기계학습 기초 - 2

2016 하계 통계 Workshop

고려대학교 통계학과

신승준 (sjshin@korea.ac.kr)

2016.7.15

목차

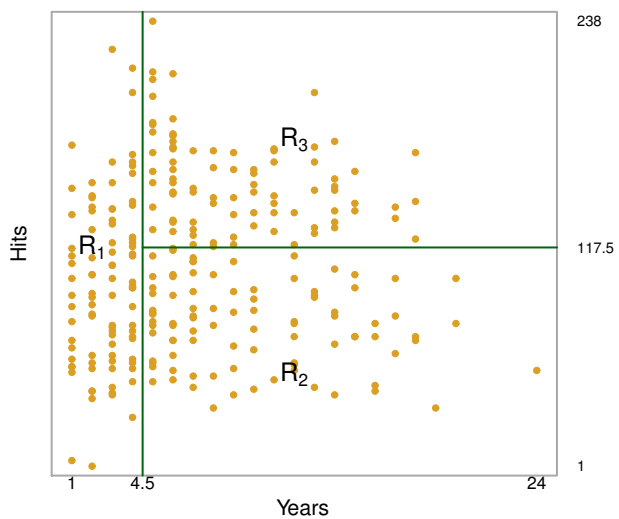
목차

나무모형의 종류

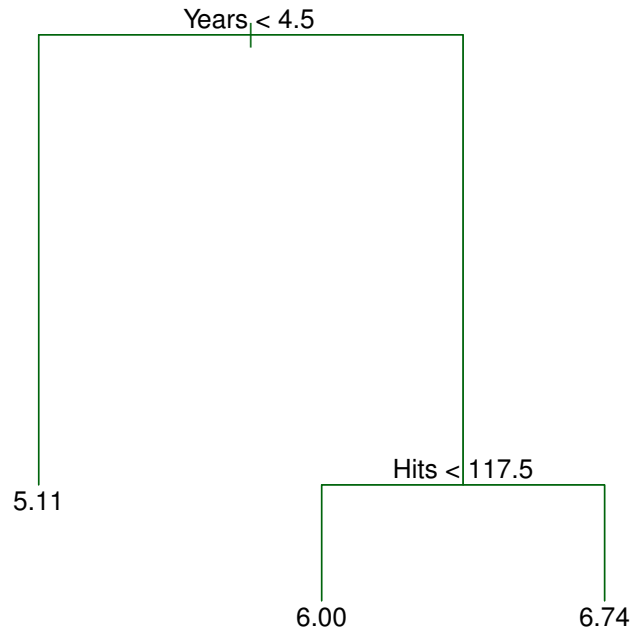
- 나무모형의 종류
  - 회귀 나무 모형 (regression tree)
  - 분류 나무 모형 (classification tree)
- 앙상블 (Ensemble) 방법
  - Bagging
  - Random Forest

회귀 나무 모형

- 반응변수가 연속형.
- 회귀 나무 모형의 알고리즘.
  - 독립변수 $\mathbf{x}$ 공간을 J개의 파티션 (partition), $R_1, \cdots, R_J$ 으로 나눈다.
  - 만약 $\mathbf{x}_0 \in R_j$라면, $\hat{y}_0 = \frac{1}{|R_j|} \sum_{i \in R_j} y_i$로 예측한다.



(a) 파티션



(b) 회귀나무모형

- 최적의 파티션 $(R_1, \cdots, R_J)$은 다음과 같이 정의된다.

$$\underset{R_1,\cdots,R_J}{\text{argmin}} \sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \bar{y}_{R_j})^2$$

  여기서, $\bar{y}_{R_j} = \frac{1}{|R_j|} \sum_{i \in R_j} y_i$.

- J 역시 알려져 있지 않다.

- J가 고정되어 있더라도, 풀 수 없는 문제이다.

회귀 나무 모형의 적합: Recursive Binary Splitting

1. 한번에 한개의 독립변수를 두지역으로만 나눔. 즉, j번 째 변수에 대하여 다음과 같은 binary 파티션을 고려.

$$R_-(j,s) = \{\mathbf{x}|x_j < s\} \text{ and } R_+(j,s) = \{\mathbf{x}|x_j \geq s\}$$

2. 해당 binary 파티션에 대하여 잔차 제곱합을 계산.

