11 The lasso

lasso = Least Absolute Selection and Shrinkage Operator
The lasso has been introduced by Robert Tibshirani in 1996 and represents another modern approach in regression similar to ridge estimation. He described it in detail in the text book "The Elements of Statistical Learning" by Hastie, T, Tibshirani, R, Friedman, J (2nd edition) Springer, available online at http://statweb.stanford.edu/ tibs/ElemStatLearn/
The benefits and properties for ridge and lasso estimators are quite different.
In ridge regression we aimed to reduce the variance of the estimators and predictions which is particularly helpful in the presence of multicollinearity.
Lasso is a tool for model (predictor) selection and consequently improvement of interpretability.
Interestingly both ridge and lasso estimators are the solutions of very similar optimization problems
Ridge:

$$\hat{\beta}_R(k) = \underset{\hat{\beta}^*}{\operatorname{argmin}} ||\vec{y} - X\hat{\beta}^*||_2^2 + k||\hat{\beta}^*||_2^2$$

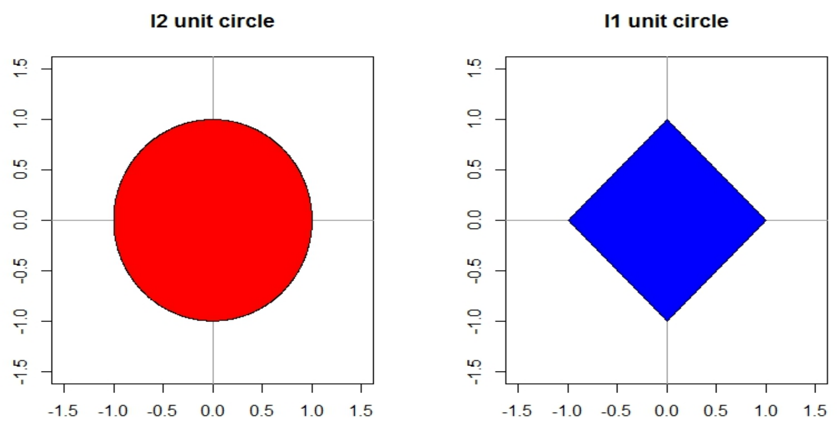
Lasso:

$$\hat{\beta}_{lasso}(\lambda) = \underset{\hat{\beta}^*}{\operatorname{argmin}} ||\vec{y} - X\hat{\beta}^*||_2^2 + \lambda||\hat{\beta}^*||_1$$

The only difference is that the penalty term for lasso uses the l_1 norm and ridge uses the squared l_2 norm.

$$||\vec{x}||_1 = \sum_{i=1}^{p} |x_i|, \qquad ||\vec{x}||_2 = \sqrt{\sum_{i=1}^{p} x_i^2}$$

The circles of radius $r := \{\vec{x} \mid ||\vec{x}|| \leq r\}$ implied by these two norms are shown in the figure



This difference has a stark implication on the solution of the two optimization problems, where the ridge estimator shrinks the slope estimates, but even for large k not necessarily to 0, the lasso will shrink the estimates to 0 for sufficiently large λ, which helps with model selection.

