Part 9: Linear models - Section 1

Dr. Nguyen Quang Huy

July 20, 2020

Overview

In this section, we will discuss about the following topic:

1. Linear regression as a machine learning algorithm: Simple regression, model estimation, multivariable regression
2. Linear model selection: best subset selection, forward stepwise selection, backward stepwise selection.
3. Shrinkage method: Bias-variance trade-off, lasso regression, ridge regression
4. Polynomial regression
5. Dimension reduction method: principal components analysis and partial least squares

Linear regression

- Linear regression can be considered a machine learning algorithm for regression tasks.
- Linear regression is a parametric model where the function f is assumed to be linear:

$$f(X) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p$$

- Potential problems of linear model:
  - Non-linearity of the response-predictor relationships.
  - Correlation of error terms.
  - Non-constant variance of error terms (heteroscedasticity).
  - Outliers.
  - Collinearity.

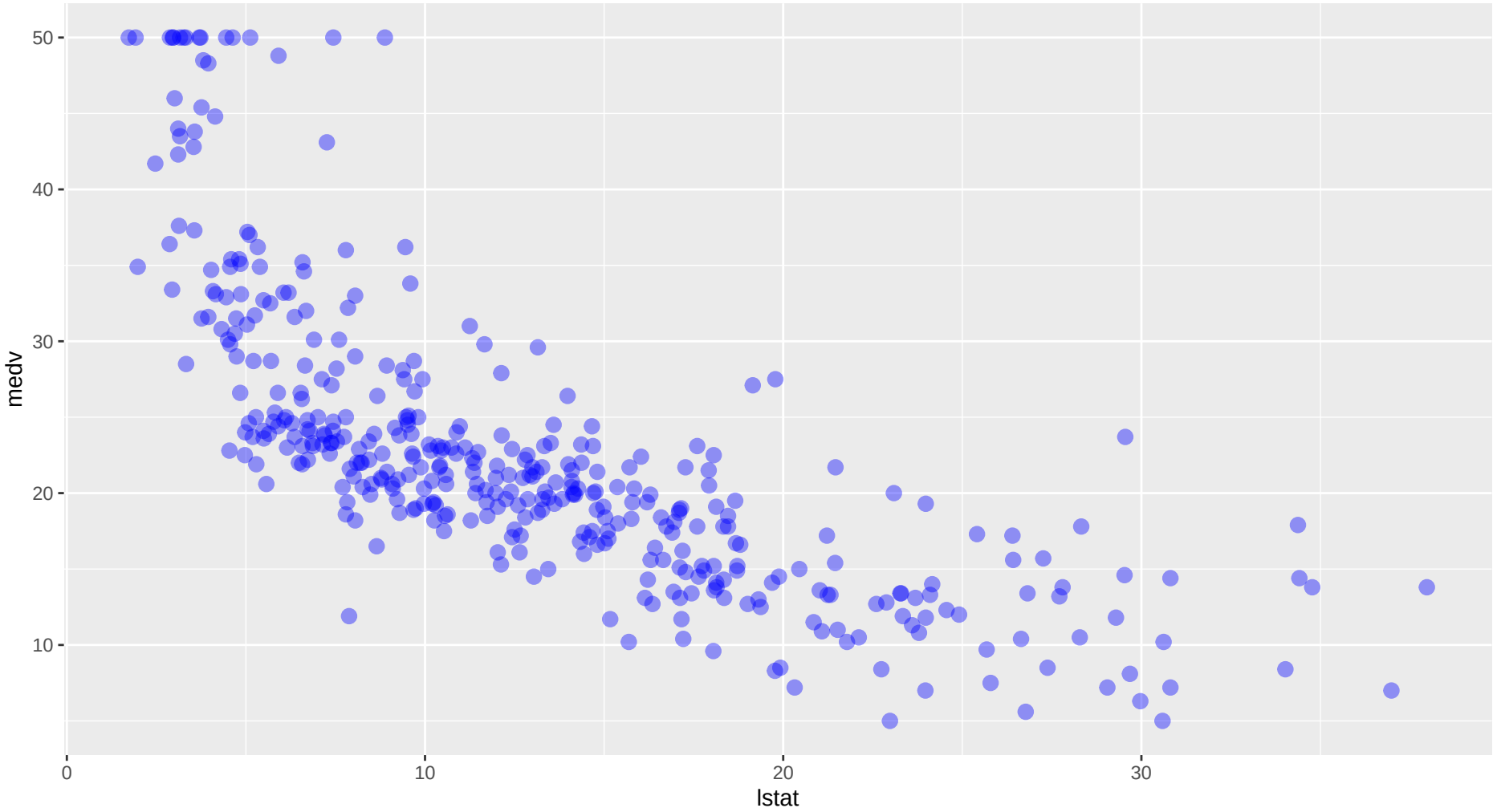
Linear regression - simple regression

- We use Boston dataset in "MASS" package where **medv** is the output.
- We believe that there is a strong relationship house price (medv) and percentage of low-status people (lstat) in the training dataset.

```
dat<-Boston
set.seed(1)
test_index<-createDataPartition(dat$medv,
            times = 1, p = 0.2,list=FALSE)
train<-dat[-test_index,]
test<-dat[test_index,]
train%>%ggplot(aes(lstat,medv))+
  geom_point(size=3,col="blue",alpha=0.4)
```

Linear regression - simple regression

Obviously, there is a negative relationship between **medv** and **lstat**



Linear regression - simple regression

If the relationship is linear, the model should be

$$medv \sim \beta_0 + \beta_1 \times lstat$$

```
lm.fit1<-lm(medv~lstat,data=train)
lm.fit1
```

```
##
## Call:
## lm(formula = medv ~ lstat, data = train)
##
## Coefficients:
## (Intercept)        lstat
##     34.9559      -0.9621
```

Linear regression - simple regression

```
summary(lm.fit1)
```

```
##
## Call:
## lm(formula = medv ~ lstat, data = train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -15.475  -4.185  -1.330   2.157  23.587
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 34.95592    0.64226   54.43   <2e-16 ***
## lstat       -0.96208    0.04355  -22.09   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 '
##
```

Linear regression - simple regression

If the relationship is linear, the model should be

$$\hat{medv} = 34.95592 - 0.96208 \times lstat$$

```
medv.pred<-predict(lm.fit1,test)
sqrt(mean((medv.pred-test$medv)^2))# model error
```