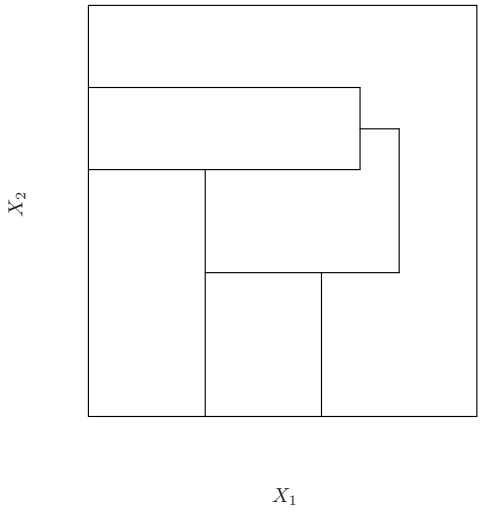
$$RSS(j,s) = \sum_{i:\mathbf{x}_i \in R_+(j,s)} (y_i - \bar{y}_+)^2 + \sum_{i:\mathbf{x}_i \in R_-(j,s)} (y_i - \bar{y}_-)^2$$

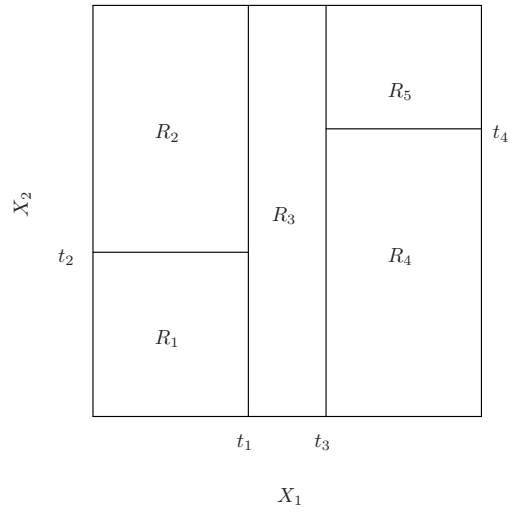
3. 최적의 binary 파티션:

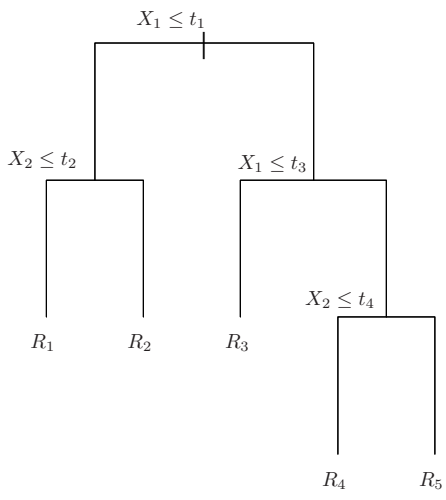
$$(\hat{j}, \hat{s}) = \underset{(j,s)}{\operatorname{argmin}}\, RSS(j,s)$$

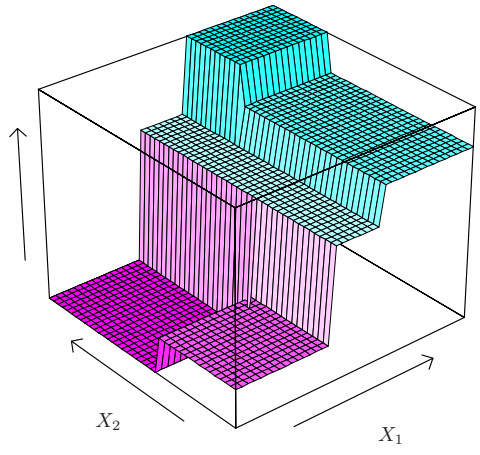
- 1-3단계를 각 파티션마다 반복.

