60611 Advanced Statistical Learning
Regularization and Variable Selection in Regression Models

Aurélie Labbe (based on D. Larocque course notes)

HEC Montréal
Sciences de la décision

Agenda

Introduction

1. **Introduction**
2. Classical variable selection methods
3. Ridge regression
4. Lasso and other methods
5. An example: Ames data
6. Grouped data
7. Using interactions
8. Sure dependence screening (SIS)
9. Generalized linear models
10. An example: German credit data
11. Other applications of regularization

Introduction

- As before, we have a target variable Y and many covariates $X_1, X_2, \ldots, X_p$.
- The covariates can be of any types: continuous, categorical (binary, nominal, ordinal).
- We will mainly use the basic linear regression setup with independent observations to introduce and describe the methods.
- Hence Y will be treated as continuous variable but we will also provide generalisations to more complex settings
- We assume we have a sample of size n from the target population and use the notation Y_i and X_{ji} to denote the value of Y and X_j for the i^{th} observation in the sample, $i = 1, \ldots, n$ and $j = 1, \ldots, p$.

The basic model

- The linear regression model with all the covariates is:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p + \epsilon$$

  where ϵ is a random variable with expectation 0 and variance σ^2, i.e. $E[\epsilon] = 0$ and $Var[\epsilon] = \sigma^2$. Moreover, the error term is assumed independent of the covariates.

- Another way to write this model is

$$\begin{aligned} E[Y|X_1, X_2, \ldots, X_p] &= \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p \\ Var[Y|X_1, X_2, \ldots, X_p] &= \sigma^2. \end{aligned}$$

- The goal is to select a "good" subset of covariates to build the best possible predictive model.

Agenda

Introduction

1. Introduction
2. **Classical variable selection methods**
3. Ridge regression
4. Lasso and other methods
5. An example: Ames data
6. Grouped data
7. Using interactions
8. Sure dependence screening (SIS)
9. Generalized linear models
10. An example: German credit data
11. Other applications of regularization

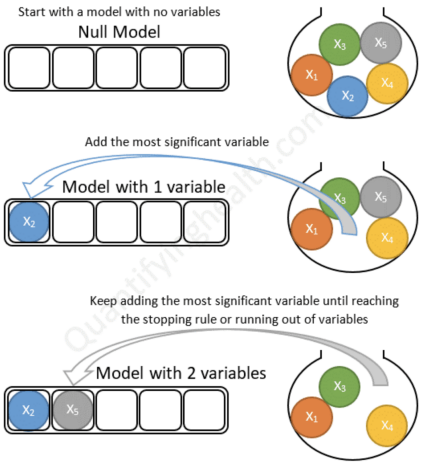
Classical variable selection methods

- Many variable selection methods were developed when the computing power was very small compared to today's standards.
- Nevertheless, some of them can still be very competitive even against modern methods.
- We'll give a description of some of them:
  - Forward selection
  - Backward selection
  - Stepwise selection
  - All Subset Selection

Forward method

- The "forward" method is simple to describe.
- At a given step, let S_{in} be the set of covariates in the model and S_{out} be the set of covariates not in the model.
- Select an entry criterion and condition and start with $S_{in} = \emptyset$ and $S_{out} = \{X_1, \ldots, X_p\}$. **Note that the intercept is always in the model**.
  - **Add** each variable in S_{out}, one at a time, to the model that contains the variables in S_{in} and compute the entry criterion for each of them.
  - Select the variable with the best value of the entry criterion, but only if it satisfies the entry condition. If no variable satisfy the entry condition, then the process stops and the selected covariates are the ones in S_{in}.
  - If the entry condition is satisfied, add the best variable to S_{in} and remove it from S_{out}.
  - Repeat 1 to 3.

Example: forward method with 5 variables



https://quantifyinghealth.com/

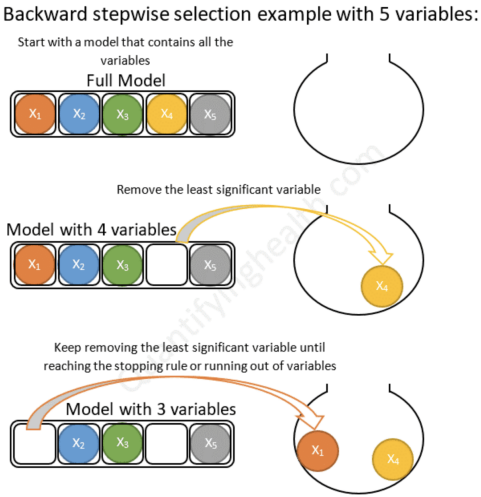
Forward method (cont')

- With the forward method, when a variable is entered, it stays in the model.
- **Entry criterion and conditions can be** :
  - The p-value of the added variable. The entry condition is that the p-value of the best variable in S_{out} is smaller than a given threshhold, for example 0.1, or 0.05.
  - AIC (or BIC). That is, at a given step, we enter the variable that decreases the AIC/BIC the most. If no variable in S_{out} can decrease the AIC/BIC, then we stop.
  - Alternatively, a validation set can be used. The variable that decreases the error the most on the validation set is entered.

Backward method

- The "backward" method is very similar but goes in the opposite direction.
- Select an exit criterion and condition and start with $S_{in} = \{X_1, \ldots, X_p\}$ and $S_{out} = \emptyset$.
  - **Remove** each variable in S_{in}, one at a time, to the model that contains the variables in S_{in} and compute the exit criterion for each of them.
  - Select the variable with the best value of the exit criterion, but only if it satisfies the exit condition. If no variable satisfy the exit condition, then the process stops and the selected covariates are the ones in S_{in}.
  - If the exit condition was satisfied, remove the best variable from S_{in} and add in to S_{out}.
  - Repeat 1 to 3.

Example: backward method with 5 variables



https://quantifyinghealth.com/

Backward method (cont')

- With the backward method, when a variable is removed, it cannot go back in the model later.
- **Exit criterion and conditions can be** :
  - The p-value of the removed variable. The exit condition is that the p-value of the worst variable in S_{in} is larger than a given threshold, for example 0.1, or 0.05.
  - AIC (or BIC). That is, at a given step, we remove the variable that decreases the AIC/BIC the most. If no variable in S_{in} can increase the AIC/BIC, then we stop.
  - Alternatively, a validation set can be used. The variable that decreases the error the most on the validation set is entered.

Stepwise method

- The "stepwise" method is generally preferable and combines the two previous methods.
- We can perform it starting with 0 variables or starting with all the variables.
- For example, here is one way to perform it starting with 0 variables:
  - Perform a forward step.
  - Perform backward steps as long as variables are removed.
  - If no variable was entered and no variable was removed, then stop. The selected covariates are the ones in S_{in}.
  - If one variable was entered and/or at least one variable was removed, then go back to the forward step.

Stepwise method (cont')

- With the stepwise method, a variable can enter the model and be removed later in the process.
- This can be useful when the covariates are correlated.
- Again, entry/exit criteria can be based on p-values, on measures like the AIC or BIC, or on a validation set.

All Subset Selection

- The forward, backward and stepwise selection methods perform a limited search and are not looking at all possible models.
- In most applications, this is sufficient.
- As its name indicates the "all subsets" method evaluates all possible models and select the one with the best value of a criterion, for example the AIC or BIC.
- The problem is that the number of potential models grows exponentially.
  - In fact, with p covariates, there is 2^p possible models.
  - With only $p = 30$ variables, there is already 1,073,741,824 models, that is more than 1 billion models.
- Nevertheless, efficient algorithms can perform an all subset selection with an ever increasing number of covariates. But it's not clear that such methods are preferable to a simple stepwise.

R functions

- The function `stepAIC` in the `MASS` package is useful to perform stepwise selection with many different families of models.
- All subsets selection can be performed with the `leaps` and `bestglm` packages.

Simple example

- Here is an example of application of the previous methods with synthetic data.
- Consider 3 covariates X_1, X_2 and X_3
- Suppose that X_1 and X_2 are correlated and X_3 is independent of X_1 and X_2
- Data are generated as follows, with $n = 100$ for training and $n = 1,000$ for testing

$$\begin{aligned} X_1 &\sim \mathcal{N}(0,1), \\ X_2 &= X_1 + \epsilon_x, \\ X_3 &\sim \mathcal{N}(0,1), \\ Y &= X_1 + X_2 + X_3 + \epsilon_y \end{aligned}$$

where ϵ_x and ϵ_y are normal variables with mean 0 and standard deviation 0.5 and 3 respectively

Simple example: simulating data

Code R

```
# generate training and test data

set.seed(364575)

# train data
> n=100
> x1=rnorm(n)
> x2=x1+.5*rnorm(n)
> x3=rnorm(n)
> y=x1+x2+x3+3*rnorm(n)
> matx=cbind(x1,x2,x3)
> dat=data.frame(y,x1,x2,x3)
> names(dat)=c("y","x1","x2","x3")

# test data
> ntest=10000
> x1new=rnorm(ntest)
> x2new=x1new+.5*rnorm(ntest)
> x3new=rnorm(ntest)
> ynew=x1new+x2new+x3new+3*rnorm(ntest)
> matxnew=cbind(x1new,x2new,x3new)
> datnew=data.frame(x1new,x2new,x3new)
> names(datnew)=c("x1","x2","x3")
```

Simple example: simulating data (cont')

Code R

```
> cor(cbind(y,x1,x2,x3))
           y        x1        x2        x3
y  1.0000000 0.6080312 0.5857846 0.3119057
x1 0.6080312 1.0000000 0.8818763 0.1216682
x2 0.5857846 0.8818763 1.0000000 0.0914663
x3 0.3119057 0.1216682 0.0914663 1.0000000
```

- We note that X_1 and X_2 are highly correlated (0.88) which might reduce the precision of their parameters estimates.

Simple example: ordinary OLS

Code R

```
# ordinary OLS
> lm.fit=lm(y~x1+x2+x3)
> summary(lm.fit)

Call:
lm(formula = y ~ x1 + x2 + x3)

Residuals:
    Min      1Q  Median      3Q     Max
-7.1025 -1.8028  0.2313  2.1827  8.6000

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   0.1114     0.3318   0.336  0.73784
x1            1.4948     0.6640   2.251  0.02665 *
x2            0.8642     0.5824   1.484  0.14115
x3            1.0087     0.3165   3.187  0.00194 **
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 3.166 on 96 degrees of freedom
Multiple R-squared:   0.44,Adjusted R-squared:  0.4225
F-statistic: 25.14 on 3 and 96 DF,  p-value: 4.331e-12
```

- We see that $\hat{\beta}_1$ and $\hat{\beta}_2$ are not as well estimated as $\hat{\beta}_3$ (true value is 1).
- The standard errors of $\hat{\beta}_1$ and $\hat{\beta}_2$ are about twice as large compared to $\hat{\beta}_3$

Simple example: stepwise search with AIC

- Here is how to perform a stepwise search with the `stepAIC` function, and then to obtain the predicted values for the new data (the test set).

Code R

```
> library(MASS)

# stepwise with AIC as the criterion
> step.AIC=stepAIC(lm.fit,direction="both")

Start:  AIC=234.4
y ~ x1 + x2 + x3

       Df Sum of Sq     RSS    AIC
<none>               962.16 234.40
- x2    1    22.065  984.22 234.67
- x1    1    50.798 1012.95 237.55
- x3    1   101.793 1063.95 242.46

# The AIC solution is already fitted in lm.fit
> predaic=predict(lm.fit,newdata=datnew)
```

Simple example: stepwise search with BIC

- We can do the same thing using the BIC...

Code R

```
# stepwise with BIC as the criterion
> step.BIC=stepAIC(lm.fit,direction="both",k=log(n))

Start:  AIC=244.82
y ~ x1 + x2 + x3

       Df Sum of Sq     RSS    AIC
- x2    1    22.065  984.22 242.48
<none>               962.16 244.82
- x1    1    50.798 1012.95 245.36
- x3    1   101.793 1063.95 250.27

Step:  AIC=242.48
y ~ x1 + x3

       Df Sum of Sq     RSS    AIC
<none>               984.22 242.48
+ x2    1     22.06  962.16 244.82
- x3    1     98.73 1082.95 247.44
- x1    1    566.78 1551.00 283.36

# The BIC solution includes only x1 and x3
> lm.fitbic=lm(y~x1+x3)
> predbic=predict(lm.fitbic,newdata=datnew)
```

Agenda

Introduction

1. Introduction
2. Classical variable selection methods
3. **Ridge regression**
4. Lasso and other methods
5. An example: Ames data
6. Grouped data
7. Using interactions
8. Sure dependence screening (SIS)
9. Generalized linear models
10. An example: German credit data
11. Other applications of regularization

Ridge Regression

- Ridge regression is one of the first shrinkage method.
- It was introduced by Hoerl and Kennard (1970), as a potential solution to the multicollinearity problem.
- Multicollinearity (or collinearity) means that some of the covariates (or linear combination of covariates) are very correlated with each other
- The main effect of multicollinearity is that it can potentially make the estimates of some of the regression parameters (some of the β's) very unstable:
  - they will have very large standard errors
  - **small changes in the data can cause great changes in the values of these estimates**
  - any inference on them (test or confidence interval) will also be very imprecise.

