##
## Coefficients:
## (Intercept)       lcavol      lweight
##     -1.0494       0.6276       0.7384
```

```r
##test data
x_test =  test_data[,columns[1:which.min(BIC)]]
lpsa_test = test_data[,9]
test_error = sum((predict(m,newdata = x_test)-lpsa_test)^2)/30
print(test_error)
```

```
## [1] 0.4924823
```

selected model

lpsa = -1.0494 + 0.6276xlcavol + 0.7384xlweight

test error = 0.49248

(c)

```r
train_errors = c()
AIC = c()

for (i in seq(1,8))
{
  x_train = train_data[,columns[1:i]]
  lpsa = train_data[,9]
  df = data.frame(x_train,lpsa)
  m = lm(lpsa~.,data = df)
  m.train_error = sum((m$fitted.values - lpsa)^2)/67
  m.aic = 67*log(m.train_error)+2*length(m$coefficients)
  train_errors = append(train_errors,m.train_error)
  AIC =append(AIC,m.aic)
  print(m.train_error)
  print(m.aic)
}
```

```
## [1] 0.6646057
## [1] -23.37361
## [1] 0.5536096
```

```
## [1] -33.6168
## [1] 0.5210112
## [1] -35.68291
## [1] 0.489776
## [1] -37.82507
## [1] 0.4786485
## [1] -37.36485
## [1] 0.4558176
## [1] -38.63939
## [1] 0.4393627
## [1] -39.10281
## [1] 0.4391998
## [1] -37.12766
```

```
## min AIC
print(which.min(AIC))
```

```
## [1] 7
```

```
##Final Model with min BIC
x_train = train_data[,columns[1:which.min(AIC)]]
lpsa = train_data[,9]
df = data.frame(x_train,lpsa)
model_aic = lm(lpsa~.,data = df)
print(model_aic)
```

```
##
## Call:
## lm(formula = lpsa ~ ., data = df)
##
## Coefficients:
## (Intercept)       lcavol      lweight          svi         lbph        pgg45
##    0.259062     0.573930     0.619209     0.741781     0.144426     0.008945
##         lcp          age
##   -0.205417    -0.019480
```

```
##test data
x_test =  test_data[,columns[1:which.min(AIC)]]
lpsa_test = test_data[,9]
test_error = sum((predict(model_aic,newdata = x_test)-lpsa_test)^2)/30
print(test_error)
```

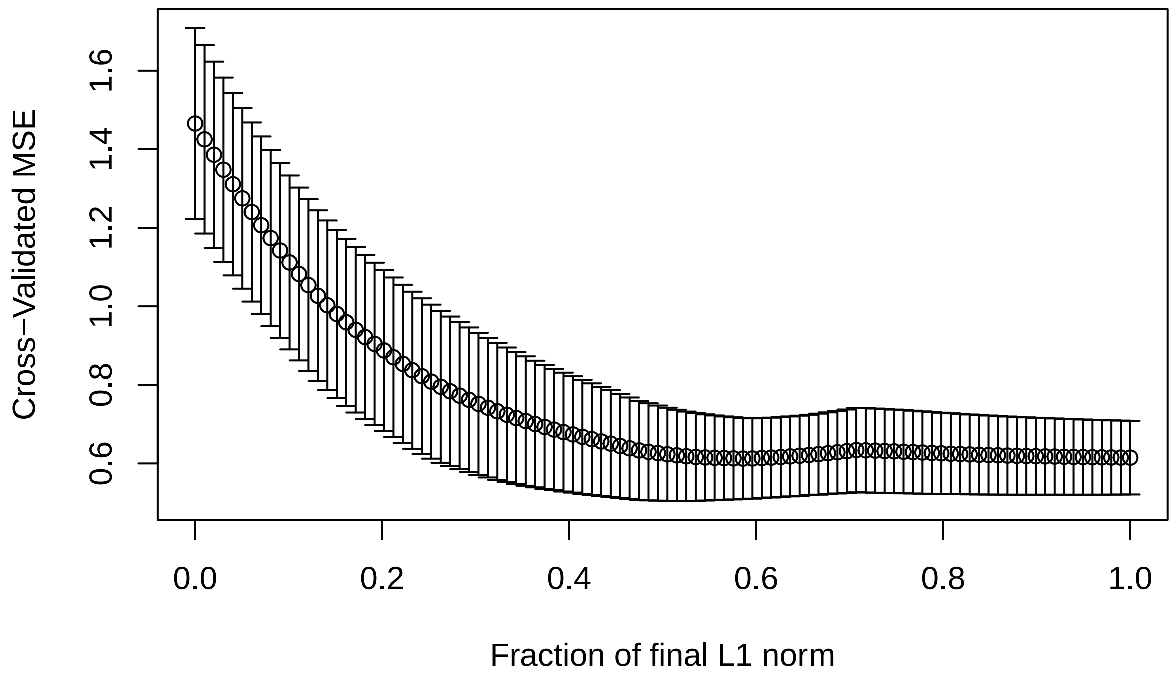
```
## [1] 0.5165135
```

Test error = 0.5165135

5

(a)