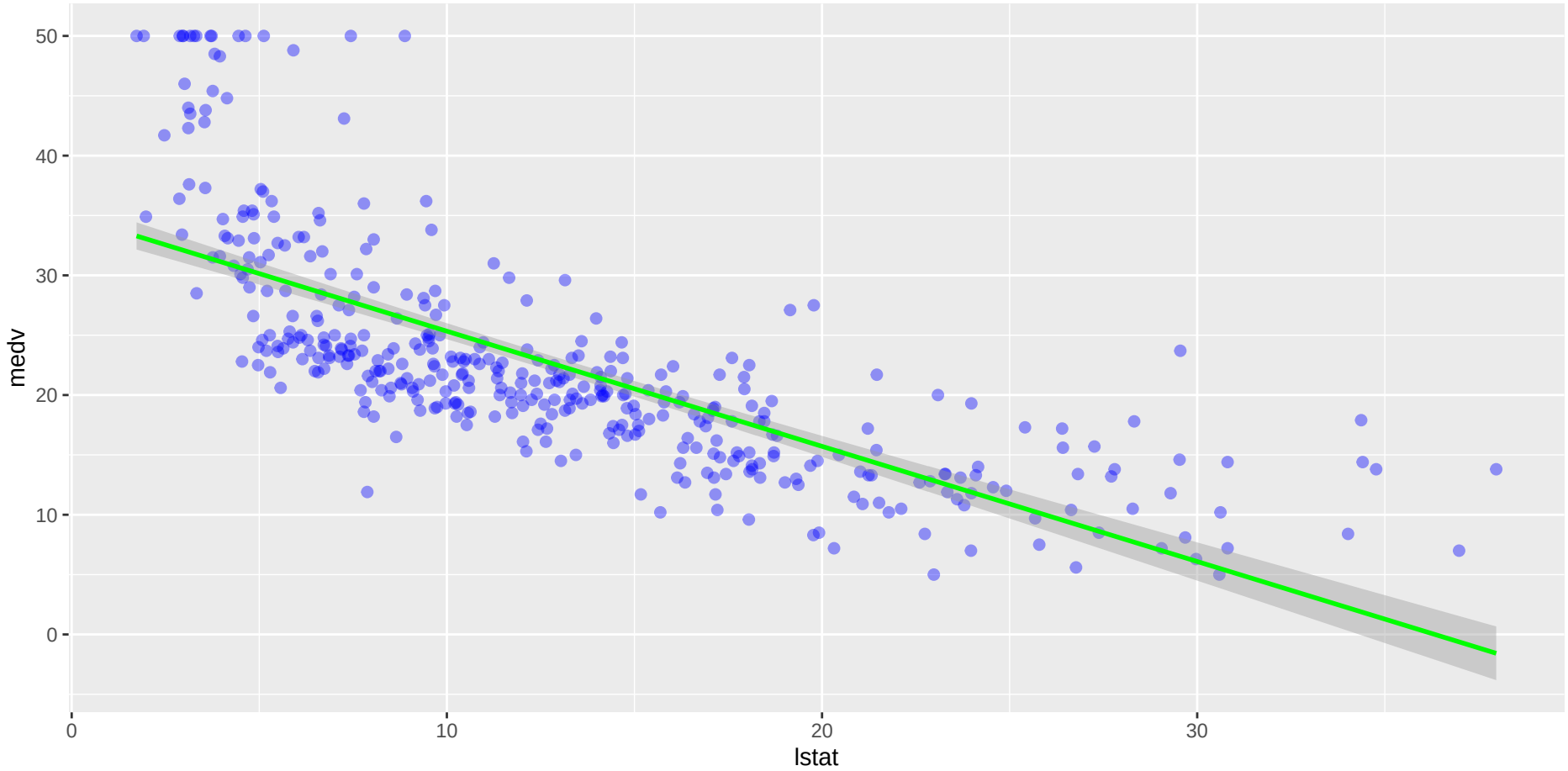
```
## [1] 5.56546
```

```
sqrt(mean((mean(train$medv)-test$medv)^2))# naive prediction
```

```
## [1] 7.971648
```

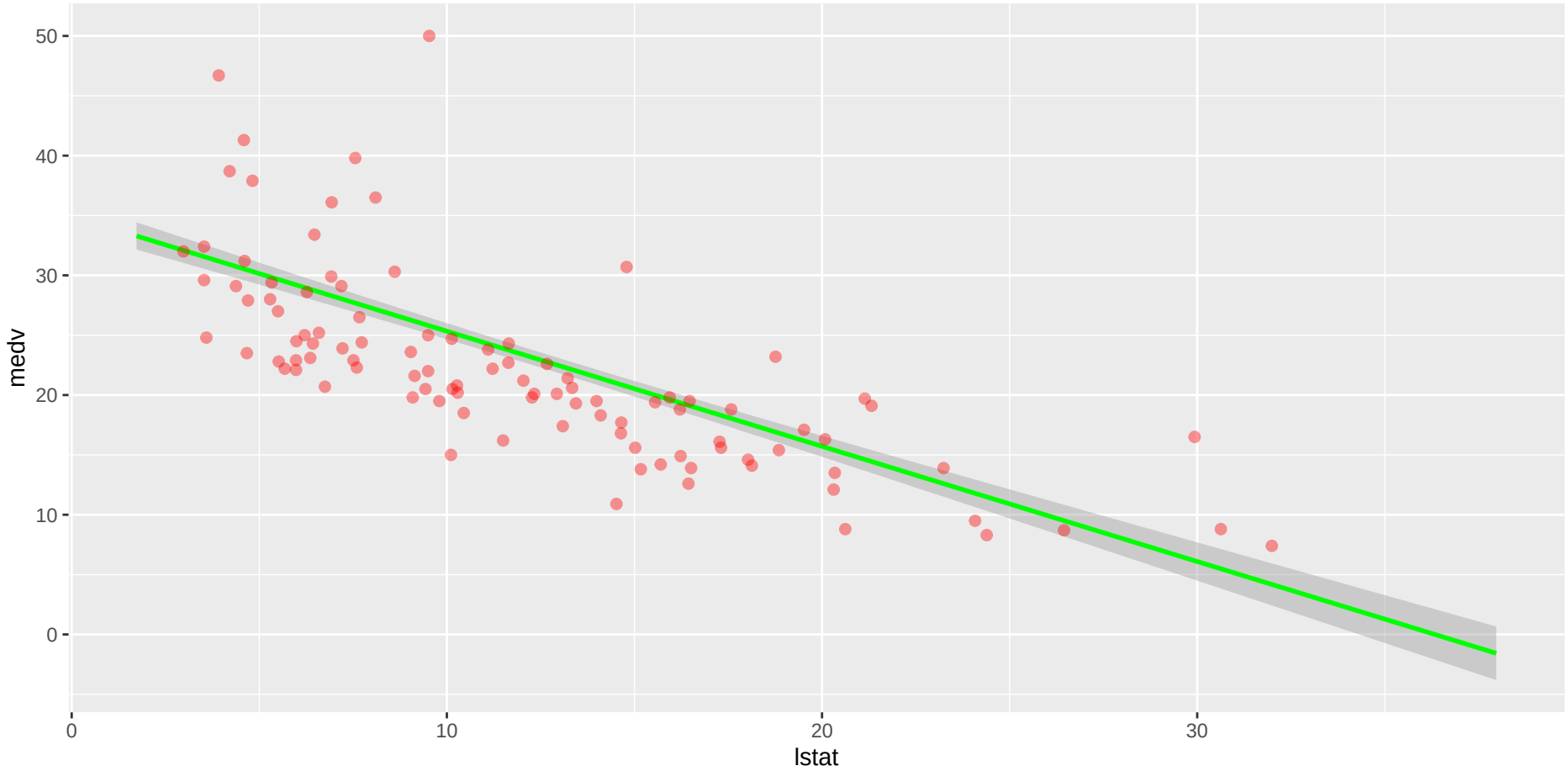
Linear regression - simple regression

```
train%>%ggplot(aes(lstat,medv))+
  geom_point(size=2,col="blue",alpha=0.4)+
  geom_smooth(method="lm",col="green")#+
```



Linear regression - simple regression

```
train%>%ggplot(aes(lstat,medv))+
  geom_smooth(method="lm",col="green")+
  geom_point(data=test,aes(lstat,medv),size=2,col="red",alpha=
```



Linear regression - model estimation

β_0 and β_1 are obtained by minimizing the loss function:

$$\hat{\beta}_0, \hat{\beta}_1 = \arg\min_{\beta_0,\beta_1} \sum_i (medv_i - \beta_0 - \beta_1 \times lstat_i)^2$$

The solution is

$$\begin{aligned} \hat{\beta}_1 &= cor(medv, lstat) \times \frac{\sigma(medv)}{\sigma(lstat)} \\ \hat{\beta}_0 &= mean(medv) - \hat{\beta}_1 \times mean(lstat) \end{aligned}$$

```
b1<-cor(train$medv,train$lstat)*sd(train$medv)/sd(train$lstat)
b1
```

```
## [1] -0.9620849
```

```
mean(train$medv)-b1*mean(train$lstat)
```

```
## [1] 34.95592
```

Linear regression - multivariable

We can add variable **dis** into the simple model.

```
lm.fit2<-lm(medv~lstat+dis,data=train)
summary(lm.fit2)
```

```
##
## Call:
## lm(formula = medv ~ lstat + dis, data = train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -16.841  -4.079  -1.475   2.018  21.387
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 38.79959    1.15715  33.530  < 2e-16 ***
## lstat       -1.06091    0.04951 -21.426  < 2e-16 ***
```