Colinearity and predictions

- In theory, multicollinearity affects the individual covariate effects but not the global predictive power of the model
- More precisely, the global effect of a group of highly correlated covariates will still be accounted for by the model, but the allocation of this global effect across the covariates will be difficult to make.
- However, the imprecision caused by multicollinearity can also affect the predictions. For instance, many variable selection methods can have trouble with multicollinearity.
- To understand ridge regression, we must see how multicollinearity affects the estimates....

Matrix notation of the regression model

- To understand how multicollinearity affects the estimates of the β coefficients, let's write the regression model using matrix notation.
- We note:

$$\mathbf{Y} = \begin{bmatrix} Y_1 \\ \vdots \\ Y_n \end{bmatrix}, \quad \tilde{\mathbf{X}} = \begin{bmatrix} 1 & X_{11} & \dots & X_{1p} \\ \vdots & \vdots & & \\ 1 & X_{n1} & \dots & X_{np} \end{bmatrix}, \quad \boldsymbol{\beta} = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \dots \\ \beta_p \end{bmatrix}, \quad \epsilon = \begin{bmatrix} \epsilon_1 \\ \vdots \\ \epsilon_n \end{bmatrix}$$

- The regression model can then be written as:

$$\boldsymbol{Y} = \tilde{\mathbf{X}}\boldsymbol{\beta} + \boldsymbol{\epsilon}$$

- The OLS estimates of $\boldsymbol{\beta} = (\beta_0, \beta_1, \dots, \beta_p)'$ are given by

$$\hat{\boldsymbol{\beta}}_{OLS} = (\tilde{\mathbf{X}}'\tilde{\mathbf{X}})^{-1}\tilde{\mathbf{X}}'\mathbf{Y}.$$

Matrix notation of the regression model (cont')

- The covariance-matrix of $\hat{\boldsymbol{\beta}}_{OLS}$ is

$$\sigma^2(\tilde{\mathbf{X}}'\tilde{\mathbf{X}})^{-1}.$$

- It can be estimated by

$$\hat{\sigma}^2(\tilde{\mathbf{X}}'\tilde{\mathbf{X}})^{-1},$$

where

$$\hat{\sigma}^2 = \frac{\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2}{n-p-1},$$

is the estimate of the error variance, and

$$\hat{Y}_i = \hat{\beta}_0 + \hat{\beta}_1 X_{1i} + \hat{\beta}_2 X_{2i} + \cdots + \hat{\beta}_p X_{pi}$$

are the predicted values.

How multicollinearity affects the estimates

- When multicollinearity is present, the matrix $(\tilde{\mathbf{X}}'\tilde{\mathbf{X}})$ can become nearly singular (the inverse does not exist) and some eigenvalues of $(\tilde{\mathbf{X}}'\tilde{\mathbf{X}})$ can be small.
- Hence, some values on the diagonal of $(\tilde{\mathbf{X}}'\tilde{\mathbf{X}})^{-1}$ can be large.
  - Hence, the standard errors of some parameters can be large, that is imprecise.
- Consequently, some estimates can also be large.
- Ridge regression tries to solve this by imposing a penalty on the magnitudes of the estimates.

Ridge regression

- Instead of using the OLS criterion, ridge regression minimizes:

$$\sum_{i=1}^{n}(Y_i - (\beta_0 + \beta_1 X_{1i} + \beta_2 X_{2i} + \cdots + \beta_p X_{pi}))^2 + \lambda \sum_{j=1}^{p} \beta_j^2, \qquad (1)$$

  where $\lambda \geq 0$ is a complexity parameter.

- Obviously, using $\lambda = 0$ reduces to OLS.
- Note also that the penalty is not applied to the intercept β_0:
  - If we do apply the penalty on β_0, we would have $\widehat{Y + c} \neq \hat{Y} + c$
  - Leaving β_0 out of the penalty avoids that.

Ridge regression (cont')

- Another way to define ridge regression is as a constrained least-squares problem. The ridge estimator minimizes

$$\sum_{i=1}^{n}(Y_i - (\beta_0 + \beta_1 X_{1i} + \beta_2 X_{2i} + \cdots + \beta_p X_{pi}))^2$$

  subject to

$$\sum_{j=1}^{p} \beta_j^2 \leq t,$$

  where t is a complexity parameter.

- To each value of λ in the previous slide, there corresponds a value of t.
- In fact, there is a one-to-one correspondence between λ and t.
- We can view the parameter t as a "budget" that we can spend on parameters.
- Ridge regression minimizes the average squared error with a limited amount of resources.

Effect of ridge regression on the β estimates

- Generally, the covariates are standardized before computing the estimates.
  - This is because the same penalty is applied to all the covariates and so it is usually more reasonable to put them on the same scale.
- Let $\tilde{\mathbf{X}}_s$ be the $n \times p$ matrix containing the standardized covariates, without a column of 1 for the intercept.
- The intercept is estimated by $\hat{\beta}_0 = \bar{Y}$.
- It can be shown that the ridge estimates of the parameters of the covariates (the intercept is not included here) are given by

$$\hat{\boldsymbol{\beta}}_{ridge} = (\tilde{\mathbf{X}}_s' \tilde{\mathbf{X}}_s + \lambda \mathbf{I})^{-1} \tilde{\mathbf{X}}_s' \mathbf{Y},$$

where $\mathbf{I}$ is the $p \times p$ identity matrix.

Effect of ridge regression on the β estimates (cont')

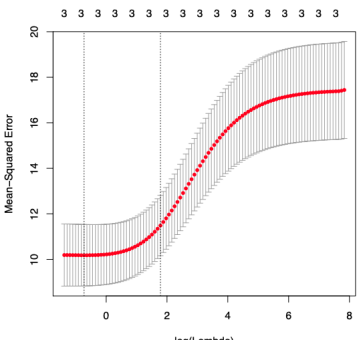
- We see that the term $\lambda \mathbf{I}$ adds a (usually) small number to the diagonal of $(\tilde{\mathbf{X}}_s' \tilde{\mathbf{X}}_s)$ before the inversion.
- This moves the matrix $(\tilde{\mathbf{X}}_s' \tilde{\mathbf{X}}_s)$ away from being close to be singular.
- Hence, the elements on the diagonal of $(\tilde{\mathbf{X}}_s' \tilde{\mathbf{X}}_s)^{-1}$ will not be as large and thus the standard errors will tend to be smaller.
- However, the penalty term introduces a bias in the estimates.
- This is why the goal is to find a value of λ that will be a good compromise between the variance and the bias.

Simple example: applying ridge regression

- The `glmnet` package can perform ridge, lasso, and more generally elastic net regressions for linear regression

Code R

```
# ridge regression with the glmnet package
> library(glmnet)

# to select the best value of lambda by cross-validation
# alpha=0 is used to specify ridge regression
> glmnet.cv = cv.glmnet(matx, y, alpha=0)
> plot(glmnet.cv)
```



Simple example: applying ridge regression (cont')

- We can then extract the best value of λ and get the predictions with the model that uses it.

Code R

```
> bestlridge=glmnet.cv$lambda.min
> bestlridge
[1] 0.4833757

> predridge=predict(glmnet.cv,new=matxnew,s=bestlridge)
> predict(glmnet.cv,s=bestlridge,type="coefficients")
4 x 1 sparse Matrix of class "dgCMatrix"
                      1
(Intercept) 0.09284517
x1          1.30583677
x2          0.91164666
x3          0.91963083
```

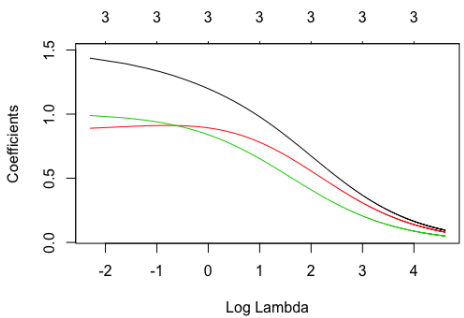
- $\hat{\beta}_1$ and $\hat{\beta}_2$ are closer to their true values compared to OLS.
- In general, the estimates will shrink (all $\hat{\beta} \to 0$ when $\lambda \to \infty$).
- But some coefficients can also increase at first when λ just begins to increase. This is the case for $\hat{\beta}_2$ here (.86 for OLS and .91 for ridge regression) with the selected lambda.

Simple example: applying ridge regression (cont')

- The following graph shows the evolution of the estimates as λ varies.

Code R

```
> plot(glmnet(matx, y, alpha=0,lambda=seq(0,100,.1)), xvar = "lambda", label = TRUE)
```



- The left end side of the graph corresponds to OLS, and then as λ increases, we see how the parameter for X_2 slightly increases at first, but then it goes down to 0 like the other parameters.

Simple example: applying ridge regression (cont')

- Finally, we can compute the mean squared error on the test set for the three models: OLS (using stepwise with AIC), OLS (using stepwise with BIC) and ridge regression with λ selected by cross-validation.

Code R

```
> # mean squared error for the three models: AIC, BIC, ridge
> c(mean((predaic-ynew)^2),mean((predbic-ynew)^2),mean((predridge-ynew)^2))
[1] 9.175325 9.451781 9.098615
```

- We see that the best performance is achieved by ridge regression followed by OLS (stepwise with AIC).
- Note that the best possible value for the predictive mean squared error is 9 because the error term, which cannot be predicted, has a variance of 9. So we are quite close to the best possible performance.

Final comments

- Originally, ridge regression was developed as a cure for multicolinearity. Today, it is viewed in a broader sense as a shrinkage method.
- Such methods trade a little bias in the estimation of the coefficients in return for a reduction of their variance, with the hope that the total mean squared error will be reduced compared to OLS.
- It is called a shrinkage method because, as we saw in the example, the coefficients are shrunk towards 0.
- Ridge regression can also be seen as a regularization method.
- In general, regularization methods are ways to promote simpler models to try to avoid over-fitting. This is often very important in situations where the number of free parameters is large compared to the sample size, because this is where over-fitting is proned to occur.

Final comments (cont')

- The forward, backward, and stepwise method perform "hard" variable selection, in the sense that a covariate is either completely in or out of the model.
- They do not perform shrinkage.
- Ridge regression does shrinkage but is not a variable selection method because all covariates remain in the model.
- It has been suggested that a way to perform variable selection with ridge regression is to select a threshold value, and to set to 0 all parameters that are below this threshold. But this does not seem to be used often in practice.
- The methods in the next section can simultaneously perform variable selection and shrinkage.

Agenda

Introduction

1. Introduction
2. Classical variable selection methods
3. Ridge regression
4. **Lasso and other methods**
   (a) The Lasso
   (b) The Elastic Net
   (c) The relaxed Lasso
5. An example: Ames data
6. Grouped data
7. Using interactions
8. Sure dependence screening (SIS)
9. Generalized linear models
10. An example: German credit data
11. Other applications of regularization

The LASSO

- The lasso, proposed by Tibshirani (1996), was a major breakthrough for regularization and shrinkage methods.
- The lasso estimates minimize

$$\sum_{i=1}^{n}(Y_i - (\beta_0 + \beta_1 X_{1i} + \beta_2 X_{2i} + \cdots + \beta_p X_{pi}))^2 + \lambda \sum_{j=1}^{p} |\beta_j|.$$

- It is very similar to ridge regression.
- The L_2 ridge penalty $\sum_{j=1}^{p}(\beta_j)^2$ is replaced by the L_1 lasso penalty $\sum_{j=1}^{p} |\beta_j|$.
- However, unlike ridge regression, there is no closed form expression for the lasso estimates.

The LASSO (cont')

- The lasso solution can also be written as a constrained minimization problem. It minimizes

$$\sum_{i=1}^{n}(Y_i - (\beta_0 + \beta_1 X_{1i} + \beta_2 X_{2i} + \cdots + \beta_p X_{pi}))^2,$$

subject to

$$\sum_{j=1}^{p} |\beta_j| \leq t.$$

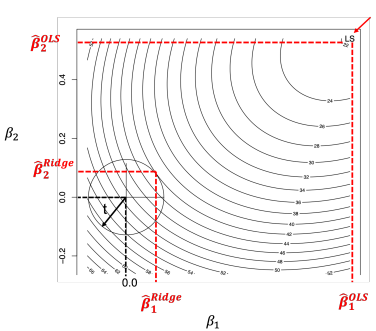
- Like for ridge regression there is a one-to-one correspondence between λ and t.

The LASSO (cont')

- Another major difference between ridge and lasso is that, as λ increases, some of the coefficients will become exactly 0.
- This is unlike ridge regression where all coefficients remain nonnegative, and only converge to 0 when λ goes to ∞.
- One way to intuitively see it is to examine the ridge and lasso constraints in the next two figures with $p = 2$...