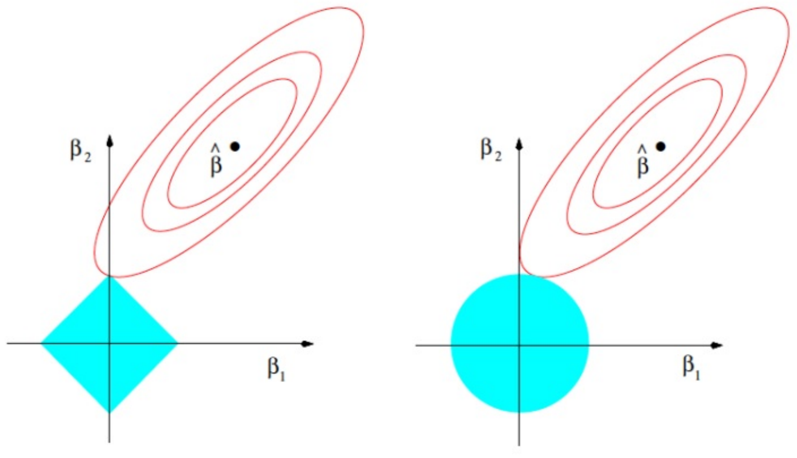
11.1 Variable Selection Problem

In the presence of many predictors (large p) it is often not clear which of them are most relevant in the relationship with the response. This makes the interpretation foggy and and unclear.
The most common solution to this problem is to choose a subset of predictors, which appear to be most important, based on trial and error using p-values to guide whether a variable should be in the model or not. Forward and backward selection procedures are examples for these.
However, this approach is data driven and will potentially result in different models each time a sample is chosen (small changes in the data can result in very different results). The study of the statistical properties of the models chosen by these algorithms is almost impossible.
A more systematic approach is to exhaustively search all possible subsets and then use some sort of criterion such as R^2_{adj}, AIC, or BIC to choose an optimal predictor subset. Unfortunately this becomes infeasible for large p.
Test results in such chosen models become unreliable, because their assumptions are now violated (i.e. we propose a model to describe a population, randomly choose data to draw conclusions about this model and therefore about the population).
A more systematic approach, which will be consistent (small changes will not result in entirely different models), which permits statistical analyses for the resulting model, and would improve the reliability would provide a great advancement in the application of regression methods for model selection.

11.2 The lasso

It is

$$\hat{\beta}_{lasso}(\lambda) = \underset{\hat{\beta}^*}{\text{argmin}} \underbrace{||\vec{y} - X\hat{\beta}^*||_2^2}_{\text{loss}} + \lambda \underbrace{||\hat{\beta}^*||_1}_{\text{penalty}}$$



(From page 71 of ESL)

The ellipses in the figure represent sets of pairs, (β_1, β_2), with equal loss, $||\vec{y} - X\hat{\beta}^*||_2^2$. For a given λ, $\hat{\beta}_{lasso}(\lambda)$ is equal to (β_1, β_2) on the ellipse with loss $||\vec{y} - X\hat{\beta}_{lasso}(\lambda)||_2^2$ with smallest l_1 norm.

More mathematical $\hat{\beta}_{lasso}(\lambda)$ is the solution of

$$\begin{aligned} &min_{\hat{\beta}^*}||\hat{\beta}^*||_1 \\ &\text{subject to } ||\vec{y}-X\hat{\beta}^*||_2 = ||\vec{y}-X\hat{\beta}_{lasso}(\lambda)||_2 \end{aligned}$$

If the ellipse for a given λ is intersected by an axis the respective component of $\hat{\beta}_{lasso}(\lambda)$ is 0, because the ellipses (ellipsoids for higher dimensions) are convex. Since the ellipsoids will increase indefinitely for increasing λ, there exists a sufficiently large λ_0 such that $\hat{\beta}_{lasso}(\lambda_0) = \vec{0}$.
Also $\hat{\beta}_{lasso}(0) = \hat{\beta}$ and the number of non-zero entries in $\hat{\beta}_{lasso}(\lambda)$ is decreasing in λ, because when λ increases so does the ellipsoid, potentially intersecting more axes.
Plotting the paths of lasso coefficient for $\lambda > 0$ visualizes the effect of the increase in the importance of the penalty term on the coefficients and them dropping to 0.

Comment:
There is no closed form expression for calculating $\hat{\beta}_{lasso}(\lambda)$. For each λ the coefficient vector has to be found through iterative processes. This is a numerically challenging problem, which has been greatly improved by LARS (an algorithm introduced by Trevor Hastie and Brad Efron).

Example 11.1.
Data from Stamey, T.A. et al. (1989)Prostate specific antigen in the diagnosis and treatment of adenocarcinoma of the prostate: II. radical prostatectomy treated patients, Journal of Urology 141(5), 10761083. now available in the lasso2 library will be used for this example.
The observations are prostate cancer patients and the following variables

```
lcavol = log(cancer volume)
lweight = log(prostate weight)
age
lbph = log(benign prostatic hyperplasia amount)
svi seminal vesicle invasion
lcp = log(capsular penetration)
gleason = Gleason score
pgg45 = percentage Gleason scores 4 or 5
lpsa = log(prostate specific antigen)
```

R code:

```
library(lasso2) # to get the data
data(Prostate)
Prostate

library(glmnet) # library including functions for finding the lasso

# glmnet requires design matrix and response variable
x<-(apply(Prostate[,-9],2,as.numeric))
y<-(Prostate[,9])

lassofit<-glmnet(x,y)
```