Linear regression - multivariable

The multi-variables model has smaller test error

```
medv.pred2<-predict(lm.fit2,test)
sqrt(mean((medv.pred2-test$medv)^2))
```

```
## [1] 5.431836
```

We can use all variables to build linear model:

```
lm.fit.all<-lm(medv~.,data=train)
medv.pred.all<-predict(lm.fit.all,test)
sqrt(mean((medv.pred.all-test$medv)^2))
```

```
## [1] 4.35529
```

IMPORTANT: DO NOT USE THE TEST SET TO EVALUATE MODELS $\rightarrow$ we use cross-validation to find the best linear model

Linear model - best subset selection

- Denote $\mathcal{M}_0$ the null model which contains no predictor.
- For $i = 1, 2, \cdots, p$
  - Fit all linear models which contain i predictors (There are $\binom{d}{i}$) models).
  - Pick the best among these models (denoted $\mathcal{M}_i$)
- Select single best model among $\mathcal{M}_0, \mathcal{M}_1, \cdots, \mathcal{M}_p$ using cross validated prediction error

$\rightarrow$ we always find the best model, but there is problem of computational time (there are 2^p models to be considered)

- For computational reasons, best subset selection cannot be applied with very large p
- Stepwise methods are attractive alternatives to best subset selection.

Linear model - forward stepwise selection

- Denote $\mathcal{M}_0$ the null model which contains no predictor.
- For $i = 0, 1, 2, \cdots, (p-1)$
  - Consider all $(p-i)$ models that augment the predictors in $\mathcal{M}_i$ with one additional predictor
  - Pick the best among these models (denoted $\mathcal{M}_{i+1}$)
- Select single best model among $\mathcal{M}_0, \mathcal{M}_1, \cdots, \mathcal{M}_p$ using cross validated prediction error

The number of models considered in forward stepwise selection is $\frac{p(p+1)}{2} - 1$

Linear model - backward stepwise selection

- Denote $\mathcal{M}_p$ the full model, which contains all p predictors.
- For $i = (p-1), (p-2), \cdots, 0$
  - Consider all i models that contain all but one of the predictors in $\mathcal{M}_i$ for a total of $i-1$ predictors.
  - Choose the best among these i model and call it $\mathcal{M}_{(i-1)}$.
- Select single best model among $\mathcal{M}_0, \mathcal{M}_1, \cdots, \mathcal{M}_p$ using cross validated prediction error.

The number of models considered in backward stepwise selection is $\frac{p(p+1)}{2} - 1$

Linear model selection - Boston dataset

We perform best subset selection to find the best model. Function **Huy.cv.lm(dat,K,varname)** calculated cross validation error of linear regression model:

```
setwd("C:/Users/AD/Desktop/Tex file/Thu latex/Introduction to
source("MyFunction.R") # goi tat ca ham so trong file myfunct
cv.lm<-Huy.cv.lm(train[,6:14],k=5,"medv") #it takes times
cv.lm$cv.error
```

```
## [1] 5.222781
```

```
var<-cv.lm$variables
model<-lm(train$medv~.,data=train[,var])
pred<-predict(model,test[,var])
sqrt(mean((pred-test$medv)^2))
```

```
## [1] 4.310004
```

Linear model selection

Load dataset **Carseats** from **ISLR** packages and build a linear model to predict the Sales variables.

- Chia dữ liệu ra thành training và test theo tỷ lệ 80% - 20%.
- Nếu giá dự đoán là giá trung bình trên tập training thì sai số dự đoán là bao nhiêu?
- Xây dựng mô hình tuyến tính đơn giản: Sales $\sim$ Price + CompPrice để dự đoán Sales. Sai số dự đoán (trên tập test) là bao nhiêu?
- Nếu sử dụng tất cả các biến số có trong dữ liệu, sai số dự đoán là bao nhiêu?
- Hãy tìm mô hình tuyến tính có cross validation error nhỏ nhất. Sai số của mô hình này trên test dataset là bao nhiêu?

Linear model - Bias and variance trade-off

Suppose that the true relationship between output y and predictors x is

$$y = f(x) + \epsilon \text{ where } Var(\epsilon) = \sigma^2$$

With $\hat{f}$ is an estimator of f; the test error can be decomposed as follows:

$$\begin{aligned}
\mathbb{E}\left[(y - \hat{f}\right]^2 &= \mathbb{E}\left[(f - \hat{f} + \epsilon\right]^2 \\
&= \mathbb{E}\left[(f - \hat{f}\right]^2 + \sigma^2 \\
&= \mathbb{E}\left[(f - \mathbb{E}(\hat{f})\right]^2 + \mathbb{E}\left[(\hat{f} - \mathbb{E}(\hat{f})\right]^2 + \sigma^2 \\
&= \left[Bias(\hat{f})\right]^2 + Variance(\hat{f}) + \sigma^2
\end{aligned}$$

- Bias variance trade-off: more complex model, bias decreases but variance increases and vice versa.
- Linear models are known as "high bias and low variance"

Linear model - shrinkage methods

- Adding variable in linear model $\rightarrow$ reducing the bias but increasing the variance.
- The subset selection methods involve using least squares to fit a linear model that contains a subset of the predictors.
- Shrinkage methods fit a model containing all p predictors using a technique that shrinks the coefficient estimates towards zero.
- Shrinkage methods result has higher bias but lower variance than the subset selection methods.
- The two best-known techniques for shrinking the regression coefficients towards zero are **ridge regression** and **the lasso.**

Linear model - ridge regression

To estimate the coefficients β_i in linear models, we minimize

$$\sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 \, x_{i,1} - \beta_2 \, x_{i,2} - \cdots - \beta_p \, x_{i,p})^2$$

In ridge regression, we find β_i^R that minimize a slightly different quantity

$$\sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 \, x_{i,1} - \beta_2 \, x_{i,2} - \cdots - \beta_p \, x_{i,p})^2 + \lambda \left(\beta_1^2 + \cdots + \beta_p^2\right)$$

where $\lambda \geq 0$ is a *tuning parameter*. Quantity $\lambda \sum \beta_i^2$ is called a *shrinkage penalty* which is small when β_i are close to 0

- When $\lambda = 0$, ridge regression will produce the least squares estimates.
- When λ is large, the impact of the shrinkage penalty grows, and the ridge regression coefficient estimates will approach zero.
- Selecting a good value for λ is critical (cross-validation is recomended)

Ridge regression - Boston dataset

We use *glmnet()* function in **glmnet** package to perform the ridge regression.

```
library(glmnet)
x<-model.matrix(medv~.,Boston)[,-14] #transform all to numeric
y<-Boston$medv
x.train<-x[-test_index,]
y.train<-y[-test_index]
x.test<-x[test_index,]
y.test<-y[test_index]
# alpha = 0 means ridge regression
ridge.reg<-glmnet(x.train,y.train,alpha=0,lambda=1)
# coef(ridge.reg): print estimated beta
ridge.pred= predict(ridge.reg, newx=x.test)
sqrt(mean((ridge.pred-y.test)^2))
```

```
## [1] 4.726685
```

Ridge regression - Boston dataset

We use cross validation approach to find the best λ. Function *cv.glmnet()* performs cross-validation as follows:

```
v.lambda<-10^seq(-3,2,length=500)# tao ra vector lambda
cv.out<-cv.glmnet(x.train, y.train, alpha = 0,
                  nfolds = 5, lambda=v.lambda)
# cv.out$lambda.min: value of lambda with smallest cv error
ridge.reg = glmnet (x.train,y.train, alpha = 0,
                    lambda = cv.out$lambda.min)
ridge.pred= predict(ridge.reg , newx=x.test)
sqrt(mean((ridge.pred-y.test)^2))
```

```
## [1] 4.717904
```