Ridge constraints: example with 2 variables ($p = 2$)

- The curves give the value of the least-squares criterion as a function of β_1 (x-axis) and β_2 (y-axis). The least-squares solution is on the top right corner of the figure.



- **Least square criterion**:

$$LS = \sum_{i=1} n(y_i - \beta_1 x_{1i} - \beta_2 x_{2i})^2$$

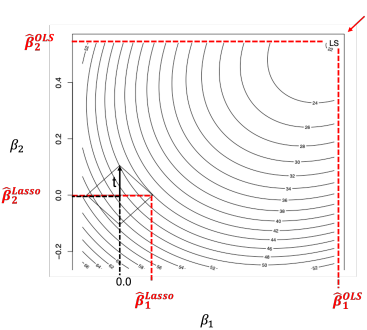
- **Ridge constraint (circle)**:

$$\sum_{j=1}^{p} \beta_j^2 = \beta_1^2 + \beta_2^2 \leq t$$

- The ridge constraint is a circle. All points inside the circle are admissible. If the least-squares solution is outside (as in this example), then the ridge solution will happen somewhere along the circle edge.

Lasso constraints: example with 2 variables ($p = 2$)

- The curves give the value of the least-squares criterion as a function of β_1 (x-axis) and β_2 (y-axis). The least-squares solution is on the top right corner of the figure.



- **Least square criterion**:

$$LS = \sum_{i=1} n(y_i - \beta_1 x_{1i} - \beta_2 x_{2i})^2$$

- **Lasso constraint (diamond)**:

$$\sum_{j=1}^{p} |\beta_j| = |\beta_1| + |\beta_2| \leq t$$

The four corners of the diamond correspond to the situation where one parameter is exactly 0.

- With the lasso, the constraint is a diamond. But the four corners of the diamond stick out compared to the other points and will attract the minimum sooner. So, as t decreases, the lasso solution will hit a corner eventually and set a first parameter to 0, thus performing variable selection.

Lasso vs ridge constraints

- From the figures, we see that lasso regression performs shrinkage and variable selection at the same time.
- Usually, the covariates are standardized before estimating the coefficients, like for ridge regression.
- As a matter of fact, most packages that perform lasso and similar methods will standardized the covariates by default (and this can optionally be overridden).

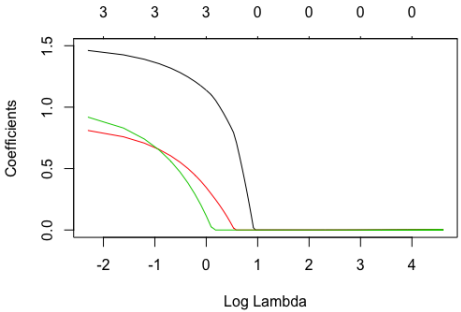
Lasso: estimates as a function of λ

- Here is the path of estimates for the previous simple example, but with lasso regression.
- Note that setting `alpha=1` in `glmnet` produces lasso regression.

Code R

```
> plot(glmnet(matx, y, alpha=1,lambda=seq(0,100,.1)), xvar = "lambda", label = TRUE)
```

Lasso: estimates as a function of λ (cont')



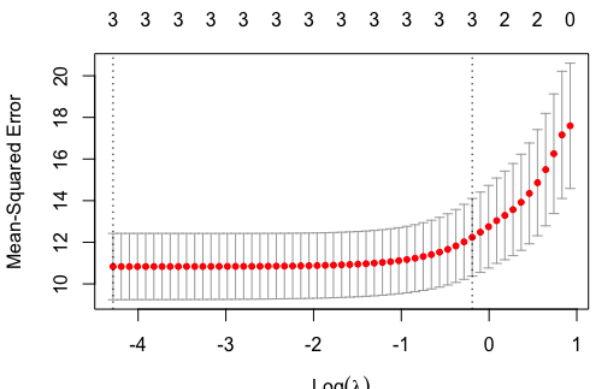
- We see that $\hat{\beta}_3$ reaches 0 first, followed by $\hat{\beta}_2$, and when λ is about 2.7 (exp(1)), then all coefficients are at 0.
- Hence, depending on the value of λ selected, the model selects either all the variables, only X_1 and X_2, or only X_1.

Lasso: optimal λ

- Let's select λ by cross-validation, as we did for ridge regression, and evaluate the performance of the resulting model.

Code R

```
> glmnet.cv = cv.glmnet(matx, y, alpha=1)
> plot(glmnet.cv)
```



Lasso: optimal λ

Code R

```
> bestllasso=glmnet.cv$lambda.min
> bestllasso
[1] 0.01376671
> predlasso=predict(glmnet.cv,new=matxnew,s=bestllasso)
> predict(glmnet.cv,s=bestllasso,type="coefficients")
4 x 1 sparse Matrix of class "dgCMatrix"
                     1
(Intercept) 0.1117049
x1          1.4912090
x2          0.8560317
x3          0.9963326
>
> # mean squared error for:
> # OLS (stepwise AIC); OLS (stepwise BIC); ridge, lasso
> c(mean((predaic-ynew)^2),mean((predbic-ynew)^2),
+   mean((predridge-ynew)^2),mean((predlasso-ynew)^2))
[1] 9.175325 9.451781 9.098615 9.167900
```

Lasso vs ridge regression

- In this example, ridge regression performs better than the lasso, who does slightly better than OLS.
- In fact the selected λ for the lasso is 0.015, which is a lot smaller than for ridge which was 0.70. The lasso coefficients are only slightly shrunk and are very close to the OLS estimates.
- When some covariates are strongly correlated, ridge regression has the tendency to shrink them together to get some kind of "average" effect. On the other hand, the lasso will tend to favor one of the correlated covariates. This phenomenon can be observed somewhat in the last two plots.
- In general, no method (lasso or ridge) dominates the other and which one is the best depends on the specific data.
- Often the lasso and ridge will have a similar predictive performance. The big advantage of the lasso is to be able to achieve it with a simpler model (less variables).

Agenda

Introduction

1. Introduction
2. Classical variable selection methods
3. Ridge regression
4. **Lasso and other methods**
   (a) The Lasso
   (b) The Elastic Net
   (c) The relaxed Lasso
5. An example: Ames data
6. Grouped data
7. Using interactions
8. Sure dependence screening (SIS)
9. Generalized linear models
10. An example: German credit data
11. Other applications of regularization

The elastic net

- The elastic net (Zhou and Hastie 2005) bridges the gap between ridge and lasso regression, by combining both the L_1 and L_2 penalties.
- It is the solution to

$$\min_{\beta_0,\ldots,\beta_p} \quad \frac{1}{n}\sum_{i=1}^{n}(Y_i - (\beta_0 + \beta_1 X_{1i} + \ldots, +\beta_p X_{pi}))^2 + \lambda[(1-\alpha)/2\sum_{j=1}^{p}(\beta_j)^2 + \alpha\sum_{j=1}^{p}|\beta_j|].$$

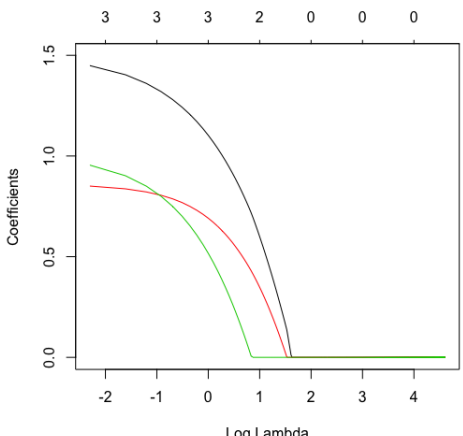
- As before λ is the shrinkage parameter, and the new parameter α is a weighting factor between the two penalties.
- When $\alpha = 0$, we have the ridge penalty, and when $\alpha = 1$, we have the lasso penalty.

Simple example (cont') with elastic net

- Let's continue the example by using the elastic net with $\alpha = .5$.

Code R

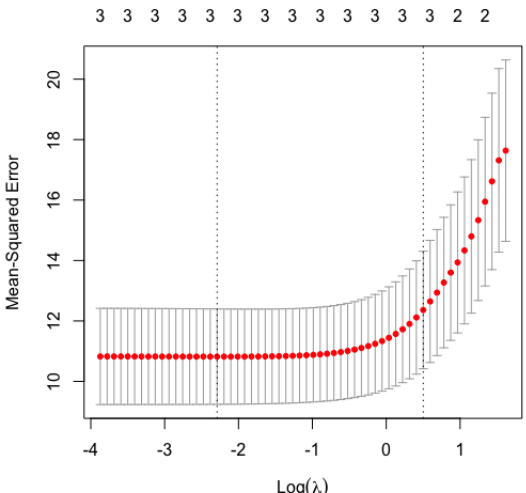
```
> plot(glmnet(matx, y, alpha=.5,lambda=seq(0,100,.1)), xvar = "lambda", label = TRUE)
```



Simple example (cont') with elastic net

Code R

```
> glmnet.cv = cv.glmnet(matx, y, alpha=0.5)
> plot(glmnet.cv)
```



Simple example (cont') with elastic net

Code R

```
> bestllel5=glmnet.cv$lambda.min
> bestllel5
[1] 0.1012784
> predelasticnet5=predict(glmnet.cv,new=matxnew,s=bestllel5)
> predict(glmnet.cv,s=bestllel5,type="coefficients")
4 x 1 sparse Matrix of class "dgCMatrix"
                      1
(Intercept) 0.1090233
x1          1.4472756
x2          0.8511481
x3          0.9540014

> # mean squared error for: OLS (stepwise AIC);
> # OLS (stepwise BIC); ridge, lasso, elastic net with alpha=1/2
> c(mean((predaic-ynew)^2),mean((predbic-ynew)^2),mean((predridge-ynew)^2),
+   mean((predlasso-ynew)^2), mean((predelasticnet5-ynew)^2))
[1] 9.175325 9.451781 9.098615 9.167900 9.139655
```

- The elastic net with $\alpha = 0.5$ does better than the lasso but not as well as ridge regression in this example.
- Note that the package `glmnet` does not have a function to automatically select the parameter α.

Agenda

Introduction

1. Introduction
2. Classical variable selection methods
3. Ridge regression
4. **Lasso and other methods**
   (a) The Lasso
   (b) The Elastic Net
   (c) The relaxed Lasso
5. An example: Ames data
6. Grouped data
7. Using interactions
8. Sure dependence screening (SIS)
9. Generalized linear models
10. An example: German credit data
11. Other applications of regularization

Relaxed Lasso

- By construction, the lasso estimates are biased towards 0.
- When there is a large number of covariates, a large penalty may be needed to perform a good variable selection, i.e. to be able to shrunk enough variables to 0.
- But then the estimates of the effects of the remaining covariates might be too small and may not be optimal to use for predictions.
- A simple solution to that is to run the lasso first to select the covariates, then do a OLS on the selected covariates.

Relaxed Lasso (cont')

- A more general idea is to run the lasso to select the covariates, then to run the lasso again using only these variables.
- The idea is that the estimated λ in the second step should be smaller, leading to estimates that are less shrunken, because the unimportant variables are already out.
- The relaxed lasso (Meinshausen, 2007) is even more general and includes these two hybrid methods.
- Each value of λ in the lasso provides a set of variables with non-zero coefficients (the selected variables).
- The relaxed lasso allows to use a different amount of shrinkage for these sets of variables. Hence, it involves another tuning parameter that must be estimated.
- Relaxed lasso is implemented in the `glmnet` package with the option `relax=TRUE`.

Agenda

Introduction

1. Introduction
2. Classical variable selection methods
3. Ridge regression
4. Lasso and other methods
5. **An example: Ames data**
6. Grouped data
7. Using interactions
8. Sure dependence screening (SIS)
9. Generalized linear models
10. An example: German credit data
11. Other applications of regularization

An Example: Ames Data

- We will use this data set throughout this course (De Cock, 2011)
- It was introduced as a modern alternative to the well-known Boston Housing data set.
- It has 2,330 observations and 82 variables and contains information from the Ames Assessor's Office used in computing assessed values for individual residential properties sold in Ames, IA from 2006 to 2010.
- The data set description can be found here: `https://ww2.amstat.org/publications/jse/v19n3/decock/DataDocumentation.txt`

Ames Data

- The data has 82 columns which include 23 nominal, 23 ordinal, 14 discrete, and 20 continuous variables (and 2 additional observation identifiers).
- The usual goal is to predict the variable SalePrice.
- The package `AmesHousing` contains the raw data and also pre-processed versions of them.

Ames Data: data pre-processing

- We first construct a function that will be used to combine together levels with few observations, for a factor variable.