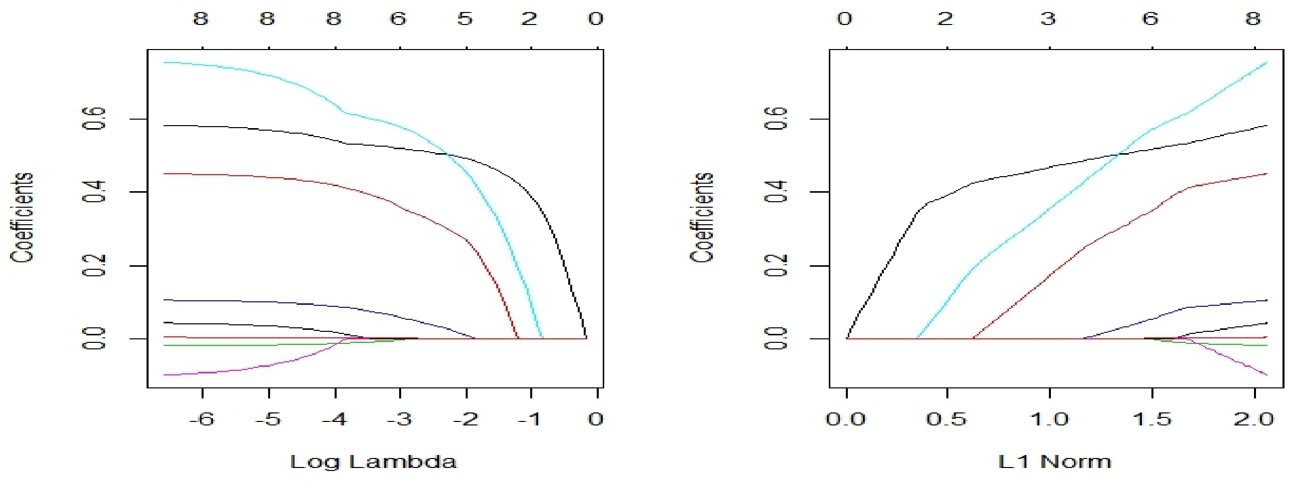
```
print(lassofit) # DF gives the number of non-zero coefficients
par(mfrow = c(1, 2))
plot(lassofit, xvar="lambda")
plot(lassofit, xvar="norm")
par(mfrow = c(1, 1))
coef(lassofit)
```

Partial output(Shows the number of non-zero entries in dependency on λ):

```
Call:  glmnet(x = x, y = y)

      Df    %Dev   Lambda
 [1,]  0 0.00000 0.843400
 [2,]  1 0.09159 0.768500
 [3,]  1 0.16760 0.700200
 [4,]  1 0.23070 0.638000
 [5,]  1 0.28320 0.581300
 [6,]  1 0.32670 0.529700
 [7,]  1 0.36280 0.482600
 [8,]  1 0.39280 0.439800
 [9,]  2 0.42210 0.400700
[10,]  2 0.44900 0.365100
[11,]  2 0.47130 0.332700
[12,]  2 0.48980 0.303100
[13,]  3 0.51250 0.276200
[14,]  3 0.53180 0.251600
[15,]  3 0.54790 0.229300
etc.
```

To visualize:



The left figure shows how the coefficients start to decrease and finally all drop to zero as the penalty becomes more important (larger log(lambda)). The right figure shows the same but plotted against the penalty ($||\hat{\beta}_{lasso}(\lambda)||_1$). As the norm of the vector is small many of the coefficients are 0 or close to 0. As the norm increases the coefficients deviate from 0 and in the end all are different from 0.

The results from this analysis indicates the order in which the variables should be dropped from the model as the importance of the penalty increases, but it does not answer which model should be chosen, i.e. which λ results in the "best" model.

11.3 Choosing λ

For changing λ the number of variables with coefficients different from 0 changes. Which of the model should we choose. One could make the decision based on AIC or BIC (we are comparing models and these were measures proposed before to measure model fit).

Another possibility is to apply **Cross Validation** for making this choice.
For Cross Validation the sample is split into K folds, $F_1, \dots, F_k$, of equal size (or as close as possible). Then the model is fit K times: each time omitting one of the folds (let's say i) for estimating the model parameters.
The outcome of this estimation is then cross validated by comparing the actual measurements with their prediction for all data in the omitted fold k:

$$e_i(\lambda) = \sum_{j \in F_i} (y_j - \hat{y}_j(\lambda))^2$$

The CV error (also called the cross validation mse) is the error averaged over the K folds.

$$CV(\lambda) = \frac{1}{n} \sum_{i=1}^{K} e_i(\lambda)$$

When $K = n$ we call this the leave-one-out cross validation. More common choice for K are five or ten.
A plot of the cross-validation error against λ usually shows first a decrease in $CV(\lambda)$ and then an increase. Choose λ such that the cross validation error is smallest.

$$\hat{\lambda} = \underset{\lambda}{\operatorname{argmin}}\, CV(\lambda)$$

Experimenters have found this choice very conservative not eliminating "sufficiently many" predictors from the model.