Linear model - the lasso

To estimate the coefficients β_i in linear models, we minimize

$$\sum_{i=1}^{n} (y_i - \beta_0 - \beta_1\ x_{i,1} - \beta_2\ x_{i,2} - \cdots - \beta_p\ x_{i,p})^2$$

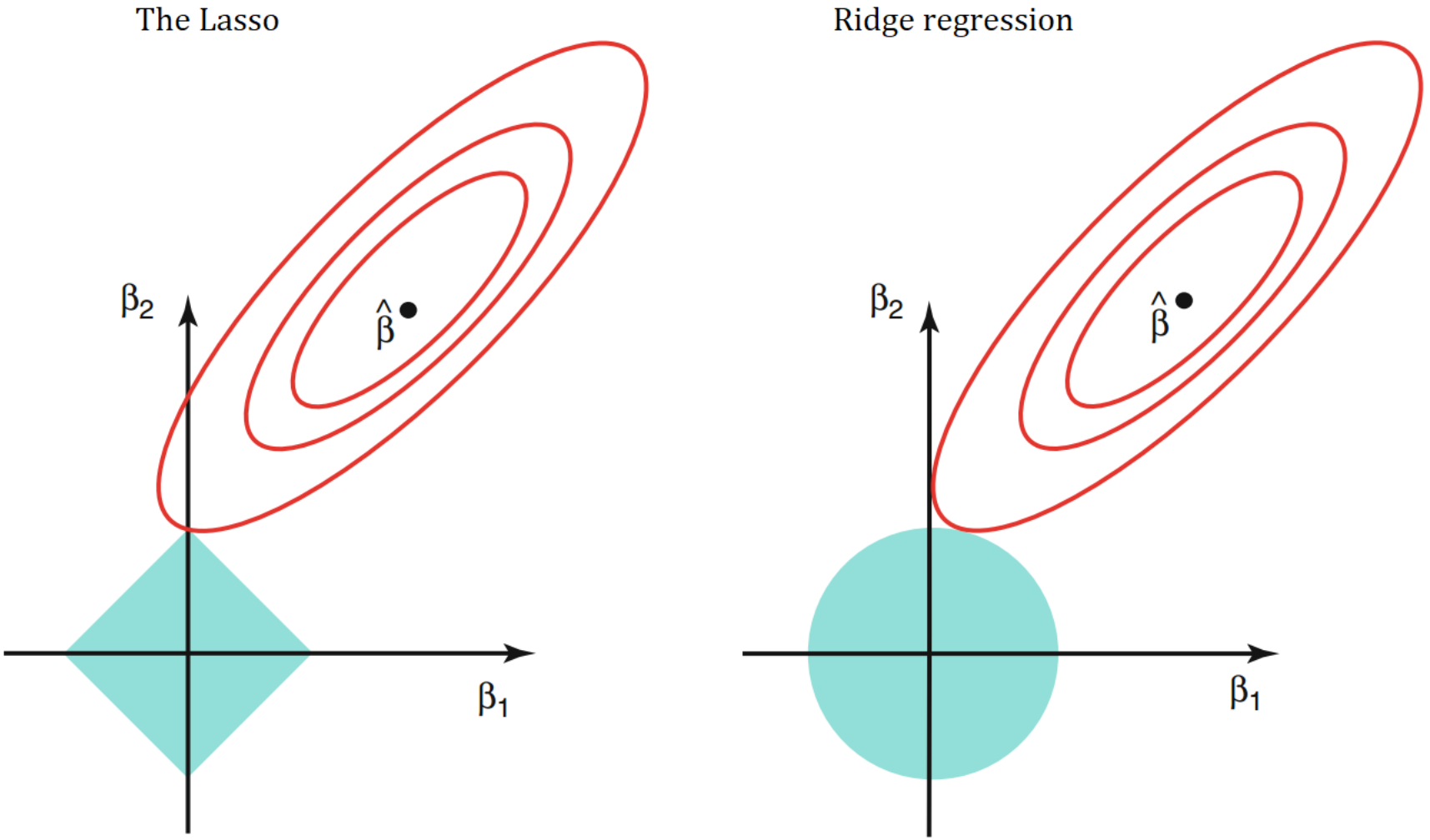
In ridge regression, we find β_i^R that minimize a slightly different quantity

$$\sum_{i=1}^{n} (y_i - \beta_0 - \beta_1\ x_{i,1} - \beta_2\ x_{i,2} - \cdots - \beta_p\ x_{i,p})^2 + \lambda\,(|\beta_1| + \cdots + |\beta_p|)$$

where $\lambda \geq 0$ is a *tuning parameter*. Quantity $\lambda \sum |\beta_i|$ is called a *shrinkage penalty* which is small when β_i are close to 0

- When $\lambda = 0$, the lasso will produce the least squares estimates.
- When λ is large, the impact of the shrinkage penalty grows, and the ridge regression coefficient estimates will approach zero.
- Selecting a good value for λ is critical (cross-validation is recomended)

Ridge regression and the Lasso



In the lasso, the ellipse will often intersect the constraint region at an axis

The lasso - Boston dataset

```
library(glmnet)
x<-model.matrix(medv~.,Boston)[,-14] #transform all to numeri
y<-Boston$medv
x.train<-x[-test_index,]
y.train<-y[-test_index]
x.test<-x[test_index,]
y.test<-y[test_index]
v.lambda<-10^seq(-3,2,length=500)# tao ra vector lambda
cv.out<-cv.glmnet(x.train, y.train, alpha = 1,
                  nfolds = 5, lambda=v.lambda)
lasso.reg<-glmnet (x.train,y.train, alpha = 1,
                   lambda = cv.out$lambda.min)
lasso.pred<-predict(lasso.reg , newx=x.test)
sqrt(mean((lasso.pred-y.test)^2))
```

```
## [1] 4.736412
```

Linear model - ridge regression and the lasso

Load dataset **Carseats** from **ISLR** packages and build a linear model to predict the Sales variables.

- Chia dữ liệu ra thành training và test theo tỷ lệ 80% - 20%.
- Ridge regression: Lựa chọn λ để tối thiểu hóa cross validation error, với λ tìm được hãy xem sai số dự đoán trên tập test.
- The lasso: Lựa chọn λ để tối thiểu hóa cross validation error, với λ tìm được hãy xem sai số dự đoán trên tập test.

Linear regression

There are many extensions to linear regression

- 1. Polynomial regression:

$$\begin{aligned} f(X) = \beta_0 \quad &+ \quad \beta_{11}\, X_1 + \beta_{12}\, (X_1)^2 + \beta_{13}\, (X_1)^3 + \cdots \\ &+ \quad \beta_{21}\, X_2 + \beta_{22}\, (X_2)^2 + \beta_{23}\, (X_2)^3 + \cdots \\ &+ \quad \cdots \end{aligned}$$