회귀 나무 모형의 적합: Recursive Binary Splitting









가지치기 (Pruning Tree)

- training error를 최소화 하는 방법은 training set에 있는 각각의 개체가 하나의 파티션을 이루도록 하는 것. $(R_1, \cdots, R_n)$
- 편향-분산 트레이드 오프로 인해 과적합 (overfitting) 발생. (KNN 에서 $K = 1$인 경우와 같음)
- 해결방법: 우선 충분히 많은 파티션을 이용하여 나무 모형을 적합한 후, 가지치기 (pruning)를 하여 subtree를 선택.

- T_0를 Full tree라 하자.
- 주어진 α에 대하여,

$$\underbrace{\sum_{m=1}^{|T|} \sum_{\mathbf{x}_i \in R_m} (y_i - \bar{y}_{R_m})^2}_{\text{loss}} + \underbrace{\alpha |T|}_{\text{penalty}}$$

  을 최소화 시키는 나무모형 T ($\subseteq T_0$) 모형을 선택.

- α는 조율모수로 나무모형의 complexity를 결정.
  - $\alpha = 0 \;\Rightarrow\; T = T_0$ (overfitting)
  - $\alpha = \infty \;\Rightarrow\; T = \phi$ (underfitting)
- 조율모수 α는 CV를 통해 결정.

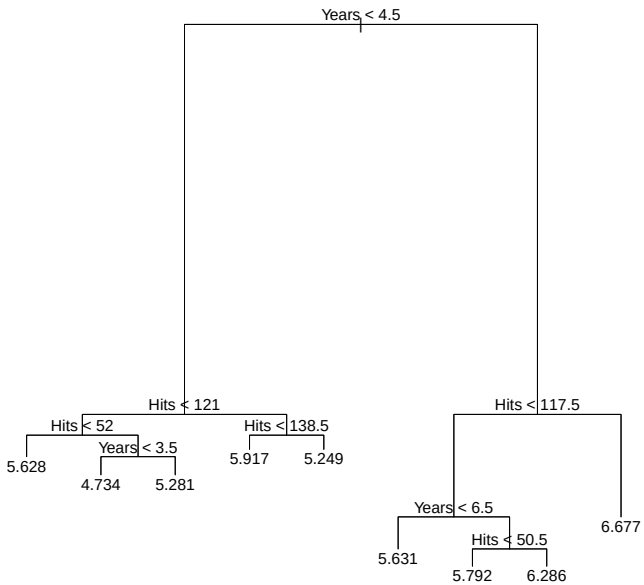
  (cross-validated 잔차제곱합을 최소화시키는 α를 선택)

회귀 나무 모형 요약

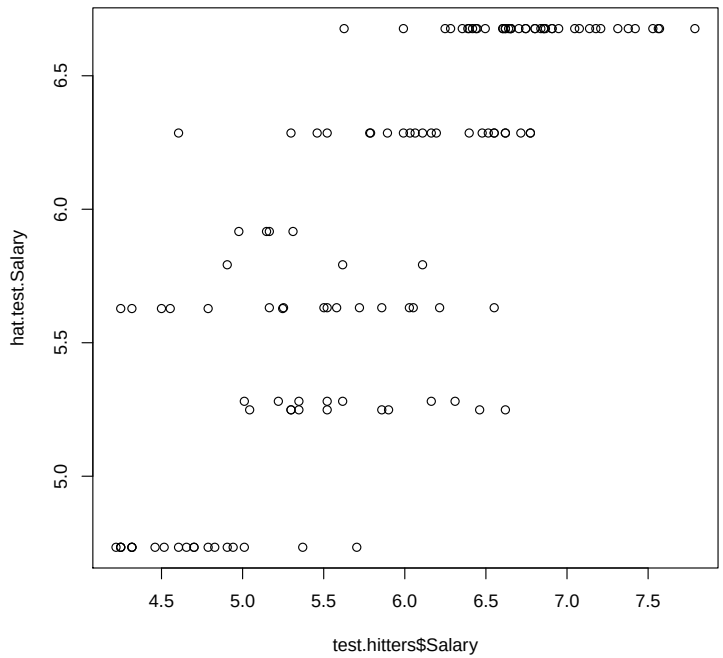
1. Recursive binary splitting을 통해 full tree T_0를 적합
2. 교차타당성 검증 (CV)을 통해 최적의 조율모수 α를 선택하고, 그에 해당하는 pruned tree, T를 최종 모형으로 선택.

```
> train.obj <- tree(Salary ~., train.hitters)
> summary(train.obj)

Regression tree:
tree(formula = Salary ~ ., data = train.hitters)
Number of terminal nodes:  9
Residual mean deviance:  0.3282 = 42.01 / 128
Distribution of residuals:
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-2.17700 -0.30080 -0.02386  0.00000  0.31770  2.03500

> plot(train.obj)
> text(train.obj, pretty = 0)
>
> hat.test.Salary <- predict(train.obj, newdata = test.hitters)
> plot(test.hitters$Salary, hat.test.Salary)
```

