AMS 597: Statistical Computing

Pei-Fen Kuan (c)

Applied Math and Stats, Stony Brook University

Linear models

- Many data sets are inherently too complex to be handled adequately by standard procedures and thus require the formulation of ad hoc models.
- The class of linear models provides a flexible framework into which many — although not all — of these cases can be fitted.
- Note: To implement the examples in this handout, we need to install the package `ISwR`

```
library(ISwR)
```

Polynomial regression

- One most straightforward extension to simple linear regression to multiple regression analysis is to include second-order and higher powers of a variable in the model along with the original linear term. That is, you can have a model like

$$y = \alpha + \beta_1 x + \beta_2 x^2 + ... + \beta_k x^k + \epsilon$$

- This obviously describes a nonlinear relation between y and x, but LS estimation is still the same since the model is still a linear model.

Polynomial regression

```
attach(cystfibr)
summary(lm(pemax ~ poly(height, 2, raw = TRUE)))
```

```
##
## Call:
## lm(formula = pemax ~ poly(height, 2, raw = TRUE))
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -51.411 -14.932  -2.288  12.787  44.933
##
## Coefficients:
##                               Estimate Std. Error t value P
## (Intercept)                  615.36248  240.95580   2.554
## poly(height, 2, raw = TRUE)1  -8.08324    3.32052  -2.434
## poly(height, 2, raw = TRUE)2   0.03064    0.01126   2.721
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 '
```

Polynomial regression

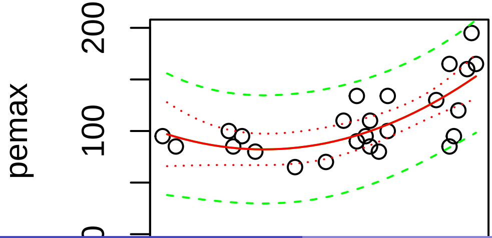
```r
pred.frame <- data.frame(height = seq(110, 180, 2))
lm.pemax.hq <- lm(pemax ~ height + I(height^2))
predict(lm.pemax.hq, interval = "pred", newdata = pred.frame)
```

```
##         fit      lwr      upr
## 1  96.90026 37.94461 155.8559
## 2  94.33611 36.82985 151.8424
## 3  92.01705 35.73077 148.3033
## 4  89.94307 34.66449 145.2217
## 5  88.11418 33.65007 142.5783
## 6  86.53038 32.70806 140.3527
## 7  85.19166 31.85979 138.5235
## 8  84.09803 31.12689 137.0692
## 9  83.24949 30.53064 135.9683
## 10 82.64604 30.09150 135.2006
## 11 82.28767 29.82873 134.7466
## 12 82.17439 29.76004 134.5887
## 13 82.30620 29.90143 134.7110
```

Polynomial regression

- Based on the predicted data, we have the plot

```
pp <- predict(lm.pemax.hq, newdata = pred.frame,
    interval = "pred")
pc <- predict(lm.pemax.hq, newdata = pred.frame,
    interval = "conf")
plot(height, pemax, ylim = c(0, 200))
matlines(pred.frame$height, pp, lty = c(1, 2,
    2), col = "green")
matlines(pred.frame$height, pc, lty = c(1, 3,
    3), col = "red")
```



Polynomial regression

- Exercise: For the same dataset, fit the response to a cubic polynomial of the dependent variable. Plot the 95% predicted and confidence bands.
- Which model (quadratic or cubic) fits the data better?

Design matrices and dummy variables

- The function `model.matrix` gives the design matrix for a given model. For example:

```
model.matrix(pemax ~ height + weight)
```

```
##    (Intercept) height weight
## 1            1    109   13.1
## 2            1    112   12.9
## 3            1    124   14.1
## 4            1    125   16.2
## 5            1    127   21.5
## 6            1    130   17.5
## 7            1    139   30.7
## 8            1    150   28.4
## 9            1    146   25.1
## 10           1    155   31.5
## 11           1    156   39.9
## 12           1    153   42.1
```

Design matrices and dummy variables

- If the same is attempted for a model containing a factor, the following happens.

```r
attach(red.cell.folate)
model.matrix(folate ~ ventilation)
```

```
##    (Intercept) ventilationN2O+O2,op ventilationO2,24h
## 1            1                    0                 0
## 2            1                    0                 0
## 3            1                    0                 0
## 4            1                    0                 0
## 5            1                    0                 0
## 6            1                    0                 0
## 7            1                    0                 0
## 8            1                    0                 0
## 9            1                    1                 0
## 10           1                    1                 0
## 11           1                    1                 0
```

Design matrices and dummy variables

- The two columns of zeros and ones are sometimes called *dummy variables*. They are interpreted exactly as above: Multiplying them by the respective regression coefficients and adding the results yields the fitted value.