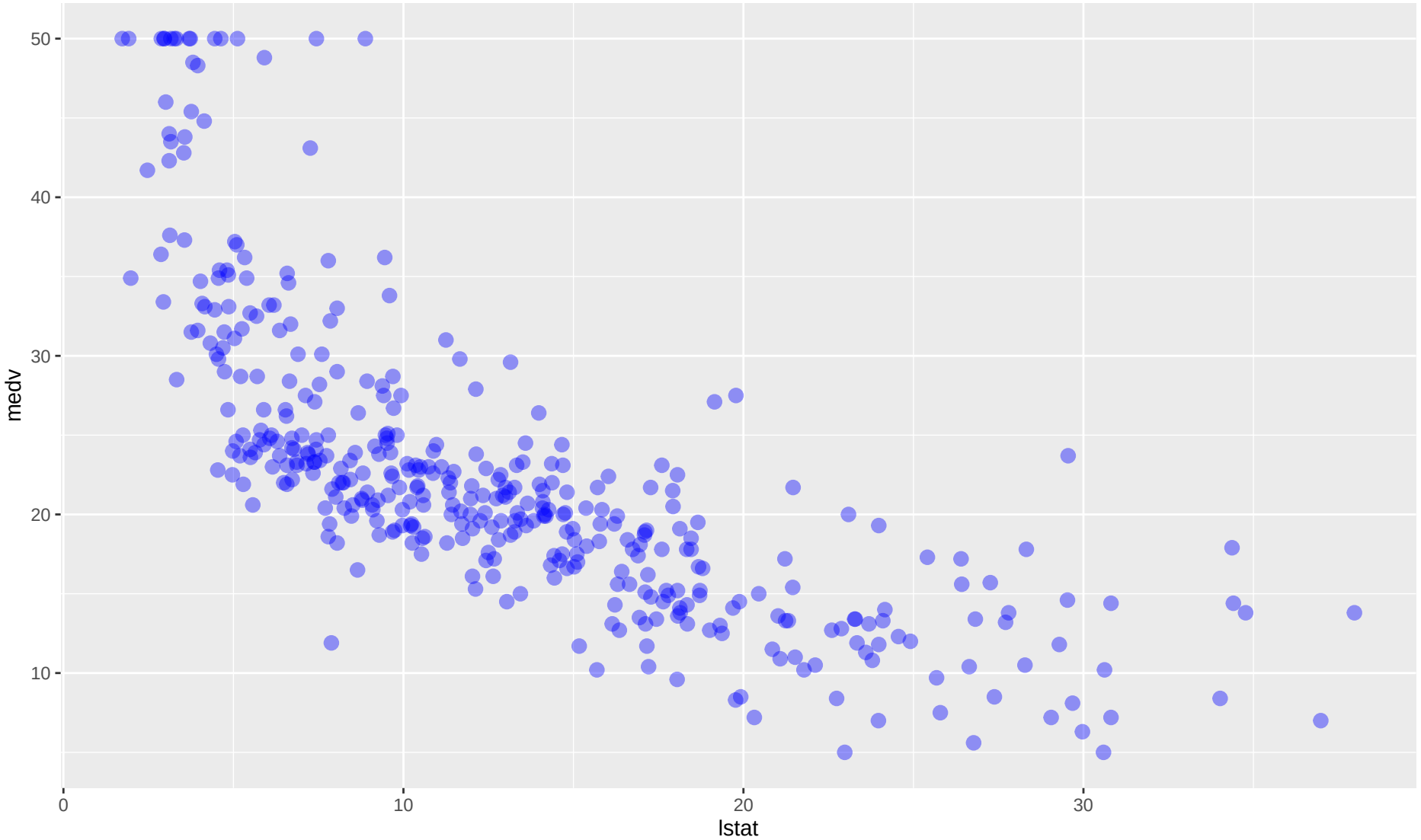
- 2. Step function regression:

$$f(X) = \beta_0 + \beta_{11}\, I_{X_1 \leq c_1} + \beta_{12}\, I_{c_1 < X_1 \leq c_2} + \beta_{13}\, I_{c_2 < X_1 \leq c_3} + \cdots$$

- 3. Regression splines.
- 4. Local regression.
- 5. Generalized addative model.

Linear regression - polynomial regression

Non-linear relationship between **medv** and **lstat**.



Linear regression - polynomial regression

Polynomial regression with Boston dataset

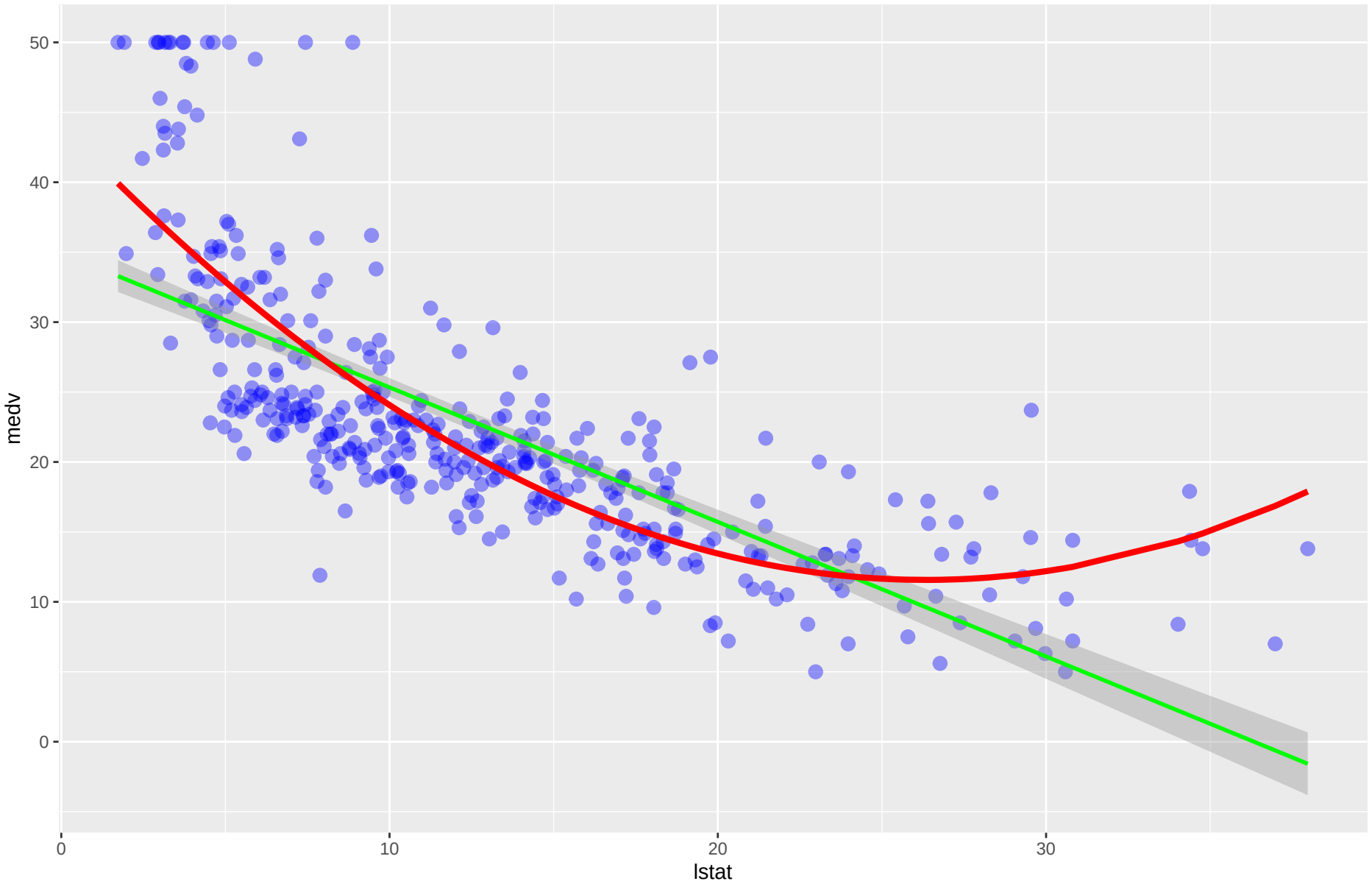
```
train<-Boston[-test_index,]
test<-Boston[test_index,]
poly.fit<-lm(medv~poly(lstat,2, raw=TRUE),data=train)
poly.pred<-predict(poly.fit,test)
sqrt(mean((poly.pred-test$medv)^2))
```

```
## [1] 5.539094
```

```
summary(poly.fit)
```

```
##
## Call:
## lm(formula = medv ~ poly(lstat, 2, raw = TRUE), data = trai
##
## Residuals:
##     Min      1Q  Median      3Q     Max
```

Linear regression - polynomial regression



Linear regression - polynomial regression

Polynomial regression with Boston dataset

```
poly.fit2<-lm(medv~lstat+lstat^2+dis,data=train)
poly.pred2<-predict(poly.fit2,test)
sqrt(mean((poly.pred2-test$medv)^2))
```

```
## [1] 5.431836
```

```
summary(poly.fit2)
```

```
##
## Call:
## lm(formula = medv ~ lstat + lstat^2 + dis, data = train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -16.841  -4.079  -1.475   2.018  21.387
##
```