```
## Loaded lars 1.3
```




```
s_min_cv = lasso.s[which.min(lasso.cv$cv)]
lasso_model = lars(x,y,type= "lasso")
lasso_coeff =  predict(lasso_model, s=s_min_cv, type="coef",mode="frac")
print(lasso_coeff)
```

```
## $s
## [1] 0.5858586
##
## $fraction
## [1] 0.5858586
##
## $mode
## [1] "fraction"
##
## $coefficients
## [1]  0.470228429  0.532009912 -0.002923121  0.107551542  0.489818064
## [6]  0.000000000  0.000000000  0.003460684
```

```
x_test = matrix(c(test_data[,1],test_data[,2],test_data[,3],test_data[,4],test_data[,5],test_data[,6],te
y_test = matrix(c(test_data[,9]),30,1)

y_predicted = predict.lars(lasso_model,newx =x_test, s=s_min_cv, type = "fit", mode="fraction")

test_error = sum((y_predicted$fit-y_test)^2)/30
print(test_error)
```

```
## [1] 0.4567557
```

test error = 0.4806732

(b)

```r
bound = lasso.cv$cv[which.min(lasso.cv$cv)] + lasso.cv$cv.error[which.min(lasso.cv$cv)]

s_one_std = lasso.s[min(which(lasso.cv$cv < bound))]
lasso_coeff_one_std =  predict(lasso_model, s=s_one_std, type="coef",mode="frac")
print(lasso_coeff_one_std)
```

```
## $s
## [1] 0.3535354
##
## $fraction
## [1] 0.3535354
##
## $mode
## [1] "fraction"
##
## $coefficients
## [1] 0.4438369 0.3557264 0.0000000 0.0000000 0.1859968 0.0000000 0.0000000
## [8] 0.0000000
```

```r
y_predicted_one_std = predict.lars(lasso_model,newx =x_test, s=s_one_std, type = "fit", mode="fraction")
test_error2 = sum((y_predicted_one_std$fit-y_test)^2)/30
print(test_error2)
```

```
## [1] 0.4940786
```

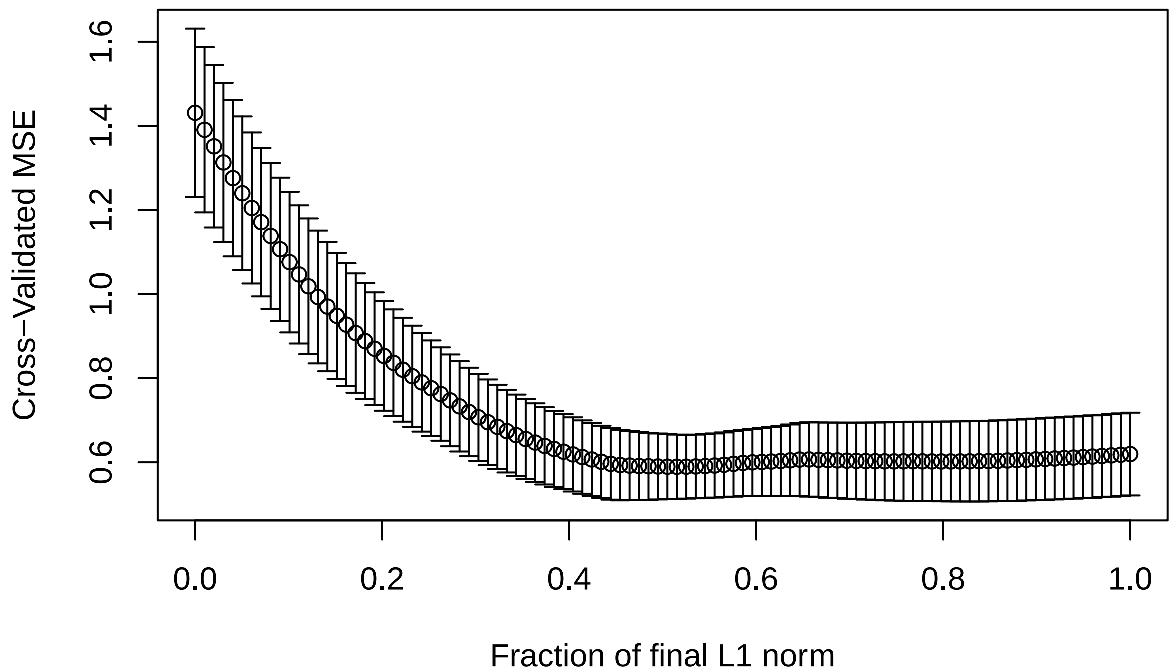
test error = 0.4708345

6

(a)