Code R

```
# function to combine levels, with less than a specified number of
# observations, of a factor variables

comblev=function(x,nmin)
{
# x = variable (must be a factor, if not the original variable is returned)
# nmin = levels with less than nmin observations will be combined
# output
#   same variable with a new level called "othcomb" replacing the
#    combined levels
# (NA's are not affected)

   if(!is.factor(x)) {return(x)}
   library(rockchalk)
   ta=table(x)
   combineLevels(x,levs = names(table(x))[table(x)<nmin], newLabel=c("othcomb"))
}
```

Ames Data: data pre-processing (cont')

Code R

```
# Prepare the ames housing data set.
# This version of the data uses the ordered factor as numeric variables and
# the non-ordered factor are left as factors, but the levels
# with less than 30 observations are combined.
# In the end, we have 22 factors and 57 numeric covariates, and 1 target "Sale_Price".

> library(AmesHousing)
> ames=make_ordinal_ames()
# remove an observation with a missing
> ames=ames[!is.na(ames$Electrical),]

# remove the variable "Utilities" because it is almost
#  constant with frequencies (1,2,2926).
> ames$Utilities=NULL

# converts the target variable (1 = 1000)
> ames$Sale_Price=ames$Sale_Price/1000

# get the names of the ordinal variables
> ord_vars=vapply(ames, is.ordered, logical(1))
> namored=names(ord_vars)[ord_vars]
# converts the ordered factors to numeric
#  (this preserves the ordering of the factor)
> ames[,namored]=data.frame(lapply(ames[,namored],as.numeric))

# get the names of the factor variables
> fac_vars=vapply(ames, is.factor, logical(1))
> namfac=names(fac_vars)[fac_vars]
```

Ames Data: data pre-processing (cont')

Code R

```
# group together levels with less than 30 observations
> ames=data.frame(lapply(ames,comblev,nmin=30))

# remove the space in the values (string) of some variables to prevent problems later
> ames[,"Exterior_1st"]=as.factor(gsub(" ","",ames[,"Exterior_1st"]))
> ames[,"Exterior_2nd"]=as.factor(gsub(" ","",ames[,"Exterior_2nd"]))

> num_vars=vapply(ames, is.numeric, logical(1))
# names of the numeric variables
> namnum=names(num_vars)[num_vars]

> dim(ames)
[1] 2929   80
> summary(ames)
```

- This version of the data set, `ames`, has 2929 observations and contains 22 factors and 58 numeric covariates, including 1 target Sale_Price.
- As explained in the `make_ames` function documentation, some observations and variables were removed, and 2 new variables were added.
- The factors have been consolidated. All levels with less than 30 observations are grouped together.

Ames Data: data pre-processing (cont')

- We also prepare another version of the data, `amesdum` where the factor variables are replaced by dummy variables.
- This version will be used when we apply methods that can not handle factor variables directly.

Code R

```
# Create dummy variables for the factors

>library(fastDummies)
>amesdum=dummy_cols(ames, remove_first_dummy=TRUE, remove_selected_columns=TRUE)

# Now all variables are numeric.
# There are 160 covariates and 1 target "Sale_Price".

> dim(amesdum)
[1] 2929  161
> summary(amesdum)
```

Ames Data: data pre-processing (cont')

- We divide both versions of the data into a training set (ntrain=1000) and a test set (ntest=1929) of "new" data to be predicted, that will be used to evaluate the performance of the different methods.

Code R

```
# Splitting the data into a training (ntrain=1000) and a
#  test (ntest=1929) set

set.seed(489565)
ntrain=1000
ntest=nrow(ames)-ntrain
indtrain=sample(1:nrow(ames),ntrain,replace=FALSE)

xdum=amesdum
xdum$Sale_Price=NULL
xdum=as.matrix(xdum)

amestrain=ames[indtrain,]
amestest=ames[-indtrain,]
amesdumtrain=amesdum[indtrain,]
amesdumtest=amesdum[-indtrain,]
xdumtrain=xdum[indtrain,]
xdumtest=xdum[-indtrain,]
```

Ames Data: data analysis

- Next is first a simple wrapper function to apply `glmnet` and get the predictions and coefficients for the tuning parameter with minimum CV error, and with the 1 SE rule.
- It also computes the Mean Absolute Error for the test set:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

- And the Mean Squared Error :

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

- It also produces some plots

Ames Data: data analysis (cont')

Code R

```
wrapglmnet=function(xtrain,ytrain,xtest,ytest=NULL,alpha)
{
   library(glmnet)
   par(mfrow=c(2,2))
   plot(glmnet(x=xtrain,y=ytrain,alpha=alpha),xvar = "lambda", label = TRUE)
   cv=cv.glmnet(x=xtrain,y=ytrain, alpha=alpha)
   plot(cv)
   pred=predict(cv,new=xtest,s="lambda.min")
   pred1se=predict(cv,new=xtest,s="lambda.1se")
   err=NA
   if(!is.null(ytest))
   {
      plot(ytest,pred)
      plot(ytest,pred1se)
      err=data.frame(mean(abs(pred-ytest)),mean((pred-ytest)^2),
      mean(abs(pred1se-ytest)),mean((pred1se-ytest)^2))
      names(err)=c("MAE", "MSE", "MAE_1SE","MSE_1SE")
   }

   co=predict(cv,s="lambda.min",type="coefficients")
   co=as.matrix(co)
   co=co[co[,1] != 0,,drop=FALSE]
   co1se=predict(cv,s="lambda.1se",type="coefficients")
   co1se=as.matrix(co1se)
   co1se=co1se[co1se[,1] != 0,,drop=FALSE]

   out=list(err,co,co1se,pred,pred1se)
   names(out)=c("error","coef","coef1se","pred","pred1se")
   out
}
```

Ames Data: basic Lasso

- We first apply the basic lasso.

Code R

```
# basic lasso
> set.seed(123456)
> las=wrapglmnet(xdumtrain,amesdumtrain$Sale_Price,xdumtest, amesdumtest$Sale_Price,1)
> dim(las$coef1se)
[1] 19  1
> dim(las$coef)
[1] 65  1
```

Ames Data: basic Lasso (cont')

Code R

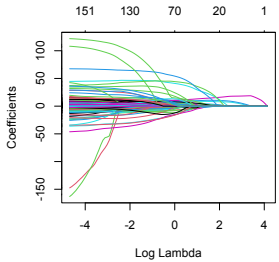
```
> las$coef1se
                                  lambda.1se
(Intercept)                     -1.460817e+02
Lot_Area                         2.834399e-05
Overall_Qual                     1.630206e+01
Year_Built                       2.367560e-02
Exter_Qual                       7.770446e+00
Bsmt_Qual                        6.936410e-01
Bsmt_Exposure                    1.375491e+00
BsmtFin_Type_1                   6.966487e-01
Total_Bsmt_SF                    1.265963e-02
First_Flr_SF                     5.911217e-03
Gr_Liv_Area                      4.375059e-02
Bsmt_Full_Bath                   1.813604e+00
Kitchen_Qual                     1.013407e+01
Fireplace_Qu                     6.590396e-01
Garage_Finish                    9.700409e-01
Garage_Cars                      1.125247e+00
Garage_Area                      3.040884e-02
Neighborhood_Northridge_Heights  9.034261e+00
Neighborhood_Northridge          5.949373e+00
```

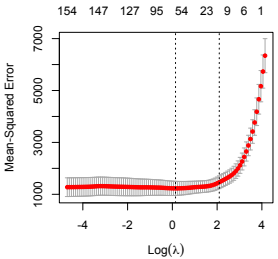
Ames Data: basic Lasso (cont')

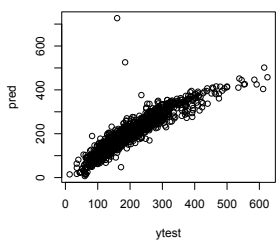
Code R

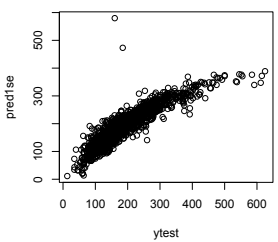
```
> las$coef
                                                lambda.min
(Intercept)                                   -6.164742e+02
Lot_Frontage                                   9.597108e-03
Lot_Area                                       5.098198e-04
Land_Slope                                    -1.284900e+00
Overall_Qual                                   1.108313e+01
Overall_Cond                                   3.096148e+00
Year_Built                                     1.694775e-01
Year_Remod_Add                                 7.601616e-02
Exter_Qual                                     8.598053e+00
Bsmt_Qual                                      5.532150e-01
Bsmt_Exposure                                  4.791563e+00
BsmtFin_Type_1                                 6.056528e-01
Bsmt_Unf_SF                                   -1.042673e-02
Total_Bsmt_SF                                  1.901902e-02
Heating_QC                                     2.572200e-01
Low_Qual_Fin_SF                               -7.094237e-03
Gr_Liv_Area                                    5.348492e-02
Bsmt_Full_Bath                                 3.041631e+00
Full_Bath                                      9.992201e-01
Bedroom_AbvGr                                 -9.732978e-01
Kitchen_AbvGr                                 -1.102846e+01
Kitchen_Qual                                   7.798982e+00
(...)

> las$err
      MAE      MSE MAE_1SE  MSE_1SE
1 18.04351 901.9403 22.2032 1258.575
```

Ames Data: basic Lasso (cont')









Ames Data: basic Lasso (cont')

- We can see that the lasso retains 65 variables (including the intercept).
- When we apply the 1-SE rule, only 19 variables remain.
- However, the MAE and MSE on the 1-SE rule model is larger than the model using the tuning parameter that minimizes the CV error.

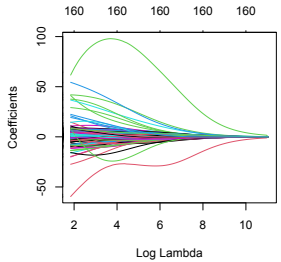
Ames Data: ridge regression

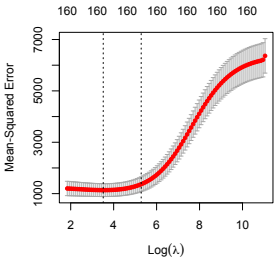
- We can also apply the basic ridge regression with the same function.

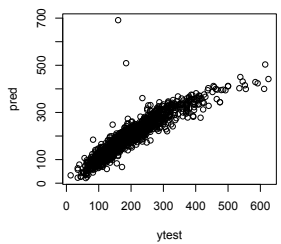
Code R

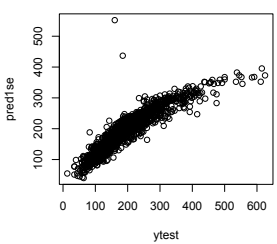
```
# basic ridge
> set.seed(126523)
> rid=wrapglmnet(xdumtrain,amesdumtrain$Sale_Price,xdumtest,amesdumtest$Sale_Price,0)
> dim(rid$coef1se)
[1] 161   1
> dim(rid$coef)
[1] 161   1
> rid$err
       MAE      MSE  MAE_1SE  MSE_1SE
1 17.83458 889.6937 20.49653 1165.339
```

Ames Data: ridge regression (cont')









Ames Data: ridge regression vs lasso

- Both the ridge using the tuning parameter that minimizes the CV error, and the model from the 1-SE rule contain 160 variables (plus the intercept).
- Again, we see that ridge does not perform variable selection.
- The MSE of the ridge estimator is very close to the one of the lasso (for the λ value minimizing the CV error), in this example.

Ames Data: OLS regression

- We can compare the previous results with an OLS regression.

Code R

```
> # Ordinary OLS regression
> lmfit=lm(Sale_Price~.,data=amesdumtrain)
> predlmfit=predict(lmfit,newdata=amesdumtest)
Warning message:
In predict.lm(lmfit, newdata = amesdumtest) :
  prediction from a rank-deficient fit may be misleading
> errlmfit=data.frame(mean(abs(predlmfit-amesdumtest$Sale_Price)),
+                     mean((predlmfit-amesdumtest$Sale_Price)^2))
> names(errlmfit)=c("MAE","MSE")
> errlmfit
       MAE      MSE
1 19.64333 1278.402
```

- We see that OLS does not do as well as the lasso and ridge regression.

Ames Data: other strategies

- We could fit an ordinary OLS to the variables selected by the lasso, or run the lasso again on them.

Code R

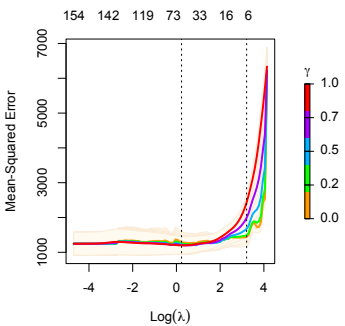
```
> # Fit an OLS with the lasso variables only
> namlas=rownames(las$coef)[-1]
> laslm=lm(Sale_Price~.,data=amesdumtrain[,c(namlas,"Sale_Price")])
> predlaslm=predict(laslm,newdata=amesdumtest)
> errlaslm=data.frame(mean(abs(predlaslm-amesdumtest$Sale_Price)),
+                     mean((predlaslm-amesdumtest$Sale_Price)^2))
> names(errlaslm)=c("MAE","MSE")
> errlaslm
       MAE      MSE
1 18.43752 956.1045

> # Apply the lasso again to the lasso variables only
> set.seed(162534)
> laslas=wrapglmnet(xdumtrain[,namlas],amesdumtrain$Sale_Price,
+ xdumtest[,namlas],amesdumtest$Sale_Price,1)
> laslas$err
      MAE      MSE  MAE_1SE  MSE_1SE
1 18.2734 941.5744 21.81417 1218.449
```

Ames Data: relaxed lasso

- We could use the relaxed lasso using the option `relax=TRUE`. Here, an extra tuning parameter (noted γ) must be estimated.