Linear regression - polynomial regression

Polynomial regression with Boston dataset

```
poly.fit3<-lm(medv~lstat+lstat^2+dis + dis^2,data=train)
poly.pred3<-predict(poly.fit3,test)
sqrt(mean((poly.pred3-test$medv)^2))
```

```
## [1] 5.431836
```

```
summary(poly.fit3)
```

```
##
## Call:
## lm(formula = medv ~ lstat + lstat^2 + dis + dis^2, data = t
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -16.841  -4.079  -1.475   2.018  21.387
##
```

Linear regresson - polynomial regression

Combine the polynomial regression and the lasso:

```
dat<-Boston
dat<-data.frame(dat,dat[,-14]^2)
x<-model.matrix(medv~.,dat)[,-14] #transform all to numeric
y<-dat$medv
x.train<-x[-test_index,]
y.train<-y[-test_index]
x.test<-x[test_index,]
y.test<-y[test_index]
v.lambda<-10^seq(-3,2,length=500)# tao ra vector lambda
cv.out<-cv.glmnet(x.train, y.train, alpha = 1, nfolds = 5, lam
lasso.reg<-glmnet (x.train,y.train, alpha = 1,lambda = cv.out$
lasso.pred<-predict(lasso.reg , newx=x.test)
sqrt(mean((lasso.pred-y.test)^2))
```

```
## [1] 3.986785
```

Linear regresson - polynomial regression

Combine the polynomial regression and the lasso (up to degree 3)

```
dat<-Boston
dat<-data.frame(dat,dat[,-14]^2, dat[,-14]^3)
x<-model.matrix(medv~.,dat)[,-14] #transform all to numeric
y<-dat$medv
x.train<-x[-test_index,]
y.train<-y[-test_index]
x.test<-x[test_index,]
y.test<-y[test_index]
v.lambda<-10^seq(-3,2,length=500)# tao ra vector lambda
cv.out<-cv.glmnet(x.train, y.train, alpha = 1, nfolds = 5, lam
lasso.reg<-glmnet (x.train,y.train, alpha = 1,lambda = cv.out$
lasso.pred<-predict(lasso.reg , newx=x.test)
sqrt(mean((lasso.pred-y.test)^2))
```

```
## [1] 3.784132
```

Linear regresson - polynomial regression

Combine the polynomial regression and the lasso (up to degree 4)

```
dat<-Boston
dat<-data.frame(dat,dat[,-14]^2, dat[,-14]^3, dat[,-14]^4)
x<-model.matrix(medv~.,dat)[,-14] #transform all to numeric
y<-dat$medv
x.train<-x[-test_index,]
y.train<-y[-test_index]
x.test<-x[test_index,]
y.test<-y[test_index]
v.lambda<-10^seq(-3,2,length=500)# tao ra vector lambda
cv.out<-cv.glmnet(x.train, y.train, alpha = 1, nfolds = 5, lam
lasso.reg<-glmnet (x.train,y.train, alpha = 1,lambda = cv.out$
lasso.pred<-predict(lasso.reg , newx=x.test)
sqrt(mean((lasso.pred-y.test)^2))
```

```
## [1] 3.790705
```

Linear regresson - polynomial regression

Cải thiện performance bằng cách tạo ra dữ liệu đạng nhân chéo ($x_i \times x_j$)

```
dat<-Boston
ii<-14
for(i in 1:13){
  for (j in 1:13){
    ii<-ii+1
    dat<-mutate(dat,dat[,i]*dat[,j])
    names(dat)[ii]<-paste0("col",i*10,j)
  }
}
set.seed(1)
test_index<-createDataPartition(dat$medv,
                                times = 1, p = 0.2,list=FALSE)
train<-dat[-test_index,]
test<-dat[test_index,]
```

Linear regresson - polynomial regression

```
x<-model.matrix(medv~.,dat)[,-14] #transform all to numeric
y<-dat$medv
x.train<-x[-test_index,]
y.train<-y[-test_index]
x.test<-x[test_index,]
y.test<-y[test_index]
v.lambda<-10^seq(-3,2,length=500)# tao ra vector lambda
cv.out<-cv.glmnet(x.train, y.train, alpha = 1, nfolds = 5, lam
lasso.reg<-glmnet (x.train,y.train, alpha = 1,lambda = cv.out$
lasso.pred<-predict(lasso.reg , newx=x.test)
sqrt(mean((lasso.pred-y.test)^2))
```

```
## [1] 3.515027
```

Linear regression - qualitative variable

We have assumed that all variables in our linear regression model are *quantitative*, some predictors are *qualitative*. Loading **Carseats** data from **ISLR** packages where *ShelveLoc* is a qualitative variable (factor).

```
dat<-Carseats
str(dat)
```

```
## 'data.frame':    400 obs. of  11 variables:
##  $ Sales       : num  9.5 11.22 10.06 7.4 4.15 ...
##  $ CompPrice   : num  138 111 113 117 141 124 115 136 132 13
##  $ Income      : num  73 48 35 100 64 113 105 81 110 113 ...
##  $ Advertising: num  11 16 10 4 3 13 0 15 0 0 ...
##  $ Population : num  276 260 269 466 340 501 45 425 108 131
##  $ Price       : num  120 83 80 97 128 72 108 120 124 124 ...
##  $ ShelveLoc   : Factor w/ 3 levels "Bad","Good","Medium": 1
##  $ Age         : num  42 65 59 55 38 78 71 67 76 76 ...
##  $ Education   : num  17 10 12 14 13 16 15 10 10 17 ...
```

Linear regression - qualitative variable

```
summary(lm(Sales~Price+ShelveLoc,data=Carseats))
```

```
##
## Call:
## lm(formula = Sales ~ Price + ShelveLoc, data = Carseats)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5.8229 -1.3930 -0.0179  1.3868  5.0780
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)     12.001802   0.503447  23.839  < 2e-16 ***
## Price           -0.056698   0.004059 -13.967  < 2e-16 ***
## ShelveLocGood    4.895848   0.285921  17.123  < 2e-16 ***
## ShelveLocMedium  1.862022   0.234748   7.932 2.23e-14 ***
##
```

Linear regression - qualitative variable

What is estimated model of *Sales* depending on *Price* and *ShelveLoc*?

Linear regression - qualitative variable

What is estimated model of *Sales* depending on *Price* and *ShelveLoc*?

$$\hat{Sales} = \begin{cases} 12.002 - 0.057 \times Price \text{ if ShelveLoc} = \text{"bad"} \\ 13.864 - 0.057 \times Price \text{ if ShelveLoc} = \text{"medium"} \\ 16.898 - 0.057 \times Price \text{ if ShelveLoc} = \text{"good"} \end{cases}$$

```
(lm(Sales~Price+as.numeric(ShelveLoc),data=Carseats))
```

```
##
## Call:
## lm(formula = Sales ~ Price + as.numeric(ShelveLoc), data =
##
## Coefficients:
##          (Intercept)                 Price  as.numeric(Sh
##              12.39268               -0.05336
```

Linear model - dimension reduction

The methods that we have discussed have controlled variance in two different ways

- Using a subset of the original variables
- Shrinking their coefficients toward zero

The dimension reduction methods:

1. Transforming original predictors $X_1, \cdots, X_p$ into new predictors $Z_1, \cdots, Z_k$ where $k << p$

$$Z_i = \sum_{j=1}^{p} \alpha_{i,j} \, X_j \text{ for } i = 1, 2, \cdots, k$$

2. Fitting a least squares model using the transformed variables.

$$Y = \beta_0 + \beta_1 Z_1 + \beta_2 Z_2 + \cdots + \beta_k Z_k + \epsilon$$

High dimensions discussion

1. Traditional techniques for regression and classification are intended for the low-dimensional setting in which $n >> p$
2. PCA and PLS are dimension-reduction techniques which are useful with high-dimensional data (where $p \sim n$ or $p > n$)
3. Example 1: Model to predict a patient's blood pressure
   - Predictors are age, gender, body mass index, ... $\rightarrow$ low-dimensional data
   - Predictors are DNA data $\rightarrow$ high-dimensional data
4. Example 2: Model to predict a customer behaviour (buy or not buy a product)
   - Predictors are age, gender, income, ... $\rightarrow$ low-dimensional data
   - Predictors are all of the search terms entered by customers of a search engine $\rightarrow$ high-dimensional data