Linearity over groups

- Suppose the group variable is an ordinal variable, e.g., age group
- In such case, there are two possible models, i.e., (a) fitting a linear regression by treating the group variable as a numeric variable (in this case, we assume linearity over groups) (b) treating the group variable as nominal by fitting one way ANOVA
- In the following example, notice that the variable `grp` is a numeric vector, and the variable `grpf` is a factor with six levels.

Linearity over groups

```
attach(fake.trypsin)
str(fake.trypsin)
```

```
## 'data.frame':    271 obs. of  3 variables:
##  $ trypsin: num  137.3 87 82.4 127.2 123.5 ...
##  $ grp    : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ grpf   : Factor w/ 6 levels "1","2","3","4",..: 1 1 1 1
```

```
summary(fake.trypsin)
```

```
##     trypsin              grp         grpf
##  Min.   :-39.96   Min.   :1.000   1: 32
##  1st Qu.:119.52   1st Qu.:2.000   2:137
##  Median :167.59   Median :2.000   3: 38
##  Mean   :168.68   Mean   :2.583   4: 44
##  3rd Qu.:213.98   3rd Qu.:3.000   5: 16
##  Max.   :390.13   Max.   :6.000   6:  4
```

Linearity over groups

```
anova(lm(trypsin ~ grpf))
```

```
## Analysis of Variance Table
##
## Response: trypsin
##            Df Sum Sq Mean Sq F value    Pr(>F)
## grpf        5 224103   44821  13.508 9.592e-12 ***
## Residuals 265 879272    3318
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 '
```

Linearity over groups

```r
anova(lm(trypsin ~ grp))
```

```
## Analysis of Variance Table
##
## Response: trypsin
##            Df Sum Sq Mean Sq F value    Pr(>F)
## grp         1 206698  206698  62.009 8.451e-14 ***
## Residuals 269 896677    3333
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 '
```

Linearity over groups

- Notice that the residual mean squares did not change very much, indicating that the two models describe the data nearly equally well.
- We can compare the simple linear model against the model where there is a separate mean for each group using a formal test as follows:

Linearity over groups

```
model1 <- lm(trypsin ~ grp)
model2 <- lm(trypsin ~ grpf)
anova(model1, model2)
```

```
## Analysis of Variance Table
##
## Model 1: trypsin ~ grp
## Model 2: trypsin ~ grpf
##   Res.Df    RSS Df Sum of Sq      F Pr(>F)
## 1    269 896677
## 2    265 879272  4     17405 1.3114 0.2661
```

Linearity over groups

- So we see that the model reduction has a non-significant p-value and hence that `model2` does not fit data significantly better than `model1`
- This technique works only when one model is a submodel of the other, which is the case here since the linear model is defined by a restriction on the group means

Multicollinearity

- One possible way to detect multicollinearity is to use the variance inflation factor VIF
- The VIF_k for the k-the predictor is defined as

$$VIF_k = \frac{1}{1 - R_k^2}$$

  where R_k^2 is the r^2 value obtained by regressing the k-th predictor on the remaining predictors
- VIF_k of 1 means that there is no correlation among the predictor and the remaining predictor variables, and hence the variance of is not inflated
- $VIF_k > 4$ warrants further investigation
- $VIF_k > 10$ indicates severe multicollinearity requiring correction

Multicollinearity

```r
# multicollinearity
set.seed(123)
n <- 50
x1 <- rnorm(n)
x2 <- rnorm(n)
x3 <- x1 + x2 + rnorm(n, sd = 0.2)
x4 <- rnorm(n)
y <- x1 - x2 - x3 + x4 + rnorm(n)
cor(cbind(x1, x2, x3, x4))
```

```
##             x1          x2         x3         x4
## x1  1.00000000 -0.03586983  0.7084494 -0.1223944
## x2 -0.03586983  1.00000000  0.6629116 -0.1562637
## x3  0.70844943  0.66291161  1.0000000 -0.2019738
## x4 -0.12239437 -0.15626371 -0.2019738  1.0000000
```