Code R

```
# relaxed lasso
> set.seed(776245)
> cv.relax=cv.glmnet(x=xtrain,y=ytrain, alpha=1,relax=TRUE)
> plot(cv.relax)
```



Ames Data: relaxed lasso (cont')

Code R

```
# predictions with optimal lambda and gamma
> pred=predict(cv.relax,new=xdumtest,s="lambda.min",gamma="gamma.min")
> pred1se=predict(cv.relax,new=xdumtest,s="lambda.1se",gamma="gamma.1se")
> errrla=data.frame(MAE=mean(abs(pred-amesdumtest$Sale_Price)),
       MSE=mean((pred-amesdumtest$Sale_Price)^2),
       MAE_1SE=mean(abs(pred1se-amesdumtest$Sale_Price)),
       MSE_1SE=mean((pred1se-amesdumtest$Sale_Price)^2))
> errrla
       MAE      MSE  MAE_1SE MSE_1SE
1 18.12304 904.1166 23.57203 1261.42
```

Ames Data: relaxed elastic net

- Finally, we could perform relaxed elastic net, but we need to fix the value of α, which is not optimized in `glmnet` using CV.
- We are not going to do it here.

Ames Data: summary of results

Code R

```
> # Putting all the results together and sorting them according to the MAE and MSE
> allres=rbind(las$err[,1:2],rid$err[,1:2],errlmfit,errlaslm,laslas$err[,1:2],errrla[,1:2])
> row.names(allres)=c("lasso","ridge","OLS","lasso-OLS","lasso-lasso","relaxed lasso")
> allres[order(allres[,1]),]
                   MAE       MSE
ridge          17.83458  889.6937
lasso          18.04351  901.9403
relaxed lasso  18.12304  904.1166
lasso-lasso    18.27340  941.5744
lasso-OLS      18.43752  956.1045
OLS            19.64333 1278.4017
> allres[order(allres[,2]),]
                   MAE       MSE
ridge          17.83458  889.6937
lasso          18.04351  901.9403
relaxed lasso  18.12304  904.1166
lasso-lasso    18.27340  941.5744
lasso-OLS      18.43752  956.1045
OLS            19.64333 1278.4017
```

- We see that ridge regression performed the best according to the MAE and MSE.

Ames Data: summary of results (cont')

- When we look more closely at the predictions, we realize that some models produce a few negative predictions, which are impossible values for the response Sale Price.
- Since these are rare occurrences, we might still use such a model and modify the negative predictions (e.g. set them to 0).
- Alternatively, we might want to use log(Sale Price) as the response instead. Maybe the predictive performance would improve.
- However, when comparing the performance of models that use different transformations of the response Y (i.e. some use Y, others $\log(Y)$ etc.), we must remember to do it on the same scale and, preferably, on the one that will be used in practice.

Agenda

Introduction

1. Introduction
2. Classical variable selection methods
3. Ridge regression
4. Lasso and other methods
5. An example: Ames data
6. **Grouped data**
7. Using interactions
8. Sure dependence screening (SIS)
9. Generalized linear models
10. An example: German credit data
11. Other applications of regularization

Grouped variables

- Sometimes, covariates are grouped together. The most common situation is when a categorical covariate is replaced by a set of dummy variables.
- In that case, it may be interesting to shrink and select these variables as a group.
- Some methods, like the grouped lasso (Yuan, 2006), will strictly enforce group selection, that is, either all variables in a group are selected, or they are all dropped.
- Other method, like the group exponential lasso (Breheny, 2015), tries to select important groups and important variables within groups.

Grouped Lasso

- Assume that the p covariates are partitioned into L groups $G_1, G_2, \ldots, G_L$ with sizes $p_1, p_2, \ldots, p_L$.
- The grouped lasso estimates minimizes

$$\sum_{i=1}^{n}(Y_i - (\beta_0 + \beta_1 X_{1i} + \beta_2 X_{2i} + \cdots + \beta_p X_{pi}))^2 + \lambda \sum_{l=1}^{L} \sqrt{p_l} \left(\sum_{j \in G_l} \beta_j^2 \right)^{1/2}.$$

- Note that if each variable is a group ($L = p$), then we fall back to the lasso.
- These methods and others are available in the package `grpreg`.

Grouped Lasso: small example

- Let's illustrate the difference between the grouped Lasso and group exponential Lasso with a simple example.
- We generate data with two covariates X_1 and X_2.
- The first one is continuous and the second one is categorical with 4 possible values, 1, 2, 3, and 4.
- We create 3 dummy variables X_{21}, X_{22}, X_{23} to represent the first three values and the last one is the reference category.
- Only X_1 and X_{21} are linked to Y, the model being

$$E[Y] = 2X_1 + X_{21}$$

Grouped Lasso: small example (cont')

- We start by fitting an ordinary linear regression model.

Code R

```
> set.seed(36566)
> x1=runif(200)
> x2=sample(1:4,200,replace=TRUE)
> x21=as.numeric(x2==1)
> x22=as.numeric(x2==2)
> x23=as.numeric(x2==3)
> y=2*x1+x21+rnorm(200)
> matx=cbind(x1,x21,x22,x23)
> summary(lm(y~x1+x21+x22+x23))

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -0.15667    0.18378  -0.852    0.395
x1           2.26017    0.23106   9.782  < 2e-16 ***
x21          0.91915    0.19077   4.818 2.91e-06 ***
x22          0.11789    0.19442   0.606    0.545
x23          0.09667    0.19726   0.490    0.625
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.9661 on 195 degrees of freedom
Multiple R-squared:  0.3876,Adjusted R-squared:  0.3751
F-statistic: 30.86 on 4 and 195 DF,  p-value: < 2.2e-16
```

- As expected, we see that the estimated parameters for X_1 and X_{21} are significant but not the parameters for the other two dummies.

Grouped Lasso: small example (cont')

- Now, let's fit a group-Lasso
- The `group` vector is used to identify the groups of variables.
- Here there are 2 groups: X_1 and $\{X_{21}, X_{22}, X_{23}\}$.

Code R

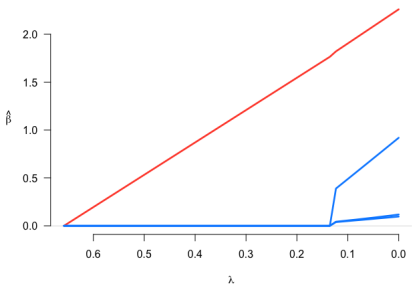
```
> library(grpreg)
> group=c(1,2,2,2)
> grlassofit=grpreg(matx, y, group, penalty="grLasso")
```

Grouped Lasso: small example (cont')

- Let's plot the results...

Code R

```
> plot(grlassofit)
```



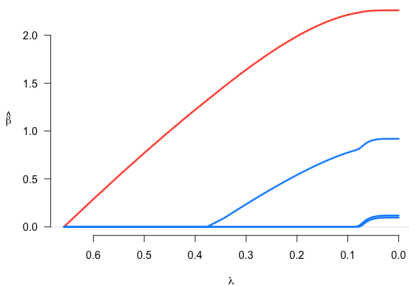
- We see that the three dummies (blue lines) are either all in the model at first and they are dropped simultaneously at some point. Note that the value $\lambda = 0$ corresponds to ordinary OLS.

Grouped Lasso: small example (cont')

- Now let's fit the group exponential lasso model, and plot the results.

Code R

```
> gelfit=grpreg(matx, y, group, penalty="gel")
> plot(gelfit)
```



- This time, the two unrelated dummies X_{22}, X_{23} are dropped early but X_{21} continues to have a non-zero coefficient for some time after that.

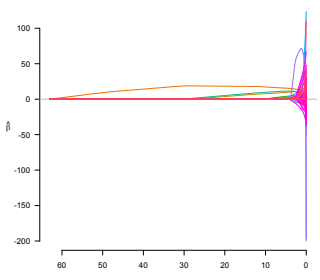
Ames Data Example

- We will use the group lasso and the exponential lasso with the Ames data, such that all dummy variables from a single categorical variable are treated as a group.

Ames Data Example: group Lasso

Code R

```
# the vector group will be the argument to grpreg
> group=c(1:57,rep(58,11),rep(59,3),60, rep(61,2),rep(62,4), rep(63,21),
   rep(64,5),65,rep(66,4), rep(67,5), rep(68,2), 69, rep(70,10),rep(71,10),
   rep(72,3),rep(73,4),74,75,rep(76,5),rep(77,2),rep(78,3),rep(79,4))

> library(grpreg)
>
> set.seed(14273)
> grlassofit=grpreg(xdumtrain, amesdumtrain$Sale_Price, group, penalty="grLasso")
> plot(grlassofit)
```



Ames Data Example: group Lasso

Code R

```
> # Group lasso
> grlassofitcv=cv.grpreg(xdumtrain, amesdumtrain$Sale_Price, group,seed=474659,penalty="grLasso")
> coefgrlasso=predict(grlassofitcv,type="coefficients")
> predgrlasso=predict(grlassofitcv,X=xdumtest)
> errgrlasso=data.frame(mean(abs(predgrlasso-amesdumtest$Sale_Price)),
+                       mean((predgrlasso-amesdumtest$Sale_Price)^2))
> names(errgrlasso)=c("MAE","MSE")
> row.names(errgrlasso)=c("group lasso")
> errgrlasso
                MAE      MSE
group lasso 17.90152 1007.482

> allres=rbind(allres,errgrlasso)
```

Ames Data Example: exponential group Lasso

- Now let's apply the group exponential Lasso ...

Code R

```
# Exponential lasso
> set.seed(73888)
> gelcv=cv.grpreg(xdumtrain, amesdumtrain$Sale_Price, group, seed=474659,penalty="gel")
> coefgel=predict(gelcv,type="coefficients")
> predgel=predict(gelcv,X=xdumtest)
> errgel=data.frame(mean(abs(predgel-amesdumtest$Sale_Price)),
                    mean((predgel-amesdumtest$Sale_Price)^2))
> names(errgel)=c("MAE","MSE")
> row.names(errgel)=c("exponential lasso")
> errgel
                       MAE      MSE
exponential lasso 18.9298 1246.616

> allres=rbind(allres,errgel)
```

Ames Data: summary of results

Code R

```
> # Best methods so far
> allres[order(allres[,1]),]
                         MAE       MSE
ridge                17.83458  889.6937
group lasso          17.90152 1007.4815
group lasso1         17.90152 1007.4815
lasso                18.04351  901.9403
relaxed lasso        18.12304  904.1166
lasso-lasso          18.27340  941.5744
lasso-OLS            18.43752  956.1045
exponential lasso    18.92980 1246.6156
OLS                  19.64333 1278.4017


> allres[order(allres[,2]),]
                         MAE       MSE
ridge                17.83458  889.6937
lasso                18.04351  901.9403
relaxed lasso        18.12304  904.1166
lasso-lasso          18.27340  941.5744
lasso-OLS            18.43752  956.1045
group lasso          17.90152 1007.4815
group lasso1         17.90152 1007.4815
exponential lasso    18.92980 1246.6156
OLS                  19.64333 1278.4017
```

- For predictive purposes, these methods do not generally have advantages over ones that treat the dummies as individual variables regardless of the original covariates.

Agenda

Introduction

1. Introduction
2. Classical variable selection methods
3. Ridge regression
4. Lasso and other methods
5. An example: Ames data
6. Grouped data
7. **Using interactions**
8. Sure dependence screening (SIS)
9. Generalized linear models
10. An example: German credit data
11. Other applications of regularization

Interactions

- To get a more flexible model, one possibility is to add transformations of the covariates to the model.
- One common example is to use the square of continuous covariates and interactions between covariates.
- It is rather straightforward to do....
- However, there are some reasons to enforce the hierarchy in the variables.
- Bien (2013) has a nice discussion about the concept of hierarchy....
- It means that if an interaction between variables is present in the model, than all lower order interactions between these variables should also be present.

Interactions and hierarchy

- We will focus on two-way interactions of the form $X_i \times X_j$ in this section.
- Note that a square $X_i^2 = X_i \times X_i$ can be seen as an interaction between X_i and itself.
- Strong hierarchy means that if $X_i \times X_j$ is in the model, then both X_i and X_j must also be in the model.
- Weak hierarchy means that if $X_i \times X_j$ is in the model, then at least one of X_i and X_j must also be in the model.

Why should we enforce strong hierarchy ?

