Linear Model Selection and Regularization

Lu Haibo

Mondy, Mar 25, 2019

Introduction

There are two reasons why we are often not satisfied with the least squares estimate:

- *prediction accuracy*: the least squares estimates often have **low bias but large variance**.
- *interpretation*: with a large number of predictors, we often would like to determine a smaller subset that exhibit the strongest effects. **In order to get the "big picture", we are willing to sacrifice some of the small details**.

Some ways in which the simple linear model can be improved:

- *Subset Selection*
- *Shrinkage*
- *Dimension Reduction*

Subset Selection

With subset selection we retain only a subset of the variables, and eliminate the rest from the model.

Best Subset Selection

Best subset regression finds for each $k \in \{0, 1, 2, \ldots, p\}$ the subset of size k that gives smallest residual sum of squares (RSS).

Algorithm *Best subset selection*

---

1. Let $\mathcal{M}_0$ denote the *null model*, which contains no predictors. This model simply predicts the sample mean for each observation.
2. For $k = 1, 2, \ldots, p$:
   a). Fit all C_p^k models that contain exactly k predictors
   b). Pick the best among these C_p^k models, and call it $\mathcal{M}_k$. Here *best* is defined as having the smallest RSS, or equivalently largest R^2.
3. Select a single best model from among $\mathcal{M}_0, \ldots, \mathcal{M}_p$ using cross-validation, AIC(C_p), BIC, or adjusted R^2.

```
library(leaps)
library(ISLR)

data(Hitters)

Hitters <- Hitters %>% na.omit()
#AIC
reg.full <- regsubsets(Salary~., data=Hitters, nvmax = 19)
summary.full <- summary(reg.full)


tibble(
  x = 1:19,
  `Cp (AIC)` = summary.full$cp,
  BIC = summary.full$bic,
  RSS = summary.full$rss
) %>%
  gather(key, value, -x) %>%
  ggplot(aes(x, value)) + geom_point() + geom_line() +
  facet_wrap(~key, scales = "free_y")
```

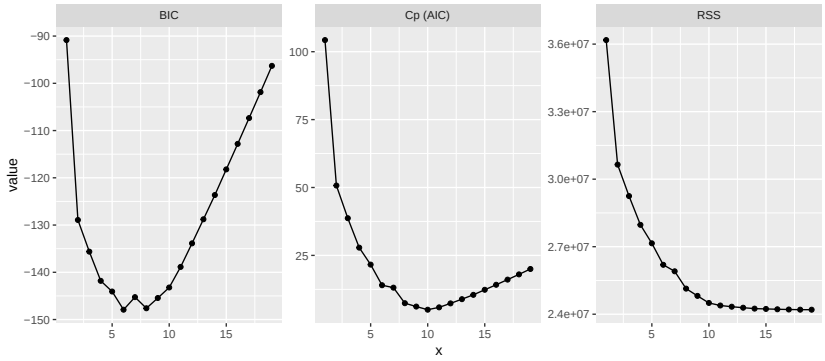


Figure 1: For the Hitters data set, three quantities are displayed for the best model containing d predictors for d ranging from 1 to 19.

Forward and Backward Stepwise Selection

Rather than search through all possible subsets (which becomes infeasible for p much larger than 40), we can seek a good path trough them.

Algorithm *Forward stepwise selection*

---

1. Let $\mathcal{M}_0$ denote the *null model*, which contains no predictors.
2. For $k = 0, 1, \ldots, p-1$:

Algorithm *Forward stepwise selection*

a). Consider all $p - k$ models that augment the predictors in $\mathcal{M}_k$ with one additional predictor.

b). Choose the *best* among these $p - k$ models, and call it $\mathcal{M}_{k+1}$. Here *best* is defined as having smallest RSS or highest R^2.

3. Select a single best model from among $\mathcal{M}_0, \ldots, \mathcal{M}_p$ using cross-validation, AIC(C_p), BIC, or adjusted R^2.

Algorithm *Backward stepwise selection*

1. Let $\mathcal{M}_p$ denote the *full model*, which contains all p predictors.

2. For $k = p, p-1, \ldots, 1$:

a). Consider all k models that contain all but one of the predictors in $\mathcal{M}_k$, for a total of $k - 1$ predictors.

b). Choose the *best* among these k models, and call it $\mathcal{M}_{k-1}$. Here *best* is defined as having smallest RSS or highest R^2.