Linear model - principal components analysis

Principal components analysis (PCA) is a popular approach for deriving a low-dimensional set of features from a large set of variables

1. The first principal component is the linear combination $Z_1 = \mathbf{X}\alpha_1$ such that

$$\begin{cases} \alpha_1^{'}\alpha_1 = 1 \\ Var(Z_1) = \alpha_1^{'}\Sigma\alpha_1 \rightarrow max \end{cases}$$

   where $\alpha_1 = \{\alpha_{11}, \alpha_{12}, \cdots, \alpha_{1p}\}$ and Σ is the covariance matrix of $\mathbf{X}$

2. The second principal component is the linear combination $Z_2 = \mathbf{X}\alpha_2$

$$\begin{cases} \alpha_2^{'}\alpha_1 = 0 \\ \alpha_2^{'}\alpha_2 = 1 \\ Var(Z_2) = \alpha_2^{'}\Sigma\alpha_2 \rightarrow max \end{cases}$$

Linear model - principal components analysis

- Σ is a covariance matrix $\rightarrow$ it is non-negative definite matrix i.e. all eigenvalues are non-negative.
- We can prove that α_1 is the eigenvector corresponding to the largest eigenvalue λ_1 of Σ.

$$\Sigma\ \alpha_1 = \lambda_1\ \alpha_1$$

- α_i is the eigenvector corresponding to the i^{th} largest eigenvalue λ_i of Σ.
- Eigenvalues of Σ are solutions (λ) of equation

$$det(\Sigma - \lambda I) = 0(number)$$

- Eigenvector corresponding to λ is the solution (α) of

$$(\Sigma - \lambda I)\alpha = \mathbf{0}(vector)\alpha^{'}\ \alpha = 1$$

Linear model - principal components analysis

```
n<-10^4
f1<-rnorm(n,0,1)
f2<-rnorm(n,0,1)
X1<-2*f1 + f2 + 2
X2<-f1 + 2*f2 + 1
```

- What is the covariance matrix of $\mathbf{X} = (X_1, X_2)$
- Calculating (manually :D) the eigenvalues (λ_1,λ_2) and the corresponding eigenvectors (α_1,α_2)
- Calculating the first and the second second principal components $Z_1 = \mathbf{X}\alpha_1$ and $Z_2 = \mathbf{X}\alpha_2$

Linear model - principal components analysis

1.

```
X<-data.frame(X1,X2)
round(cov(X),0)
```

```
##     X1 X2
## X1   5  4
## X2   4  5
```

2.

$$det(\Sigma - \lambda I) = 0 \rightarrow (5-\lambda)^2 - 4^2 = 0 \rightarrow \begin{cases} \lambda_1 = 9 \\ \lambda_2 = 1 \end{cases}$$

3. Suppose that $\alpha_1 = (\alpha_{11}, \alpha_{12})'$

$$(\Sigma - 9I)\alpha_1 = \mathbf{0} \rightarrow \begin{cases} -4\alpha_{11} + 4\alpha_{12} = 0 \\ \alpha_{11}^2 + \alpha_{12}^2 = 1 \end{cases} \rightarrow \begin{cases} \alpha_{11} = 1/\sqrt{2} \\ \alpha_{12} = 1/\sqrt{2} \end{cases}$$

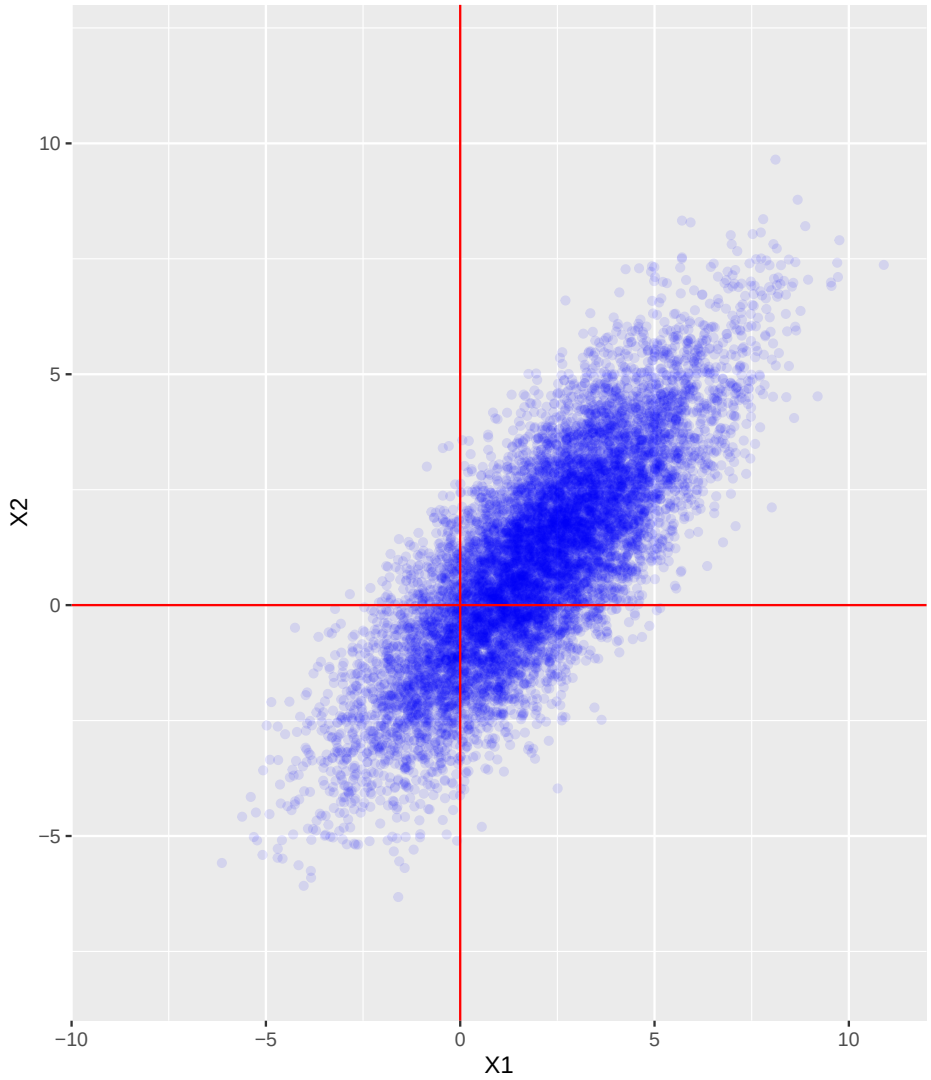
Linear model - principal components analysis