- **Many statisticians think that we should always enforce strong hierarchy, unless the specific theory dictates something else.**
- To understand why, consider now the following model:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_{12} X_1 X_2$$

- The effect of X_2 is $\beta_2 + \beta_{12} X_1$
- If we set $\beta_2 = 0$ (i.e.removing X_2 from the model), then the effect of X_2 becomes $\beta_{12} X_1$.
- Hence, we are imposing that the effect of X_2 is 0 when $X_1 = 0$.
- Again, there is generally no justification to assume something like this. The same argument can be made for X_1.
- Hence, both individual variables should be included when an interaction between them is included. This is the strong hierarchy principle.

Comment

- The same argument is valid to never remove the intercept from the model
- To understand why, assume first a simple model

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2.$$

- If we don't include the intercept in the model i.e. we suppose $\beta_0 = 0$, we are imposing that the mean of Y is 0 when $X_1 = 0$ and $X_2 = 0$.
- Unless we have theoretical reasons to think this is true, there is not justification for setting $\beta_0 = 0$ right from the start.
- This is why we always leave the intercept in the model, even if it's not statistically significant.

Strong hierarchy and variable selection

- If we apply any method that performs variable selection, e.g. lasso, stepwise, among all main effects and two-way interactions, we may end up with a set of variables that do not satisfy any of the hierarchy principles.
- However, it is not clear if enforcing the hierarchy principle (weak or strong) is preferable for predictive purposes.
- Bien (2013) proposes an adaptation of the lasso that can enforce weak or strong hierarchy.
- It is implemented in the package `hierNet`.
- There is also another method developed by Lim (2015) under this settings, implemented in a package called `glinternet`.

Strong hierarchy and variable selection

- If methods or implementations dealing with hierarchy are not available, there is an easy way to enforce it.
- We can just run any variable selection method on all variables (main effects and interactions), see what variables are kept, and then add manually the variables to have a model with the desired hierarchy.
- For example, the variable selection method selected $X_1 * X_2$ and X_2, but not X_1. Then we can just add X_1 manually after that.
- But once again, if the main goal is prediction, then it is not clear if a model following the hierarchy principle will be better....

Agenda

Introduction

1. Introduction
2. Classical variable selection methods
3. Ridge regression
4. Lasso and other methods
5. An example: Ames data
6. Grouped data
7. Using interactions
8. **Sure dependence screening (SIS)**
9. Generalized linear models
10. An example: German credit data
11. Other applications of regularization

Sure Independence Screening (SIS)

- Sometimes, the number of predictors p is a lot larger than the number of observation n.
- This is often labeled the $p >> n$ case.
- In this case, the performance of the variable selection methods can be adversely affected and the computational cost to apply some of them may be too high.
- A new area of research appeared in 2008 with the important paper by Fan (2008).
- The idea is to develop a screening method that can quickly reduce the number of predictors to a more manageable size (usually below the sample size) without losing any important variable.
- Then another variable selection method, like the lasso, can be used on the remaining variables.
- They introduced the method "Sure Independence Screening" (SIS) to achieve the screening part.

Sure Independence Screening (SIS)

- The idea is quite simple and is based on correlation learning.
- Basically, the correlation between each predictor and the response Y is computed.
- Then they are ranked according to the magnitude (in absolute value) of the correlation.
- One recommends to keep the top $n - 1$ or top $n/\log(n)$ variables with highest correlation.
- Fan (2008) show that, under some conditions, including sparsity, the probability that all true variables (the ones related to Y) will be selected by this method is high.
- This method is implemented in the package `SIS`
- Screening methods have now been proposed for generalized linear regression (Fan, 2010) and survival data (Zhao, 2012). Robust versions have also been proposed (Li, 2012).

Agenda

Introduction

1. Introduction
2. Classical variable selection methods
3. Ridge regression
4. Lasso and other methods
5. An example: Ames data
6. Grouped data
7. Using interactions
8. Sure dependence screening (SIS)
9. **Generalized linear models**
10. An example: German credit data
11. Other applications of regularization

The case of GLM's

- The methods presented so far were for the case with a continuous target Y and used the least squares as the basic criterion.
- Other situations are very common, e.g having a binary Y, or a Y that represents a count variable (the number of occurrences of an event in a given time interval).
- The generalized linear models family encompasses these cases and many more, including Cox models for censored data.

The case of GLM's (cont')

- The least squares criterion is equivalent to using the likelihood method assuming that the error term is normally distributed.
- Hence, the elastic net presented earlier could have equivalently been written as the following minimization problem:

$$\min_{\beta_0,\ldots,\beta_p} \frac{1}{n} \quad \sum_{i=1}^{n} w_i l(Y_i, \beta_0 + \beta_1 X_{1i} + \ldots, +\beta_p X_{pi}) \\ +\lambda \left[(1-\alpha)/2 \sum_{j=1}^{p} (\beta_j)^2 + \alpha \sum_{j=1}^{p} |\beta_j| \right],$$

  where $l(Y_i, \beta_0 + \beta_1 X_{1i} + \ldots, +\beta_p X_{pi})$ is the negative log-likelihood contribution for observation i. The w_i's are observation weights that are all set to 1 by default.
- Indeed, when the error is assumed normal, then this contribution is $\frac{1}{2}(Y_i - (\beta_0 + \beta_1 X_{1i} + \ldots, +\beta_p X_{pi}))^2$.
- This is the problem solved by `glmnet` in general.

The case of GLM's (cont')

- `glmnet` fits generalized linear models with a penalized maximum likelihood criterion, including Gaussian (ordinary linear regression), logistic, Poisson, and Cox regression.
- We'll see an example in the next section with the German credit data.

Agenda

Introduction

1. Introduction
2. Classical variable selection methods
3. Ridge regression
4. Lasso and other methods
5. An example: Ames data
6. Grouped data
7. Using interactions
8. Sure dependence screening (SIS)
9. Generalized linear models
10. **An example: German credit data**
11. Other applications of regularization

German Credit Data

- The German credit data set (Lichman, 2013) is another data set that we will use a lot.
- The data and description can be found here:

  `https://archive.ics.uci.edu/ml/support/statlog+(german+credit+data)`

- This data set classifies 1000 people described by a set of 20 attributes as good or bad credit risks.
- The target variable, V21, is binary and is recoded to 0-1 (1-2 in the original data); 0=good risk and 1=bad risk.
- We work with a first version of the data set that includes 9 numeric covariates, 11 factor covariates, and the target variable.
- The goal of this example is to show you how to apply a logistic regression Lasso model

German Credit Data: data preprocessing

- As we did for the Ames data set, we first prepare the data.

Code R

```
 ################################
# Prepare the German Credit data set.

# This version of the data uses the factor variables directly
# There are 20 covariates and 1 target "V21": 11 factor covariates,
#     9 numeric covariates (including 3 binary), and 1 binary target

> gercred=read.table(paste("german.data",sep=""))

# recode the target and two binary covariates to 0-1
> gercred$V21=as.numeric(gercred$V21==2)
> gercred$V19=as.numeric(gercred$V19=="A192")
> gercred$V20=as.numeric(gercred$V20=="A201")

# get the names of the factor variables
> fac_vars=vapply(gercred, is.factor, logical(1))
> namfac=names(fac_vars)[fac_vars]

# names of the numeric variables
> num_vars=vapply(gercred, is.numeric, logical(1))
> namnum=names(num_vars)[num_vars]
```

German Credit Data: data preprocessing (cont')

Code R

```
> summary(gercred)

   V1           V2          V3           V4           V5           V6          V7           V8
 A11:274   Min.   : 4.0   A30: 40   A43    :280   Min.   :  250   A61:603   A71: 62   Min.   :1.000
 A12:269   1st Qu.:12.0   A31: 49   A40    :234   1st Qu.: 1366   A62:103   A72:172   1st Qu.:2.000
 A13: 63   Median :18.0   A32:530   A42    :181   Median : 2320   A63: 63   A73:339   Median :3.000
 A14:394   Mean   :20.9   A33: 88   A41    :103   Mean   : 3271   A64: 48   A74:174   Mean   :2.973
           3rd Qu.:24.0   A34:293   A49    : 97   3rd Qu.: 3972   A65:183   A75:253   3rd Qu.:4.000
           Max.   :72.0             A46    : 50   Max.   :18424                       Max.   :4.000
                                    (Other): 55
   V9          V10          V11          V12          V13          V14          V15          V16
 A91: 50   A101:907   Min.   :1.000   A121:282   Min.   :19.00   A141:139   A151:179   Min.   :1.000
 A92:310   A102: 41   1st Qu.:2.000   A122:232   1st Qu.:27.00   A142: 47   A152:713   1st Qu.:1.000
 A93:548   A103: 52   Median :3.000   A123:332   Median :33.00   A143:814   A153:108   Median :1.000
 A94: 92              Mean   :2.845   A124:154   Mean   :35.55                         Mean   :1.407
                      3rd Qu.:4.000              3rd Qu.:42.00                         3rd Qu.:2.000
                      Max.   :4.000              Max.   :75.00                         Max.   :4.000

   V17          V18          V19          V20          V21
 A171: 22   Min.   :1.000   Min.   :0.000   Min.   :0.000   Min.   :0.0
 A172:200   1st Qu.:1.000   1st Qu.:0.000   1st Qu.:1.000   1st Qu.:0.0
 A173:630   Median :1.000   Median :0.000   Median :1.000   Median :0.0
 A174:148   Mean   :1.155   Mean   :0.404   Mean   :0.963   Mean   :0.3
            3rd Qu.:1.000   3rd Qu.:1.000   3rd Qu.:1.000   3rd Qu.:1.0
            Max.   :2.000   Max.   :1.000   Max.   :1.000   Max.   :1.0
```

German Credit Data: data preprocessing (cont')

- Here is another version with dummies, replacing the factor covariates.

Code R

```
> library(fastDummies)
> gercreddum=dummy_cols(gercred, remove_first_dummy=TRUE, remove_selected_columns=TRUE)

# Now all variables are numeric.
# There are 48 covariates and 1 binary target "V21".

> summary(gercreddum)
       V2              V5              V8             V11             V13             V16
 Min.   : 4.0   Min.   :  250   Min.   :1.000   Min.   :1.000   Min.   :19.00   Min.   :1.000
 1st Qu.:12.0   1st Qu.: 1366   1st Qu.:2.000   1st Qu.:2.000   1st Qu.:27.00   1st Qu.:1.000
 Median :18.0   Median : 2320   Median :3.000   Median :3.000   Median :33.00   Median :1.000
 Mean   :20.9   Mean   : 3271   Mean   :2.973   Mean   :2.845   Mean   :35.55   Mean   :1.407
 3rd Qu.:24.0   3rd Qu.: 3972   3rd Qu.:4.000   3rd Qu.:4.000   3rd Qu.:42.00   3rd Qu.:2.000
 Max.   :72.0   Max.   :18424   Max.   :4.000   Max.   :4.000   Max.   :75.00   Max.   :4.000
      V18             V19             V20             V21           V1_A12          V1_A13
 Min.   :1.000   Min.   :0.000   Min.   :0.000   Min.   :0.0   Min.   :0.000   Min.   :0.000
 1st Qu.:1.000   1st Qu.:0.000   1st Qu.:1.000   1st Qu.:0.0   1st Qu.:0.000   1st Qu.:0.000
 Median :1.000   Median :0.000   Median :1.000   Median :0.0   Median :0.000   Median :0.000
 Mean   :1.155   Mean   :0.404   Mean   :0.963   Mean   :0.3   Mean   :0.269   Mean   :0.063
 3rd Qu.:1.000   3rd Qu.:1.000   3rd Qu.:1.000   3rd Qu.:1.0   3rd Qu.:1.000   3rd Qu.:0.000
 Max.   :2.000   Max.   :1.000   Max.   :1.000   Max.   :1.0   Max.   :1.000   Max.   :1.000
(...)
```

German Credit Data: data preprocessing (cont')

- For the example, we create a training data set of size 600 and a test set of new data of size 400.

Code R

```
# Splitting the data into a training (ntrain=600) and a
#  test (ntest=400) set

> set.seed(364565)
> ntrain=600
> ntest=nrow(gercred)-ntrain
> indtrain=sample(1:nrow(gercred),ntrain,replace=FALSE)

> xdum=gercreddum
> xdum$V21=NULL
> xdum=as.matrix(xdum)

> gercredtrain=gercred[indtrain,]
> gercredtest=gercred[-indtrain,]
> gercreddumtrain=gercreddum[indtrain,]
> gercreddumtest=gercreddum[-indtrain,]
> gerxdumtrain=xdum[indtrain,]
> gerxdumtest=xdum[-indtrain,]
```

German Credit Data: logistic regression Lasso

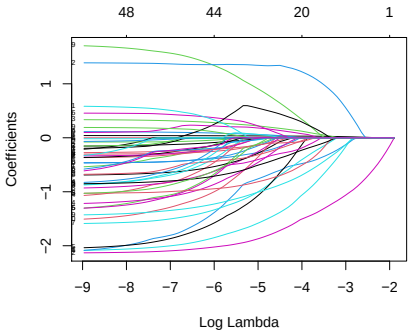
- We can fit a logistic regression model with the lasso with `glmnet`.