3. Select a single best model from among $\mathcal{M}_0, \ldots, \mathcal{M}_p$ using cross-validation, AIC(C_p), BIC, or adjusted R^2.

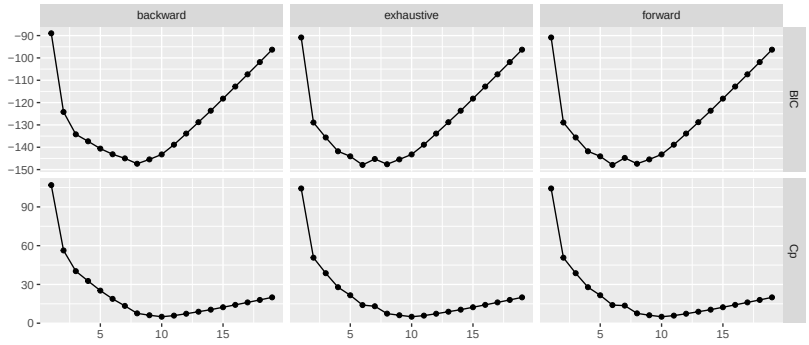
Forward and Backward stepwise selection is a *greedy algorithm*, producing a nested sequence of models. In this sense it might seem *sub-optimal* compared to best-subset election. However, there are several reasons why it might be preferred:

- *Computational*: for large p we cannot compute the best subset sequence, but we can always compute the forward stepwise sequence (even when $p \gg N$) [1]
- *Statistical*: forward and backward stepwise is a *more constrained* search, and will have low variance, but perhaps more bias.

[1] Backward selection can only be used when $p < N$, while forward stepwise can always be used.

```
tibble(
  method = c("exhaustive", "forward", "backward")
) %>%
  mutate(
    model = method %>%
      map(~regsubsets(Salary~., data=Hitters, nvmax = 19, method = .)) %>%
      map(summary),
    Cp = model %>% map("cp"),
    BIC = model %>% map("bic"),
    vars = Cp %>% map(~1:19)
```

```
) %>%
unnest(Cp, BIC, vars) %>%
gather(key, value, Cp, BIC) %>%
ggplot(aes(x = vars, y = value)) + geom_line() + geom_point() +
facet_grid(key~method, scales = "free_y") +
labs(x = "", y = "")
```



Shrinkage Methods

By retaining a subset of the predictors and discarding the rest, subset selection produces a model that is *interpretable* and possibly lower prediction error than the full model.

However, because it is a discrete process — variables are either retained or discarded – it often exhibits high variance, and so doesn't reduce the prediction error of the full model.

Shrinkage methods are more continous, and don't suffer as much from high variability.

Ridge Regression

Ridge regression shrinks the regression coefficients by imposing a penalty on their size.

- Least squares estimate

$$RSS = \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 .$$

- *Ridge regression*

$$\sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 = RSS + \lambda \sum_{j=1}^{p} \beta_j^2,$$

  where $\lambda \geq 0$ is a *tuning parameter*, to be determined separately.

- $\lambda \sum_{j=1}^{p} \beta_j^2$, called a *shrinkage penalty*, it has the effect of *shrinking* the estimates of β_j towards zero
- when $\lambda = 0$, the penalty term has no effect.
- when $\lambda \to \infty$, $\beta_j = 0$, $j = 1, 2, \ldots, p$.

An equivalent way to write the ridge problem is

$$\hat{\beta}^{ridge} = arg \min_{\beta} \sum_{i=1}^{N} (y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2,$$
$$\text{subject to} \sum_{j=1}^{p} \beta_j^2 \leq t.$$

The Lasso

The penalty $\lambda \sum \beta_j^2$ will shrink all of the coefficients towards zero, but it **will not set any of them exactl to zero** (unless $\lambda = \infty$)

This may not be a problem for prediction accuracy, but it can create a challenge in model interpretation in settings in which the number of variables p is quite large.