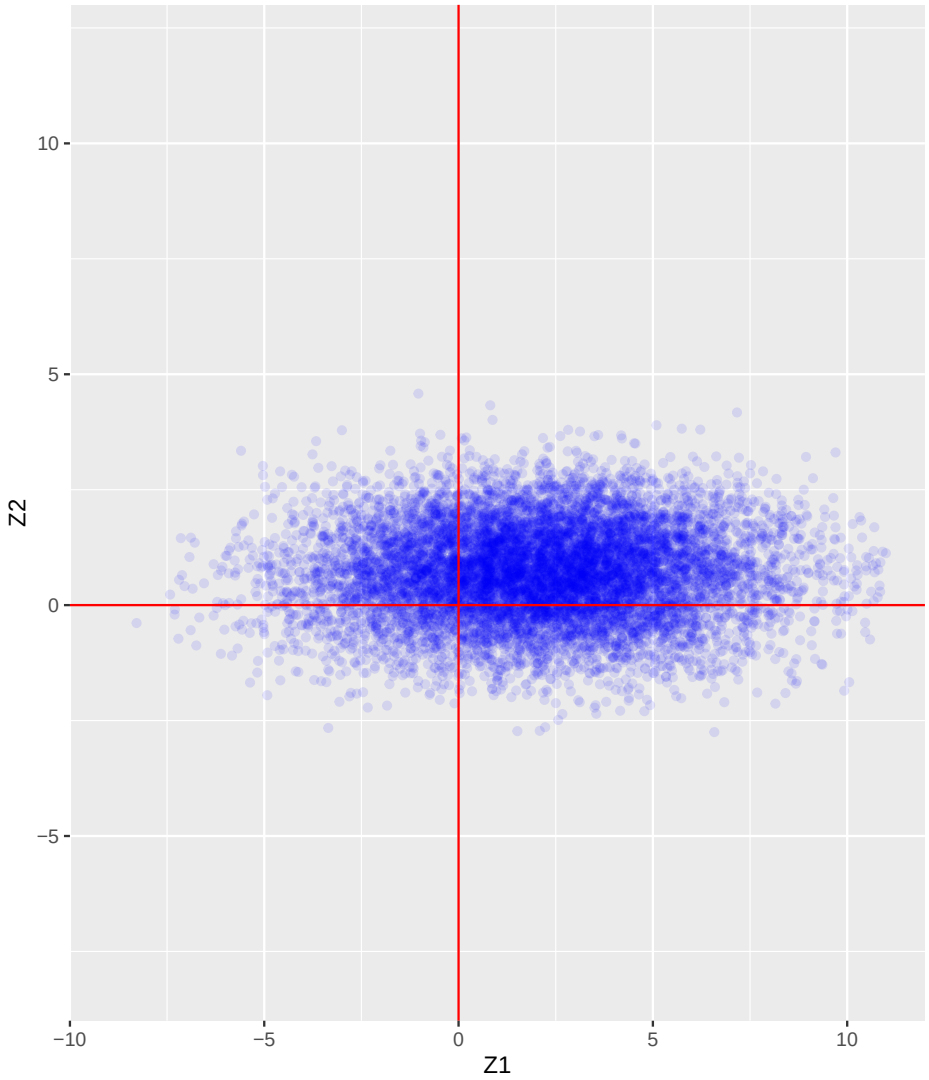
Similarly, we have $\alpha_2 = (1/\sqrt{2}, -1/\sqrt{2})'$.

```
alpha1<-c(1/sqrt(2),1/sqrt(2))
alpha2<-c(1/sqrt(2),-1/sqrt(2))
Z1<-as.matrix(X)%*%alpha1
Z2<-as.matrix(X)%*%alpha2
X<-mutate(X,Z1=Z1,Z2=Z2)
```

```
p1<-ggplot(X,aes(X1,X2))+geom_point(alpha=0.1,color="blue")+
  xlim(-9,11)+ylim(-8,12)+geom_hline(yintercept=0,color="red")
  geom_vline(xintercept=0,color="red")
p2<-ggplot(X,aes(Z1,Z2))+geom_point(alpha=0.1,color="blue")+
  xlim(-9,11)+ylim(-8,12)+geom_hline(yintercept=0,color="red")
  geom_vline(xintercept=0,,color="red")
```

Linear model - principal components analysis





Linear model - principal compinents analysis

1. We should standardize each predictor prior to generating the principal components so that all variables are on the same scale.
2. In the absence of standardization, the high-variance variables will tend to play a larger role in the principal components obtained.
3. The PCR approach involves identifying linear combinations that best represent the predictors $X_1, X_2, \cdots, X_p$ in an *unsupervised* way, since the response Y is not used to help determine the principal component directions.
4. In PCR, there is no guarantee that the directions that best explain the predictors will also be the best directions to use for predicting the response.

$\rightarrow$ they present partial least squares (PLS), a supervised alternative to PCR

Linear model - partial least squares

1. The first component is the linear combination $Z_1 = \mathbf{X}\gamma_1$; $\gamma_1 = \{\gamma_{11}, \gamma_{12}, \cdots, \gamma_{1p}\}$ such that γ_{1j} is the coefficient from the simple linear regression of Y onto X_j
2. Taking the residual $X_j^{(1)}$ from the simple linear regression of X_j, $j = 1, 2, \cdots, p$, on Z_1;
3. The second component is the linear combination $Z_2 = \mathbf{X}^{(1)}\gamma_2$; $\gamma_2 = \{\gamma_{21}, \gamma_{22}, \cdots, \gamma_{2p}\}$ such that γ_{2j} is the coefficient from the simple linear regression of Y onto $X_j^{(1)}$
4. Taking the residual $X_j^{(2)}$ from the simple linear regression of $X_j^{(1)}$, $j = 1, 2, \cdots, p$, on Z_1 and Z_2;
5. ...

Linear model - PCA and PLS practice

1. Load dataset **Boston** from **MASS** packages and build a linear model to predict the Salary of basket-ball players.

```
dat<-Boston
```

2. Standardize all numerical variables (except for **medv**)
3. Calculate the correlation matrix of numerical variables. Calculate the eigenvalues and corresponding eigenvectors of the correlation matrix.
4. Calculate the 1^{st}, 2^{nd}, 3^{rd}, *cdots*, principal components using PCA
5. Split data into trainning set and test set (80%-20%)
6. Build a linear model based on these components and non-numeric varibles.

Linear model - PCA and PLS practice

2. Standardize all numerical variables (except for **Salary**)

```
standardize<-function(x){x<-(x-mean(x,na.rm=TRUE))/sd(x,na.rm
for (col in names(dat)){
  if((col!="medv")&class(dat[,col]) %in% c("integer","numeric"
    dat[,col]<-standardize(dat[,col])
  }
}
```

3. Calculate the correlation matrix of numerical variables

```
p<-dim(dat)[2]-1
cor.M<-cor(dat)[1:p,1:p]
ev<-eigen(cor.M) # $value: eigen value, $vectors: eigenvectors
```

Linear model - PCA and PLS practice

4. Calculate the 1^{st}, 2^{nd}, 3^{rd}, *cdots*, principal components using PCA

```
newdat<-as.data.frame(as.matrix(dat[,-(p+1)])%*%ev$vectors)
names(newdat)<-paste0("Z",1:p)
```

5. Split data into trainning set and test set (80%-20%)

```
set.seed(1)
test_index<-createDataPartition(dat$medv, times = 1, p = 0.2,l
y<-dat$medv
# data in lasso or ridge must be matrix (not data.frame)
x.train<-as.matrix(cbind(newdat[-test_index,],
                         newdat[-test_index,]^2,
                         newdat[-test_index,]^3))
y.train<-y[-test_index]
x.test<-as.matrix(cbind(newdat[test_index,],
                        newdat[test_index,]^2,
                        newdat[test_index,]^3))
```

Linear model - PCA and PLS practice

6. Build a linear model based on these components and non-numeric varibles.

```
v.lambda<-10^seq(-5,1,length=1000)# tao ra vector lambda cho
cv.out<-cv.glmnet(x.train, y.train, alpha = 1, nfolds = 5, lam
lasso.reg<-glmnet (x.train,y.train, alpha = 1,lambda = cv.out$
lasso.pred<-predict(lasso.reg , newx=x.test)
sqrt(mean((lasso.pred-y.test)^2))
```

```
## [1] 4.035644
```

```
v.lambda<-10^seq(-5,1,length=1000)# tao ra vector lambda cho
cv.out<-cv.glmnet(x.train, y.train, alpha = 2, nfolds = 5, lam
lasso.reg<-glmnet (x.train,y.train, alpha = 2,lambda = cv.out$
lasso.pred<-predict(lasso.reg , newx=x.test)
sqrt(mean((lasso.pred-y.test)^2))
```

```
## [1] 4.000053
```