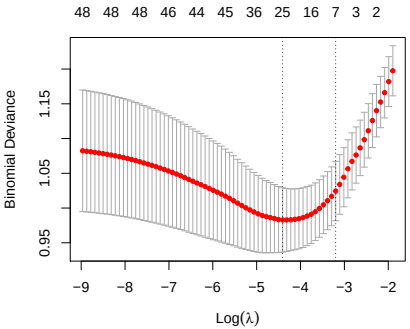
Code R

```
> set.seed(162738)
> library(glmnet)
> plot(glmnet(gerxdumtrain, gercredtrain$V21, family="binomial",alpha=1),
+      xvar = "lambda", label = TRUE)
> cvgerlasso=cv.glmnet(gerxdumtrain, gercredtrain$V21, family="binomial",
+                      alpha=1)
> plot(cvgerlasso)
> coeflassoger=predict(cvgerlasso,new=gerxdumtest,s="lambda.min",
+                      type="coefficients")
> length(coeflassoger[coeflassoger[,1] != 0 ,])
[1] 26
> predlassoger=predict(cvgerlasso,new=gerxdumtest,s="lambda.min",
+                      type="response")
> predlassoger[1:10]
 [1] 0.14378885 0.14396036 0.06977683 0.37842756 0.35945105 0.50497403 0.46602743 0.43738353
 [9] 0.31482076 0.01000160
```

- The lasso keeps 26 covariates (plus the intercept) out of the 48.

German Credit Data: logistic regression Lasso (cont')





Predictions

- We have estimated probabilities $\hat{p}$ for the test set.
- But we need 0-1 predictions $\hat{y}$.
- We have to select a threshold c to define a classification rule.
- The rule will be

$$\text{if } \hat{p} > c \quad \text{then } \hat{y} = 1 \quad ; \quad \text{if } \hat{p} \leq c \quad \text{then } \hat{y} = 0.$$

- In many applications, selecting $c = .5$ is reasonable, especially, if the target variable is nearly balanced (about as much 0's as 1's) and if the consequences of making classification errors are similar.

Predictions (cont')

- More precisely, with a binary target, the different outcomes of the predictions are the following:

Prediction ($\hat{y}$)	True value (y) 0	1
0	good	error
1	error	good

- Here, we will work with a gain matrix.

Finding the optimal cutoff using the gain matrix

- If the gain of both decisions are the same, then the gain matrix is:

	True value (y)	
Prediction ($\hat{y}$)	0	1
0	1	0
1	0	1

- It means that we receive a gain of 1 unit if we correctly predict a 0, or if we correctly predict a 1. We receive nothing if we make an error.
- In this case, maximizing the average gain amounts to maximizing

$$\begin{aligned} & P(\hat{y}=0, y=0) \times 1 + P(\hat{y}=1, y=0) \times 0 \\ + \quad & P(\hat{y}=0, y=1) \times 0 + P(\hat{y}=1, y=1) \times 1 \\ = \quad & P(\hat{y}=y), \end{aligned}$$

  which is the good classification rate.

- Thus, when we select the best model as the one maximizing the good classification rate, we implicitly work with the above gain matrix.

Finding the optimal cutoff using the gain matrix (cont')

- In some circumstances however, the consequences of the different errors and good decisions are not symmetric.
- In general, we can specify a gain matrix like this:

Prediction ($\hat{y}$)	True value (y) 0	1
0	g_{00}	g_{01}
1	g_{10}	g_{11}

- The goal would again be to maximize the average gain, which is:

$$\begin{aligned} & P(\hat{y}=0, y=0) \times g_{00} + P(\hat{y}=1, y=0) \times g_{10} \\ + \quad & P(\hat{y}=0, y=1) \times g_{01} + P(\hat{y}=1, y=1) \times g_{11}. \end{aligned}$$

Finding the optimal cutoff using the gain matrix (cont')

- This function finds the best threshold c, by estimating the average gain.

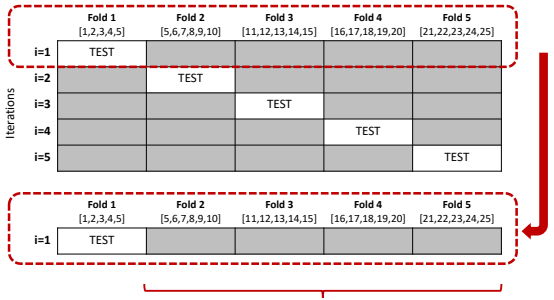
Code R

```
bestcutp=function(predcv,y,gainmat=diag(2),cutp=seq(0,1,.02),plotit=FALSE)
{
# predcv = vector of predicted probabilities (obtained out-of-sample by CV for example)
# y = vector of target (0-1)
# gainmat = gain matrix (2X2)  (we want to maximize the gain)
# (1,1) = gain if pred=0 and true=0      (1,2) = gain if pred=0 and true=1
# (2,1) = gain if pred=1 and true=0      (2,2) = gain if pred=1 and true=1
# cutp=vector of thresholds to try
# plotit=plot or not the results

# value: a list with
# 1) matrix giving the thresholds and estimated mean gains
# 2) the threshold with maximum gain, with the associated mean gain

   nc=length(cutp) ;  gain=rep(0,nc)
   for(i in 1:nc)
   {
      pred=as.numeric(predcv>cutp[i])
      gain[i]=mean(gainmat[1,1]*(pred==0)*(y==0)+gainmat[1,2]*(pred==0)*(y==1)+
      gainmat[2,1]*(pred==1)*(y==0)+gainmat[2,2]*(pred==1)*(y==1))
   }
   if(plotit){plot(cutp,gain,type="l",xlab="threshold",ylab="gain")}
   out=list(NULL,NULL)
   out[[1]]=cbind(cutp,gain)
   out[[2]]=out[[1]][which.max(gain),]
   out
}
```

Finding the optimal cutoff using the gain matrix (cont')

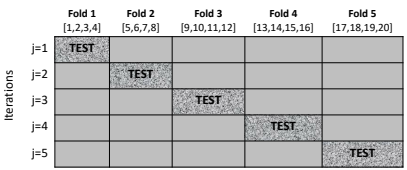
- In order to use the previous function, we need estimated probabilities for the training data.
- It would not be right to use directly the predicted probabilities that we got from a call to `predict`, because these are in-sample estimations computed using the same data that built the model.
- The following function computes estimated probabilities by cross-validation with `glmnet`.
- Note that for each CV fold, the tuning parameter in `glmnet` is also chosen by CV.
- Hence, there are two levels of cross-validation:
  - The outer CV is used to compute estimated probabilities
  - The inner CV estimates the tuning parameter for a given fold of the outer CV.

Illustration with a simple example



Outer cross validation:

- At each iteration (*i=1,...,5*), we train the glmnet model on 4 folds using the optimal λ hyperparameter
- The optimal λ hyperparameter is different for each iteration *i*
- The predictions $\hat{p}$ are obtained on the test folds



Inner cross validation:

- The CV is used to obtain the optimal hyperparameters of the glmnet model at iteration *i* of the outer CV
- At each iteration *j* (*j=1,...,5*), we train a glmnet over a grid of λ hyperparameters
- The MSE for each value of λ is computed on the test folds and the best λ is chosen at the end

Finding the optimal cutoff using the gain matrix (cont')

Code R

```
# function to get cross-validated estimated probabilities from glmnet
# the best lambda is chosen by CV in each fold
# the output can be used to find the best threshold afterwards

predcvglmnet=function(xtrain,ytrain,k=10,alpha=1)
{
# xtrain=matrix of predictors
# ytrain=vector of target (0-1)
# k= # folds in CV
# alpha=alpha parameter in glmnet

# value: the CV predicted probabilities

   library(glmnet)
   set.seed(375869)
   n=nrow(xtrain)
   pred=rep(0,n)
   per=sample(n,replace=FALSE)
   tl=1
   for(i in 1:k)
   {
      tu=min(floor(tl+n/k-1),n)
      if(i==k){tu=n}
      cind=per[tl:tu]
      fit=cv.glmnet(xtrain[-cind,],ytrain[-cind], family="binomial",alpha=alpha)
      pred[cind]=predict(fit,new=xtrain[cind,],s="lambda.min",type="response")
      tl=tu+1
   }
   pred
}
```

Finding the optimal cutoff using the gain matrix (cont')

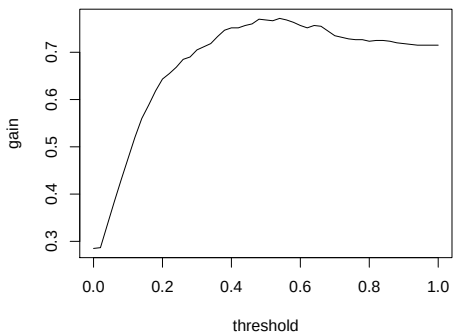
- We can now use both functions to get the optimal threshold.
- We first use the identity gain matrix (i.e. the criterion is the good classification rate).

Code R

```
# computing CV estimated probabilities with glmnet
> set.seed(16274)
> pred=predcvglmnet(gerxdumtrain,gercredtrain$V21,k=10,alpha=1)
>
> # estimating the best threshold with the identity gain matrix
> res=bestcutp(pred,gercredtrain$V21,gainmat=diag(2),cutp=seq(0,1,.02),plotit=TRUE)
> res[[2]]
     cutp      gain
0.5400000 0.7716667
```

Finding the optimal cutoff using the gain matrix (cont')

- The best threshold is $c = 0.54$, with an estimated good classification rate of 0.77.



Computing the good classification rate

Code R

```
# using the best threshold to obtain the predictions
> predlassoger01=as.numeric(predlassoger>res[[2]][1])

# good classification rate on the test set
> mean(gercredtest$V21==predlassoger01)
[1] 0.6975

# a naive rule would get a good classification rate of
> max(mean(gercredtest$V21),1-mean(gercredtest$V21))
[1] 0.6775
```

- We obtain a true good classification rate of 0.6975 on the test set, and a naive rule, assigning everyone to the 0 class would get a good classification rate of 0.677.
- Hence, the lasso logistic regression performs only a little better compared to the naive rule.

Computing AUC and ROC curves

- The package `ROCR` provides numerous measures of performances for binary classifiers, with the possibility of plotting them.
- Here is how to get a ROC curve, the corresponding AUC (Area Under the Curve), and one version of a lift chart (here the cumulative gain chart), for the test data.

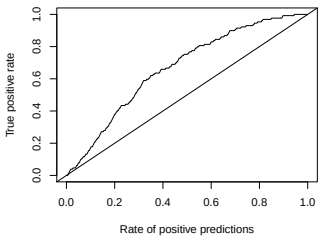
Computing AUC and ROC curves (cont')

Code R

```
> predrocr=prediction(predlassoger,gercredtest$V21)
> roc=performance(predrocr,"tpr","fpr")
> plot(roc)
> abline(a=0,b=1)
> performance(predrocr,"auc")
An object of class "performance"
Slot "x.name":
[1] "None"

Slot "y.name":
[1] "Area under the ROC curve"

Slot "alpha.name":
[1] "none"

Slot "x.values":
list()

Slot "y.values":
[[1]]
[1] 0.746074
```

- The AUC is 0.746.

Computing AUC and ROC curves (cont')



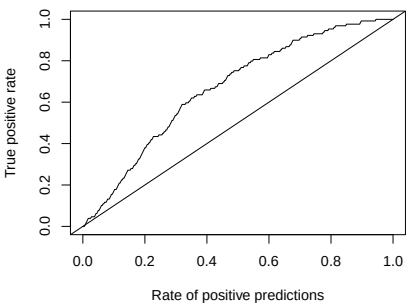
- Note that we used the test data here but in practice, the value of Y would not be available for the new data.
- Hence, we would need to compute the ROC curve, AUC and lift chart with the training data.
- In that case, we must remember to use proper estimation of the probabilities (like the ones obtained by CV above), in order to get honest estimates.

Lift curve

- One can also plot a lift curve

Code R

```
> lift=performance(predrocr,"tpr","rpp")
> plot(lift)
> abline(a=0,b=1)
```



Customizing the gain matrix

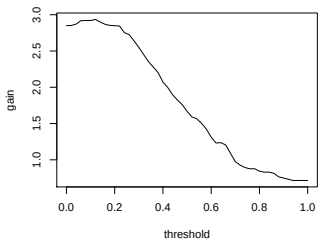
- Instead of using the identity gain matrix, we can assume that it is 10 times more profitable to detect a bad risk, than to detect a good risk.
- The gain matrix is

	True value (y)	
Prediction ($\hat{y}$)	0	1
0	1	0
1	0	10

Customizing the gain matrix (cont')

Code R

```
> # estimating the best threshold if it is 10 times better to detect a bad risk than a good one
> set.seed(18965)
> res1=bestcutp(pred,gercredtrain$V21,plotit=TRUE, gainmat=rbind(c(1,0),c(0,10)))
> res1[[2]]
 cutp  gain
0.120 2.935
```



- In that case, the best threshold is 0.12. It is lower than 0.5 since we now want to classify more people as bad risks because the reward is higher if we are right.