The *lasso*:

$$\sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^{p} |\beta_j| = RSS + \lambda \sum_{j=1}^{p} |\beta_j|.$$

- the L_1 penalty has the effect of forcing some of the coefficient estimates to **be exactly equal to zero** when the tuning parameter λ is sufficiently large.
- the lasso performs *variable selection* (the lasso yields *sparse* models)

An equivalent way to write the lasso problem is

$$\hat{\beta}^{lasso} = arg \min_{\beta} \sum_{i=1}^{N} (y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2,$$
$$\text{subject to} \sum_{j=1}^{p} |\beta_j| \leq t.$$

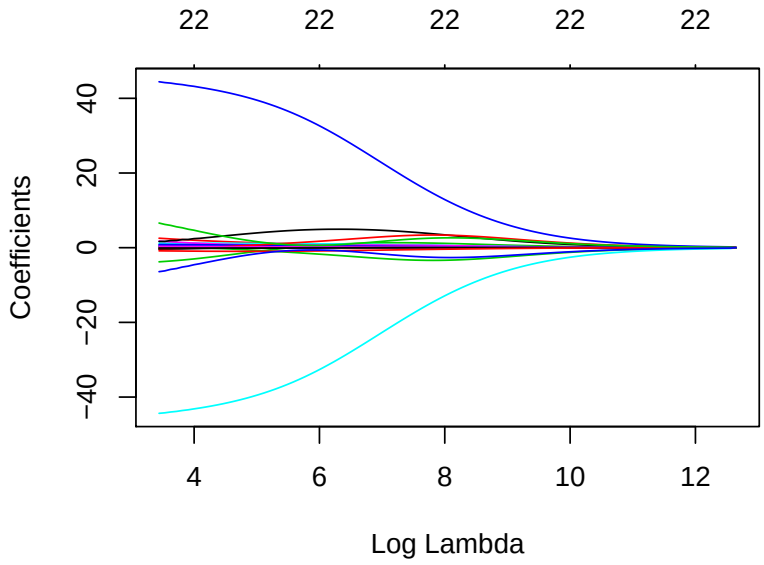
Example

```
set.seed(1)
library(rsample)
library(glmnetUtils)

Hitters <- Hitters %>% initial_split(prop = 0.7)
```

```
#1. ridge
ridge_fit <- glmnet(Salary ~ ., data = training(Hitters), alpha = 0)

plot(ridge_fit, xvar = "lambda")
```



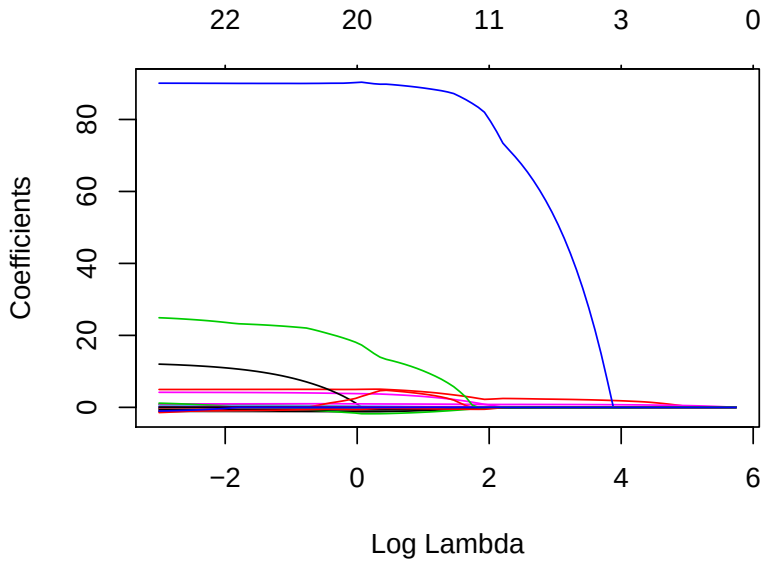

```
ridge_cv <- cv.glmnet(Salary ~ ., data = training(Hitters), alpha = 0)

ridge_pred <- predict(ridge_fit, testing(Hitters), s = ridge_cv$lambda.min)
#test mse for ridge
mse(testing(Hitters) %>% pull(Salary), ridge_pred)
```

```
## [1] 173273.5
```

```
#2. lasso
lasso_fit <- glmnet(Salary ~ ., data = training(Hitters), alpha = 1)

plot(lasso_fit, xvar = "lambda")
```




```
lasso_cv <- cv.glmnet(Salary ~ ., data = training(Hitters), alpha = 1)

lasso_pred <- predict(lasso_fit, testing(Hitters), s = lasso_cv$lambda.min)
#test mse for lasso
mse(testing(Hitters) %>% pull(Salary), lasso_pred)
```

```
## [1] 183483
```

```
#3. best subset selection

subsets_fit <- regsubsets(Salary ~ ., data = training(Hitters))

subsets_fit %>% summary %>% .$cp %>% which.min
```

```
## [1] 8
```

```
coef(subsets_fit, id = 8)
```

```
## (Intercept)       AtBat        Hits
##  75.2218309  -1.6121104   5.8818414
##       Walks       CRuns        CRBI
##   4.3187709   0.5138557   0.9199649
##      CWalks   DivisionW     PutOuts
##  -0.8586009 -91.6139745   0.2190869
```

```
subsets_pred <- lm(Salary ~ Walks + CAtBat + CHits + CHmRun + Division + PutOuts,
                   data = training(Hitters)) %>%
```