The following modification has been suggested based on the principle:
"All else being equal (up to one standard error), go for the simpler model".(Ryan Tibshirani)

Given that the cross validation error can be estimated for each fold by

$$\frac{e_i(\lambda)}{n_i} = \frac{1}{n_i} \sum_{j \in F_i} (y_j - \hat{y}_j(\lambda))^2$$

where n_i is the sample size in F_i, it is possible to estimate the standard error of the cross validation error based on those K estimates (for each λ). This gives rise to the

The one standard error rule
We first find the usual minimizer $\hat{\lambda}$ and then define $\hat{\lambda}^*$, as the largest λ such that the cross validation error curve is still within one standard error of $CV(\hat{\lambda})$.

$$\hat{\lambda}^* = \max_{CV(\lambda) \leq CV(\hat{\lambda}) + SE(\hat{\lambda})} \lambda$$

R code:

```
# Cross Validation
cvlassofit<-cv.glmnet(x,y)
print(cvlassofit)
plot(cvlassofit)
coef(cvlassofit, s = "lambda.min") # to get the coefficients for the "optimal" lambda
coef(cvlassofit, s = "lambda.1se") # to get the coefficients for the one SE lambda
```

Partial output (now showing choices for λ based on cross validation):

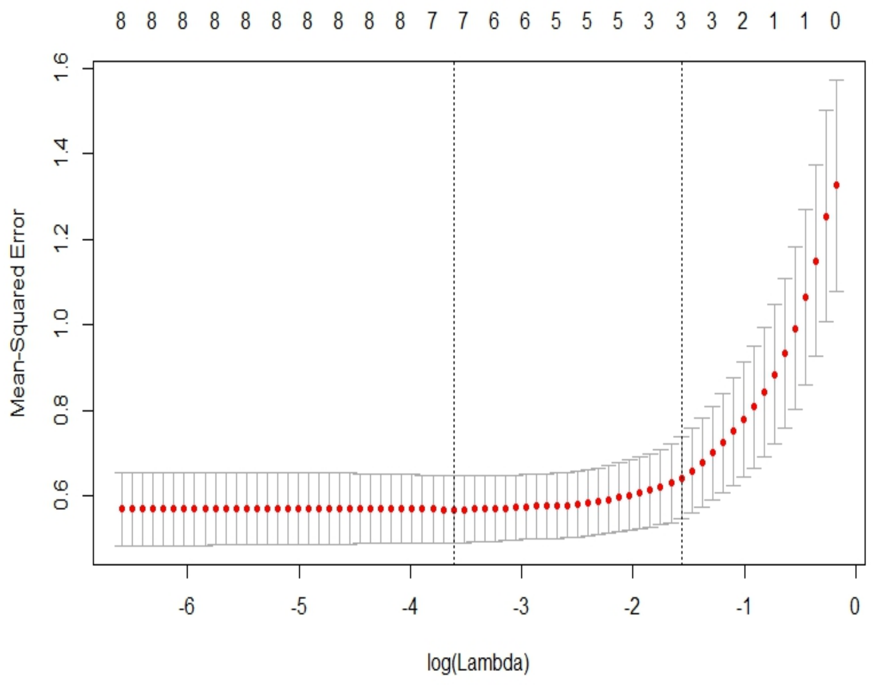
```
etc.
[61,]  8 0.65460 0.003175
[62,]  8 0.65460 0.002893
[63,]  8 0.65460 0.002636
[64,]  8 0.65470 0.002402
[65,]  8 0.65470 0.002189
[66,]  8 0.65470 0.001994
[67,]  8 0.65470 0.001817
[68,]  8 0.65470 0.001656
[69,]  8 0.65470 0.001509
[70,]  8 0.65470 0.001375

$lambda.min
[1] 0.005549189

$lambda.1se
[1] 0.2292932

attr(,"class")
[1] "cv.glmnet"
```

And to



The vertical lines in the figure show $\hat{\lambda}$ (left) and $\hat{\lambda}^*$ (right). The numbers on top of the figure give the number of non-zero coefficients. Which means that for this example instead of using seven predictors we would be using three predictors for the selected model if we would choose the one standard error estimate.
At this point one should go back and reestimate the model using the BLUE based on the variables chosen.