예제: 05_tree_reg_hitters.R



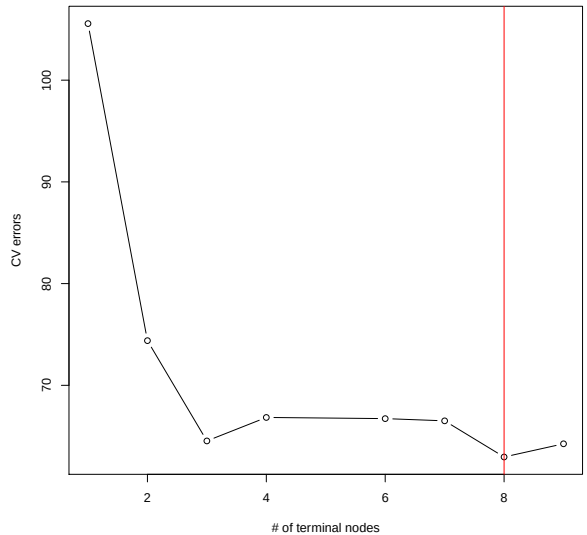
(c) Full Regression Tee T_0

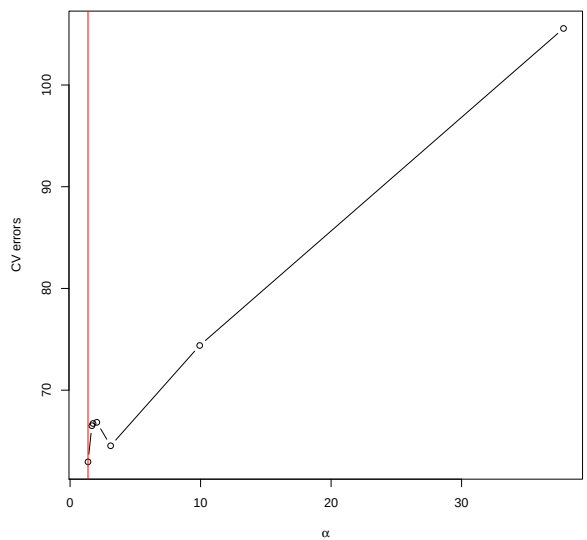


(d) y_i vs $\hat{y}_i$

예제: 05_tree_reg_hitters.R

```
# cross validation
cv.obj <- cv.tree(train.obj)
size  <- cv.obj$size
error <- cv.obj$dev
alpha <- cv.obj$k

best.size <- size[which.min(error)]
best.alpha <- alpha[which.min(error)]

plot(size, error, type = "b",
     xlab = "# of terminal nodes",
     ylab = "CV errors")
abline(v = best.size, col = 2)

plot(alpha, error, type = "b",
     xlab = expression(alpha),
     ylab = "CV errors")
abline(v = best.alpha, col = 2)
```





```
> prune.train.obj <- prune.tree(train.obj, best = best.size)
> plot(prune.train.obj)
> text(prune.train.obj, pretty = 0)
>
> hat.prune.test.Salary <- predict(prune.train.obj, newdata = test.hitters)
> plot(test.hitters$Salary, hat.prune.test.Salary)
>
> # training error
> mean((test.hitters$Salary - hat.test.Salary)^2)
[1] 0.2985096
>
> # test errpr
> mean((test.hitters$Salary - hat.prune.test.Salary)^2)
[1] 0.2718084
```