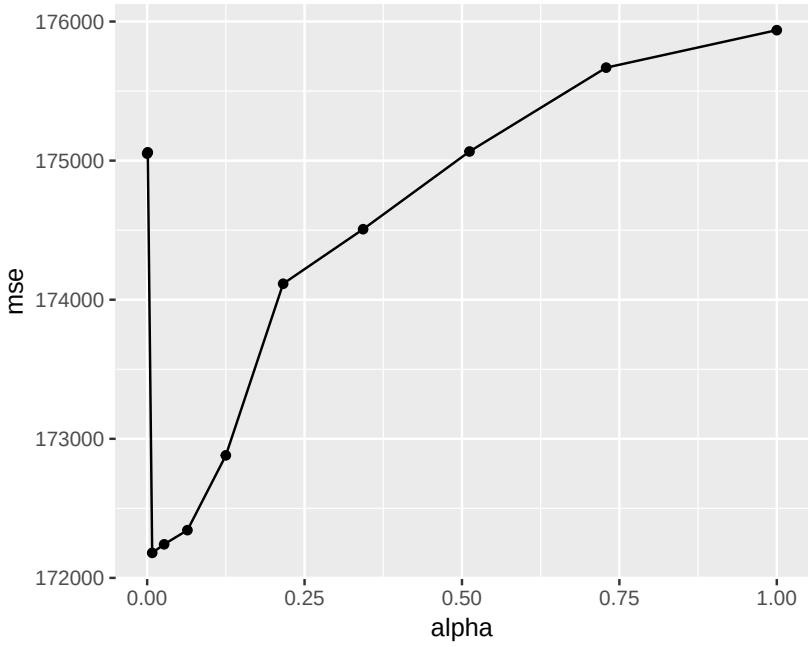
```
  predict(newdata = testing(Hitters))

#test mse for best subset
mse(testing(Hitters) %>% pull(Salary), subsets_pred)

## [1] 159595.4

#4. elastic net
cva <- cva.glmnet(Salary ~., data = training(Hitters))

tibble(
  alpha = cva$alpha,
  lambda = cva$modlist %>% map_dbl("lambda.min")
) %>%
  mutate(
    model = alpha %>% map(~glmnet(Salary ~., data = training(Hitters), alpha = .)),
    pred = map2(model, lambda, ~predict(.x, testing(Hitters), s = .y)),
    mse = pred %>% map_dbl(~mean((testing(Hitters) %>% .$Salary - .x)^2))
  ) %>%
  ggplot(aes(alpha, mse)) + geom_line() + geom_point()
```



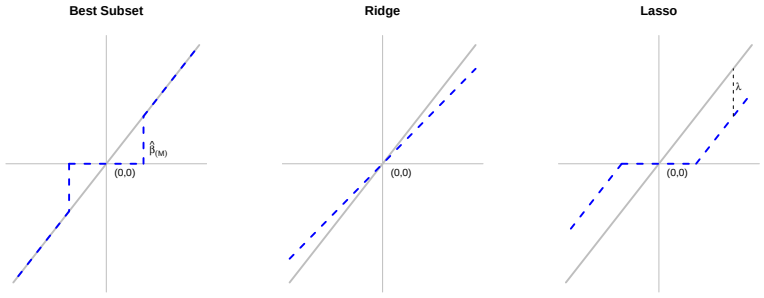
Discussion: Subset Selection, Ridge Regression and the Lasso