Multicollinearity

```r
fit1 <- lm(y ~ x1 + x2 + x3 + x4)
library(car)
vif(fit1)
```

```
##        x1        x2        x3        x4
## 23.682775 21.045086 42.232951  1.043507
```

```r
fit2 <- lm(y ~ x1 + x2 + x4)
vif(fit2)
```

```
##       x1       x2       x4
## 1.018414 1.028266 1.042561
```

Multicollinearity

- Possible remedies:
- Subset selection (e.g., previous example based on correlation matrix)
- Shrinkage estimators: Ridge and Lasso (regularization)

Ridge regression

- The ordinary linear square (OLS) method estimates β_j's by minimizing
$$RSS = \sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij})^2$$
- Ridge regression estimates β_j's by minimizing
$$RSS + \lambda \sum_{j=1}^{p} \beta_j^2$$
where $\lambda \geq 0$ is the tuning parameter
- Ridge regression belongs to the class of penalized regression framework
- The penalty term in ridge regression is also known as L_2 penalty
- As $\lambda \to 0$ $\hat{\beta}_{ridge} \to \hat{\beta}_{OLS}$
- As $\lambda \to \infty$ $\hat{\beta}_{ridge} \to 0$

Ridge regression

- The advantage of ridge regression over OLS can be explained by the bias-variance trade-off
- The OLS estimates have high variance but zero bias
- As λ increases, the shrinkage of the ridge coefficient estimates leads to a substantial reduction in the variance of the predictions, at the expense of a slight increase in bias
- For complicated data, lower MSE can be achieved at $\lambda > 0$

$$MSE = Variance + Bias^2$$

- Most importantly, if $p > n$, then OLS estimates do not have a unique solution, whereas ridge regression can still perform well by trading off a small increase in bias for a large decrease in variance

Ridge regression

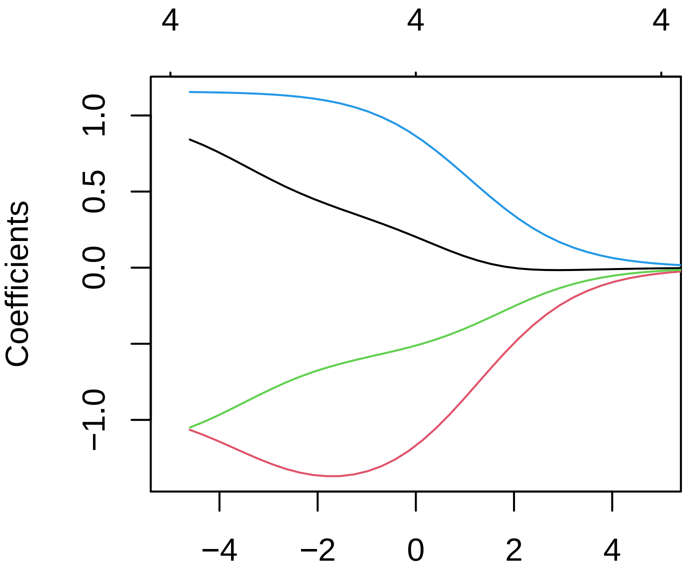
```r
library(glmnet)
```

```
## Warning: package 'glmnet' was built under R version 4.0.2
```

```r
x <- cbind(x1, x2, x3, x4)
lambda <- 10^seq(10, -2, length = 100)
# alpha=0 is for ridge regression, alpha=1 for
# lasso
fit3 <- glmnet(x, y, alpha = 0, lambda = lambda)
```

Ridge regression

```
plot(fit3, xvar = "lambda", xlim = c(-5, 5))
```



Ridge regression

```r
coef(glmnet(x, y, alpha = 0, lambda = 0.01))
```

```
## 5 x 1 sparse Matrix of class "dgCMatrix"
##                        s0
## (Intercept)  0.006972247
## x1           0.842351766
## x2          -1.064960908
## x3          -1.050266103
## x4           1.154041794
```

Ridge regression

```r
coef(glmnet(x, y, alpha = 0, lambda = 100))
```

```
## 5 x 1 sparse Matrix of class "dgCMatrix"
##                       s0
## (Intercept) -0.201065732
## x1          -0.005968152
## x2          -0.054014382
## x3          -0.030556099
## x4           0.036135903
```