예제: 05_tree_reg_hitters.R

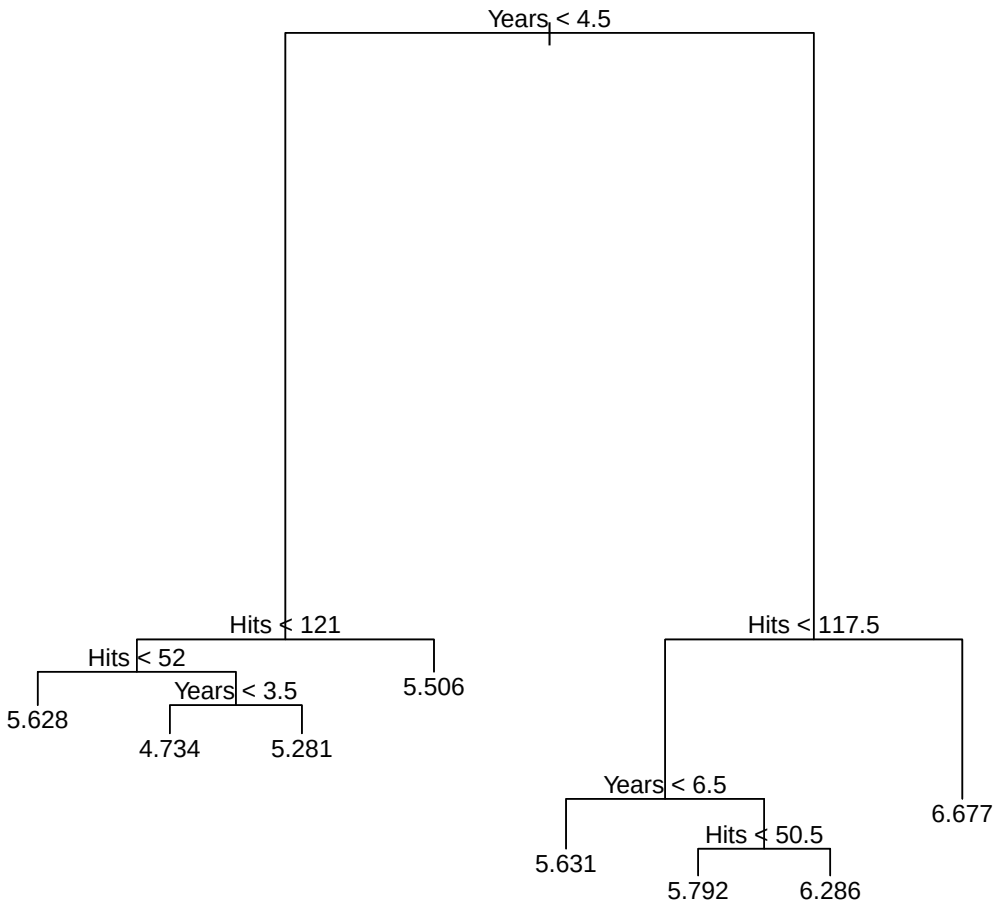


Figure: Final Model after pruning

분류 나무 모형

- 기본적으로 회귀 나무 모형과 아주 흡사.
- 다른점:
  - 예측: 파티션내에서 가장 많이 관측된 training sample 범주로 분류.
  - 적합: 잔차 제곱합을 기준으로 쓸 수 없음.

- 잔차제곱합(RSS)을 기준으로 사용할 수 없음.
- $\hat{p}_{m,\ell}$: R_m 내의 ℓ번째 범주에 해당하는 training sample의 상대도수.
- 지니 계수 (Gini index)