AUC as a Concordance Index

- There is another interpretation to the AUC.
- It can be seen as a concordance measure, called C-index, concordance index or Harrell's index.
- Let y_i, be a binary (0-1) response and $\hat{p}_i$ be estimations of $P(Y_i = 1)$ from a model, $i = 1, 2, \ldots, n$.
- Consider all pairs of observations (i, j).
- A pair is called valid if $y_i \neq y_j$.
- Among the valid pairs, a pair is called:
  - concordant if $(y_i > y_j$ and $\hat{p}_i > \hat{p}_j)$ or if $(y_i < y_j$ and $\hat{p}_i < \hat{p}_j)$.

    (i.e. the predicted probs and the true observations are in the same order)
  - discordant if $(y_i > y_j$ and $\hat{p}_i < \hat{p}_j)$ or if $(y_i < y_j$ and $\hat{p}_i > \hat{p}_j)$.

    (i.e. the predicted probs and the true observations are in the reverse order)
  - tied if $\hat{p}_i = \hat{p}_j$.

AUC as a Concordance Index (cont')

- To compute the C-index, we let a concordant pair receive a weight of 1, a discordant pair receive a weight of 0, and a tied pair receive a weight of 0.5.
- The C-index is the sum of the weights over all valid pairs divided by the number of valid pairs.

  i.e. it is the proportion of concordant pairs.
- This type of measure if often used with survival data and we will come back to it later in the chapter on this subject.
- **We'll see that we get the same value as the AUC.**

AUC as a Concordance Index (cont')

- Here is a simple function to compute the C-index, and an example of use to compute the C-index with the previous example.

Code R

```
 cindexbasic=function(phat,y)
{
     n=length(phat)
     cc=0
     npair=0
     for(i in 1:(n-1))
     {
         for(j in (i+1):n)
         {
             if(y[i]!=y[j])
             {
                 cc=cc+(phat[i] > phat[j])*(y[i]>y[j])+
                     (phat[i] < phat[j])*(y[i]<y[j])+
                     0.5*(phat[i]==phat[j])
                 npair=npair+1
             }
         }
     }
     cc/npair
 }

> cindexbasic(predlassoger,gercredtest$V21)
[1] 0.746074
```

- **We see that we get the same value as the AUC.**

Agenda

Introduction

1. Introduction
2. Classical variable selection methods
3. Ridge regression
4. Lasso and other methods
5. An example: Ames data
6. Grouped data
7. Using interactions
8. Sure dependence screening (SIS)
9. Generalized linear models
10. An example: German credit data
11. **Other applications of regularization**

Other Applications of Regularization

- Regularization can be useful in a variety of problems.
- For example, weight decay in neural networks is regularization applied to the weights.
- Using L_1 type (lasso) regularization can provide sparse and more interpretable solutions in many contexts.
- One example is principal components analysis (PCA)...

The example of PCA

- PCA is a classical dimension reduction method.
- Assume we have p variables $X_1, \ldots, X_p$. The PCA builds uncorrelated linear combinations (components) of these variables to try to recover a large portion of the variability in the data with just a few components.
- These components take the form

$$\begin{aligned} v_1 &= \alpha_{11}x_1 + \alpha_{12}x_2 + \cdots + \alpha_{1p}x_p \\ v_2 &= \alpha_{21}x_1 + \alpha_{22}x_2 + \cdots + \alpha_{2p}x_p \\ &\cdots \\ v_p &= \alpha_{p1}x_1 + \alpha_{p2}x_2 + \cdots + \alpha_{pp}x_p. \end{aligned}$$

- The vectors $\alpha_i = (\alpha_{i1}, \alpha_{i2}, \ldots, \alpha_{ip})$, $i = 1, 2, \ldots, p$ are the weights (or coefficients) for each component.

The example of PCA

- The first weight vector α_1 is computed such that v_1 has the largest variance, under the constraint that $||\alpha_1||_2^2 = 1$, where $||\cdot||_2$ denotes the L_2 norm, that is $||a||_2 = \left(\sum_{i=1}^{n} a_i^2\right)^{1/2}$.
- A constraint is required because without it, the variance could be made as large as we want by increasing the size of $||\alpha_1||_2^2$.
- The second weight vector, α_2 is computed such that v_2 has the largest variance under the constraints that $||\alpha_2||_2^2 = 1$, and is orthogonal to α_1 (i.e. $\alpha_{11}\alpha_{21} + \alpha_{12}\alpha_{22} + \cdots + \alpha_{1p}\alpha_{2p} = 0$).
- This implies that v_1 and v_2 have a correlation of 0.
- In general, the k^{th} weight vector, α_k is computed such that v_k has the largest variance under the constraints that $||\alpha_k||_2^2 = 1$, and is orthogonal to $\alpha_1, \ldots, \alpha_{k-1}$.

Illustration: PCA

- Here is an example with data used in another course.
- In a survey, 200 adults were asked the following questions regarding a certain type of store.
- The responses are provided on a 5 points Likert scale where:

  1 = not important

  2 = slightly important

  3 = moderately important

  4 = important

  5 = very important

Illustration: PCA (cont')

- **For you, how important is it:**
  - X_1 that the store offers good prices every day?
  - X_2 that the store accepts major credit cards?
  - X_3 that the store offers quality products?
  - X_4 that sellers are familiar with the products?
  - X_5 that there are special sales regularly?
  - X_6 that known brands are available?
  - X_7 that the store has its own credit card?
  - X_8 that the service is fast?
  - X_9 that there is a wide selection of products?
  - X_{10} that the store accepts payment by debit card?
  - X_{11} that the staff is courteous?
  - X_{12} that the store has in stock the advertised products?

Illustration: PCA (cont')

- The data are on a SAS data file.
- We first import and scale the data so that each variable has a mean of 0 and a variance of 1.

Code R

```
# import the data from SAS
> library(haven)
> datpca=read_sas("factor2.sas7bdat")
> datpca=as.matrix(datpca)
> datpca=apply(datpca,2,scale)
```

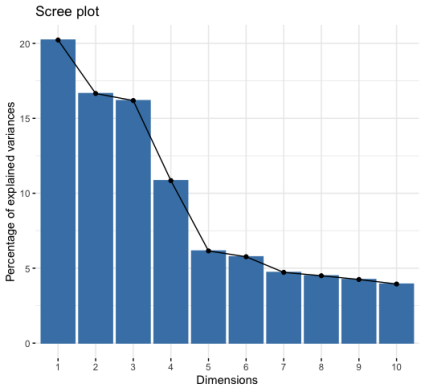
Illustration: PCA (cont')

- Then we can apply ordinary PCA and get a scree plot, using the package `factoextra`.

Code R

```
> pca=prcomp(datpca)
> # visualize the results
> library(factoextra)
Loading required package: ggplot2
Welcome! Want to learn more? See two factoextra-related books at https://goo.gl/ve3WBa
> get_eig(pca)
       eigenvalue variance.percent cumulative.variance.percent
Dim.1   2.4273080        20.227567                    20.22757
Dim.2   1.9988532        16.657110                    36.88468
Dim.3   1.9423538        16.186282                    53.07096
Dim.4   1.3012928        10.844106                    63.91506
Dim.5   0.7383181         6.152651                    70.06772
Dim.6   0.6917446         5.764538                    75.83225
Dim.7   0.5666232         4.721860                    80.55411
Dim.8   0.5396515         4.497096                    85.05121
Dim.9   0.5096297         4.246914                    89.29812
Dim.10  0.4732469         3.943724                    93.24185
Dim.11  0.4551982         3.793318                    97.03517
Dim.12  0.3557801         2.964834                   100.00000
> fviz_eig(pca)
```

Illustration: PCA (cont')



- Three or four components seem reasonable according to the scree plot.
- The first 4 recover 63% of the variance in the data.

Illustration: PCA (cont')

- Here are the coefficients of the first 4 components.

Code R

```
> # getting the coefficient of the first four components
> pca[[2]][,1:4]
           PC1         PC2          PC3         PC4
x1  -0.2080887  0.10438295 -0.072166670  0.64557275
x2   0.1924313 -0.05175073  0.525394507  0.08878296
x3   0.3884026  0.36447898 -0.100384469 -0.03686866
x4   0.3555739 -0.33806182 -0.248294736  0.14474449
x5  -0.1445682  0.05482364 -0.081649632  0.69376903
x6   0.2940860  0.33867561 -0.006820791  0.07672541
x7   0.2302601 -0.14129518  0.505368466  0.15924928
x8   0.3542234 -0.40108752 -0.213356225  0.13304289
x9   0.2796442  0.42116179 -0.100878927  0.06634746
x10  0.2680050 -0.14705494  0.451825758  0.12709504
x11  0.3082952 -0.33815955 -0.345003710  0.03105961
x12  0.3338052  0.35692958 -0.077738551  0.03550084
```

- All the variables contribute (i.e. have a non-zero coefficients) to all components.
- If the goal is simply to use the new variables without necessarily interpreting them, then this is probably not a problem....

Illustration: PCA (cont')

- But sometimes we want to be able to understand these components...
- For instance, people often use factor analysis methods in this situation.
- One way to proceed is to use a rotation method. These methods try to produce a solution with many small coefficients (loadings in the factor analysis literature) and only a few large coefficients.
- Then the analyst can hand pick the variables that are related to each component (factor in terms of factor analysis), by setting a threshold under which a loading is judged not important.
- The components would also be easier to interpret if many coefficients were 0.
- Even if interpretation is not important, shrinking the coefficients might be a good idea, for instance when the number of variables is large compared to the number of observations.

Illustration: PCA (cont')

- This is where lasso-type regularization can be helpful.
- Zou (2006) proposed a clever way to perform shrinkage on the coefficients by recasting the PCA problem as a regression problem.
- Then they are able to use a lasso penalty in order to shrink the coefficients, and bring some of them to 0.
- It is implemented in the package `elasticnet`.

Illustration: PCA (cont')

Code R

```
> library(elasticnet)
> spcafit=spca(datpca,K=4,sparse="penalty",para=rep(0,4))
```

- The argument `K` specifies the number of components that we want
- The argument `para` provides the values of the λ tuning parameter for each component.
- In this first example, we set them all $\lambda's$ to 0, and no shrinkage is performed.
- We'll thus recover the original PCA solution.

Illustration: PCA (cont')

Code R

```
> spcafit

Call:
spca(x = datpca, K = 4, para = rep(0, 4), sparse = "penalty")

4 sparse PCs
Pct. of exp. var. : 20.2 16.7 16.2 10.8
Num. of non-zero loadings :  12 12 12 12
Sparse loadings
       PC1    PC2    PC3    PC4
x1  -0.208  0.104 -0.072  0.646
x2   0.192 -0.052  0.525  0.089
x3   0.388  0.364 -0.100 -0.037
x4   0.356 -0.338 -0.248  0.145
x5  -0.145  0.055 -0.082  0.694
x6   0.294  0.339 -0.007  0.077
x7   0.230 -0.141  0.505  0.159
x8   0.354 -0.401 -0.213  0.133
x9   0.280  0.421 -0.101  0.066
x10  0.268 -0.147  0.452  0.127
x11  0.308 -0.338 -0.345  0.031
x12  0.334  0.357 -0.078  0.036
```

Illustration: PCA (cont')

- If we use a λ of 32 for each component, we get the following sparse solution:

Code R

```
> spcafit=spca(datpca,K=4,sparse="penalty",para=rep(32,4))
> spcafit

Call:
spca(x = datpca, K = 4, para = rep(32, 4), sparse = "penalty")

4 sparse PCs
Pct. of exp. var. : 17.4 18.0 16.3 11.2
Num. of non-zero loadings :  3 4 3 2
Sparse loadings
      PC1   PC2   PC3   PC4
x1  0.000 0.000 0.000 0.697
x2  0.000 0.000 0.564 0.000
x3  0.000 0.560 0.000 0.000
x4  0.563 0.000 0.000 0.000
x5  0.000 0.000 0.000 0.718
x6  0.000 0.426 0.000 0.000
x7  0.000 0.000 0.607 0.000
x8  0.600 0.000 0.000 0.000
x9  0.000 0.512 0.000 0.000
x10 0.000 0.000 0.560 0.000
x11 0.568 0.000 0.000 0.000
x12 0.000 0.493 0.000 0.000
```

Illustration: PCA (cont')

- In fact, this is the solution that we get from a classical factor analysis.
- The first component is related to "service", the second one to "products", the third one to "payment", and the last one to "price".
- One practical problem here is that the package `elasticnet` does not provide an automatic way to select the tuning parameters.
- In this example, we tried different values before getting this one. However, a recent paper (Gajjar, 2017) proposes a method for selecting the tuning parameter with a genetic algorithm.

Final comments

- There are numerous other use of regularization.
- L_1 type regularization can be used in clustering; see Witten (2010) and the associated package `sparcl`.
- It can also be used in an interesting way with random forests; see Friedman (2008) and the package `pre` .
- We will come back to this combination of random forests and regularization later.