- Best subset selection drops all variables with coefficient smaller than the Mth largest. This is a form of "hard-thresholding"

Estimator	Formula
Best subset (size M)	$\hat{\beta}_j \cdot I(\lvert\hat{\beta}_j\rvert \geq \lvert\hat{\beta}_{(M)}\rvert)$
Ridge	$\hat{\beta}_j/(1+\lambda)$
Lasso	$\text{sign}(\hat{\beta}_j)(\lvert\hat{\beta}_j\rvert - \lambda)_+$

Table 4: In the case of an orthonormal input matrix X the three procedures have explicit solutions. Each method applys a simple transformation to the least squares estimate $\hat{\beta}_j$.

- Ridge regression does a proportional shinkage
- Lasso translates each coefficient by a constant factor λ, *truncating* at zero. This is called "soft thresholding"



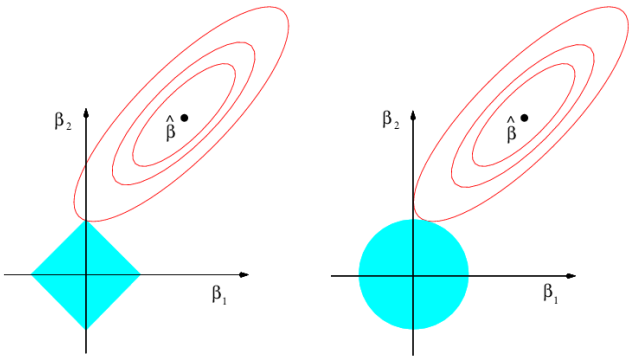


Figure 2: Estimation picture for the lasso(left) and ridge regression (right). Shown are contours of the error and constraint functions. The solid blue areas are the constraint regions $|\beta_1| + |\beta_2| \leq t$ and $\beta_1^2 + \beta_2^2 \leq t^2$, respectively, while the red ellipses are the contours of the least squares error functions.

We can generalize ridge regression and the lasso:

$$\hat{\beta} = arg\min_{\beta} \left\{ \sum_{i=1}^{N} (y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j|^q \right\}$$

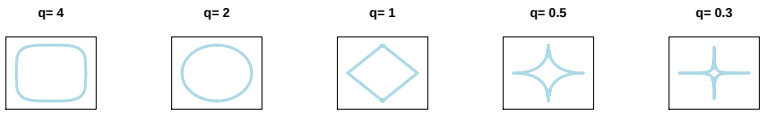


Figure 3: Contours of constant values of $\sum_j |\beta_j|^q$ for given values if q

- The value $q = 0$ correspnds to variable subset selection, as the penalty simply counts the number of nonzero parameters.
- The case $q = 1$ (lasso) is the smallest q such that the constraint region is *convex*; non-convex constraint regions make the optimization problem more difficult.

Dimension Reduction Methods

The methods that we have discussed so far in this chapter have controlled variance in two different ways

- using a subset of the original variables
- by shrinking their coefficients toward zero

Both of these methods are defined using the original predictors, $X_1, X_2, \ldots, X_p$.

We now explore a class of approaches that *transform the predictors* and then fit a least squares model using the transformed variables. We will refer to these techniques as *dimension reduction methods*.

Let $Z_1, Z_2, \ldots, Z_M$ represent M ($< p$) *linear combinations* of our original p predictors. That is,

$$Z_m = \sum_{j=1}^{p} \phi_{jm} X_j,$$

for some constants $\phi_{1m}, \phi_{2m}, \ldots, \phi_{pm}$, $m = 1, \ldots, M$. We can then fit the linear regression model

$$y_i = \theta_0 + \sum_{m=1}^{M} \theta_m z_{im} + \epsilon_i, \qquad i = 1, \ldots, n.$$

- If the constants $\phi_{1m}, \phi_{2m}, \ldots, \phi_{pm}$ are chosen wisely, the such dimension reduction approaches can often outperform least squares regression.
- Dimension reduction serves to contrain the estimated β_j coefficients:[2]

  $$\beta_j = \sum_{m=1}^{M} \theta_m \phi_{jm}$$

  where $y_i = \beta_0 + \sum_{i=1}^{p} \beta_j x_{ij} + \epsilon_i$.