$$G_m = \sum_{\ell=1}^{L} \hat{p}_{m,\ell}(1 - \hat{p}_{m,\ell})$$

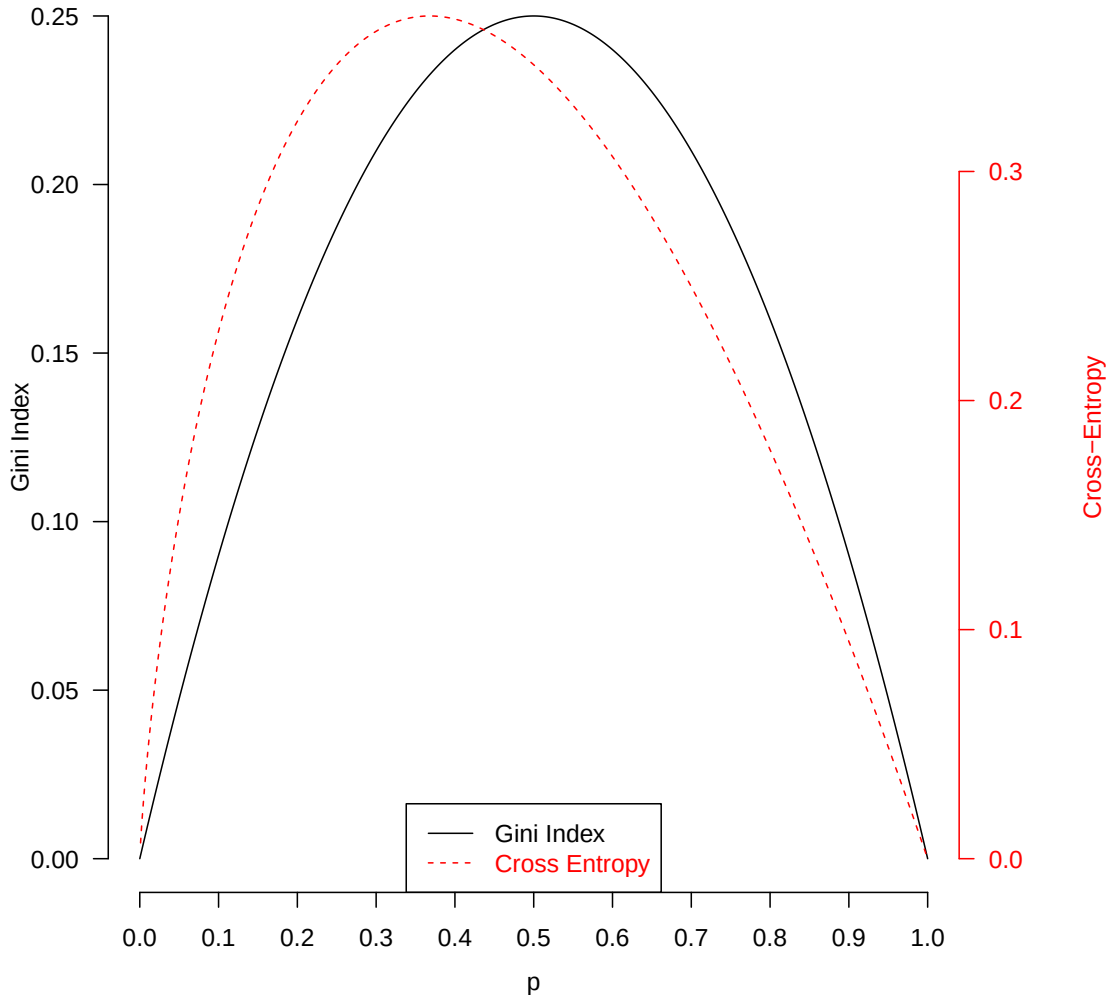
  - R_m 내에서 특정한 범주에 해당하는 개체수가 많을수록 작은 값을 가짐

  $\Rightarrow$ measure of node purity.

- Cross Entropy

$$D_m = -\sum_{\ell=1}^{L} \hat{p}_{m,\ell} \log \hat{p}_{m,\ell}$$

분류 나무 모형: node purity measure



```
> obj <- tree(High ~. -Sales, Carseats)
>
> summary(obj)

Classification tree:
tree(formula = High ~ . - Sales, data = Carseats)
Variables actually used in tree construction:
[1] "ShelveLoc"   "Price"       "Income"      "CompPrice"   "Population"  "Adve
Number of terminal nodes:  27
Residual mean deviance:  0.4575 = 170.7 / 373
Misclassification error rate: 0.09 = 36 / 400
>
> plot(obj)
> text(obj, pretty = 0)
>
```