Ridge regression

```
coef(glmnet(x, y, alpha = 0, lambda = 1e+05))
```

```
## 5 x 1 sparse Matrix of class "dgCMatrix"
##                        s0
## (Intercept) -2.117390e-01
## x1          -6.917426e-06
## x2          -5.620034e-05
## x3          -3.215885e-05
## x4           3.746404e-05
```

Lasso regression

- Lasso regression estimates β_j's by minimizing

$$RSS + \lambda \sum_{j=1}^{p} |\beta_j|$$

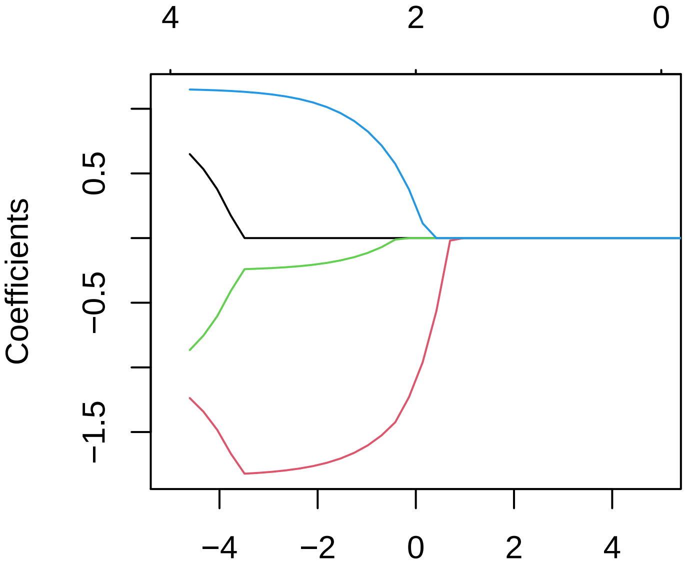
  where $\lambda \geq 0$ is the tuning parameter
- The penalty term in lasso regression is also known as L_1 penalty
- As with ridge regression, the lasso shrinks the coefficient estimates towards zero
- However, in the case of the lasso, the L_1 penalty has the effect of forcing some of the coefficient estimates to be exactly equal to zero when λ is sufficiently large
- Thus, lasso performs variable selection

Lasso regression

```r
library(glmnet)
x <- cbind(x1, x2, x3, x4)
lambda <- 10^seq(10, -2, length = 100)
# alpha=0 is for ridge regression, alpha=1 for
# lasso
fit4 <- glmnet(x, y, alpha = 1, lambda = lambda)
```

Lasso regression

```
plot(fit4, xvar = "lambda", xlim = c(-5, 5))
```



Elastic net regularization

- Although lasso can perform variable selection, it has some limitations
- In "large p, small n" case (high-dimensional data with few examples), the LASSO selects at most n variables before it saturates
- If there is a group of highly correlated variables, then the LASSO tends to select one variable from a group and ignore the others
- To overcome these limitations, the elastic net regularization is proposed which estimates β_j's by minimizing

$$RSS + \lambda_1 \sum_{j=1}^{p} |\beta_j| + \lambda_2 \sum_{j=1}^{p} \beta_j^2$$

Cross-validation

- The cross validation (CV) is a general framework used to tune λ's in the penalized regression
- For example in $K = 5$ fold CV, the data is splitted into 5 roughly equal-sized parts

1	2	3	4	5
Train	Train	Validation	Train	Train

- For the kth part (third above), we fit the model to the other $K - 1$ parts of the data
- W calculate the MSE of the fitted model when predicting the kth part of the data
- We do this for $k = 1, 2, ..., K$ and combine the K estimates of prediction error.

Cross-validation

```r
set.seed(123)
cv.out <- cv.glmnet(x, y, alpha = 1, nfolds = 5)
cv.out
```

```
##
## Call:  cv.glmnet(x = x, y = y, nfolds = 5, alpha = 1)
##
## Measure: Mean-Squared Error
##
##      Lambda Index Measure     SE Nonzero
## min 0.00228    74   1.026 0.1591       4
## 1se 0.26174    23   1.162 0.2238       3
```

Cross-validation

```
plot(cv.out)
```