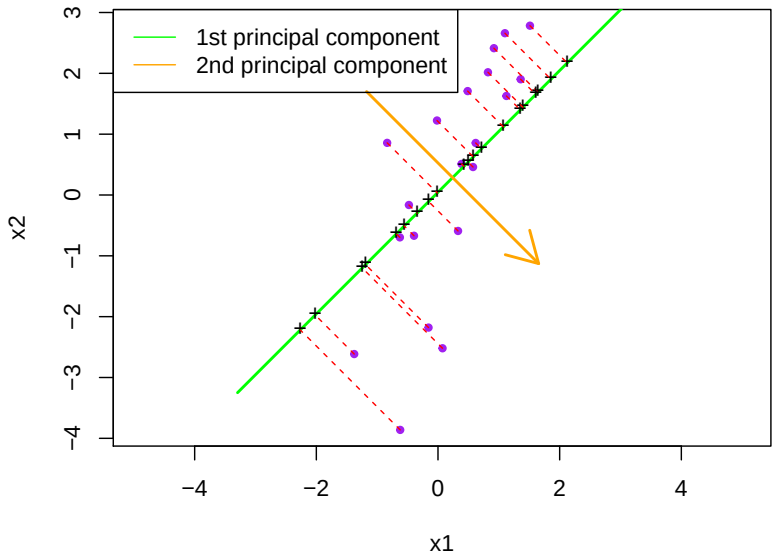
[2] This constraint on the form of the coefficients has the potential to bias the coefficient estimates. However, in situations where p is large relative to n, selecting a value of $M \ll p$ can significantly reduce the variance of the fitted coefficients.

Principal Components Regression

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

PCA is a technique for reducing the dimension of a $n \times p$ data matrix X. The *first principle component* direction[3] of the data is that along which the observations *vary the most*.

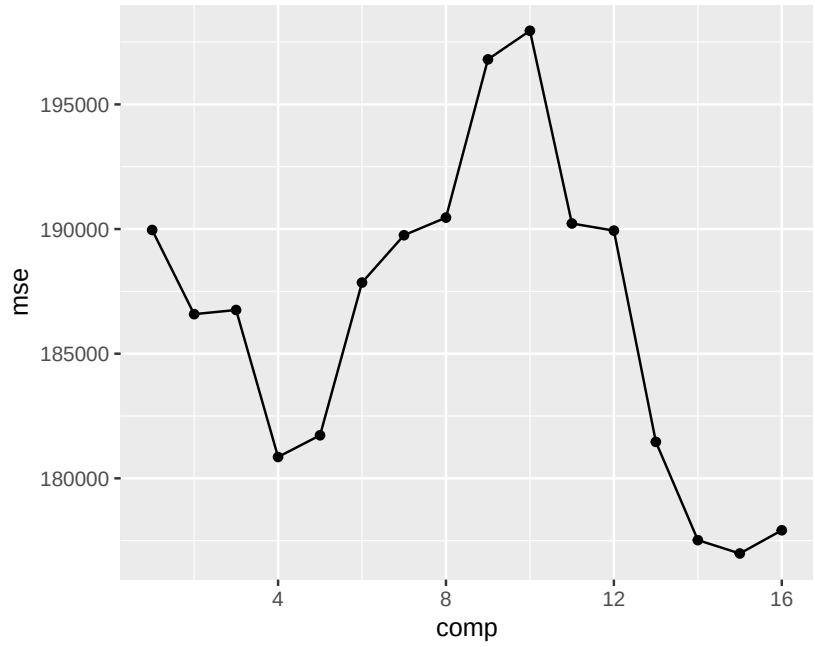
[3] It can be computed by the eigen vector of the covariance matrix of X.




```
library(pls)

pcr.fit <- pcr(Salary ~ . - League - Division - NewLeague,
               data=training(Hitters), scale =T)

tibble(
  comp = 1:16
) %>%
  mutate(
    pred = comp %>% map(~predict(pcr.fit, testing(Hitters), .)),
    mse = pred %>% map_dbl(~mse(testing(Hitters) %>% pull(Salary), .))
  ) %>%
  ggplot(aes(comp, mse)) + geom_line() + geom_point()
```