```r
gamma = 1
bols= lm(lpsa~.,data = train_data)$coefficients[2:9]
w= bols
## adjusting x for adaptive lasso
x_adaptive = x
x_test_adaptive = x_test
for (j in seq(1,8))
{
  w[j] = 1/abs(bols[j])
  x_centered = x[,j] - mean(x[,j])
  x_adaptive[,j] = x_centered/(abs(w[j])^gamma)
}

adaptivelasso.cv = cv.lars(x_adaptive,y,K=5,index= lasso.s,mode="fraction")
```




```r
s_min_cv_adpative = lasso.s[which.min(adaptivelasso.cv$cv)]
print(s_min_cv_adpative)
```

```
## [1] 0.5151515
```

```r
adpative_lasso_model = lars(x_adaptive,y,type= "lasso")
coeffs =  predict(adpative_lasso_model, s=s_min_cv_adpative, type="coef",mode="frac")$coefficients
coeffs_alasso = coeffs
#Divide coeffs by w to get adaptive lasso coeffs
for (i in seq(1,8))
{
  coeffs_alasso[i] = coeffs[i]/abs(w[i])
}
print(coeffs_alasso)
```

```
## [1] 0.463249442 0.487790277 0.000000000 0.075864827 0.419446328 0.000000000
## [7] 0.000000000 0.002361025
```

```r
y_predicted_alasso = x_test%*%coeffs_alasso
test_error_alasso = sum((y_predicted_alasso - y_test)^2)/30
print(test_error_alasso)
```

```
## [1] 0.4531596
```

S = 0.8686869 coefficients = 0.543134167 0.595205705 -0.014889037 0.134621147 0.667500063 -0.143386850 0.000000000 0.007342938 test error = 0.5308645

(b)

```
bound = adaptivelasso.cv$cv[which.min(adaptivelasso.cv$cv)] + adaptivelasso.cv$cv.error[which.min(adapti

s_one_std = lasso.s[min(which(adaptivelasso.cv$cv < bound))]
print(s_one_std)
```

```
## [1] 0.3434343
```

```
coeffs=  predict(adpative_lasso_model, s=s_one_std, type="coef",mode="frac")$coefficients
coeffs_alasso = coeffs
#Divide coeffs by w to get adaptive lasso coeffs
for (i in seq(1,8))
{
  coeffs_alasso[i] = coeffs[i]/abs(w[i])
}
print(coeffs_alasso)
```

```
## [1] 0.4397365 0.3352068 0.0000000 0.0000000 0.1670476 0.0000000 0.0000000
## [8] 0.0000000
```

```
y_predicted_alasso = x_test%*%coeffs_alasso
test_error_alasso = sum((y_predicted_alasso - y_test)^2)/30
print(test_error_alasso)
```

```
## [1] 0.930129
```

s = 0.4949495

coefficients =0.461672898 0.474494704 0.000000000 0.065169827 0.391733101 0.000000000 0.000000000 0.002017095

test error = 0.4510459

(c)

Variable selection's best solution had test error of 0.49248 and it retained two variables lcavol and lweight.

Lasso's best solution had test error of 0.4708345 and it retained 5 variables lcavol, lweight, lbph , svi, pgg45

Adaptive lasso produced best error of 0.4510459 and it retained 5 variables lcavol, lweight, lbph , svi, pgg45.

On comparing adaptive lasso with lasso and variable selection we can see variable selection produced most sparse model but it had relatively high error. Adaptive lasso and lasso both produced models with 5 parameters but adaptive lasso had better error than lasso.