Example: PCA on Face Feature Selection

```
load("data/face.Rd")
par(mfrow=c(1,3))
for(i in 1:3)
  {
    im <- matrix(data=rev(im.train[i,]), nrow=96, ncol=96)
    image(1:96, 1:96, im, col=gray((0:255)/255))
  }
```

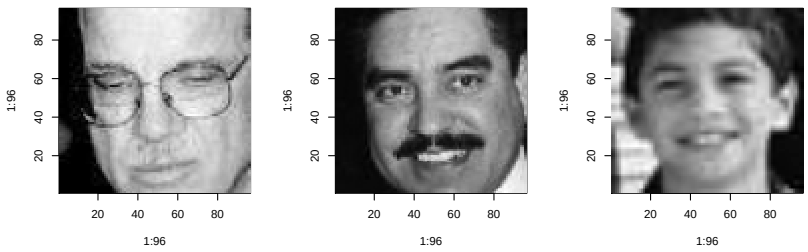


Figure 4: 3 faces saved by 96 × 96 pixels.

```
pc <- svd(scale(im.train))
eig.perc <- cumsum((pc$d)^2)/sum((pc$d)^2)
eig.perc[1:10]
```

```
##  [1] 0.3491269 0.4453086 0.5332194 0.5849549
```

```
##  [5] 0.6175806 0.6428016 0.6598505 0.6755274
##  [9] 0.6907955 0.7026048
```

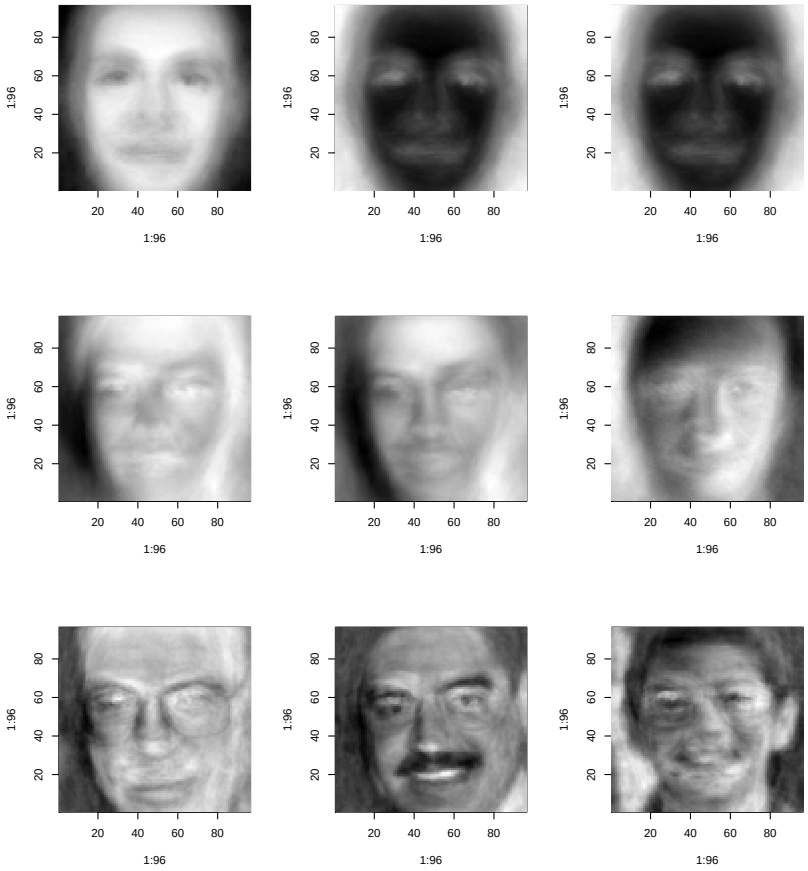


Figure 5: Top: 3 faces with the 1st principle component; Middle: 3 faces with first 10 principle components; Bottom: 3 faces with first 200 principle components.