예제: 05_tree_class_carseat.R

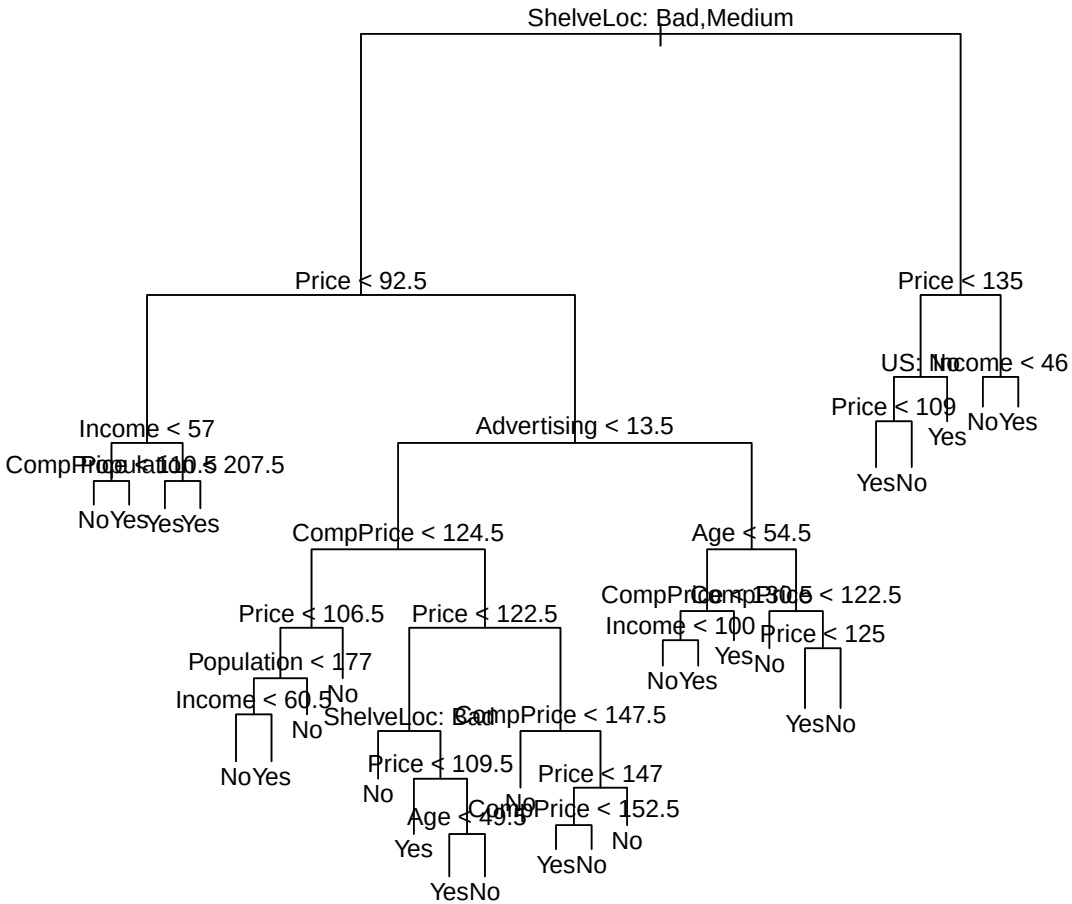
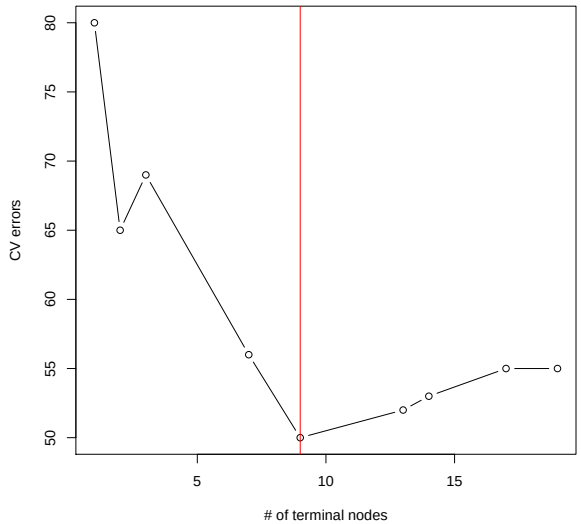


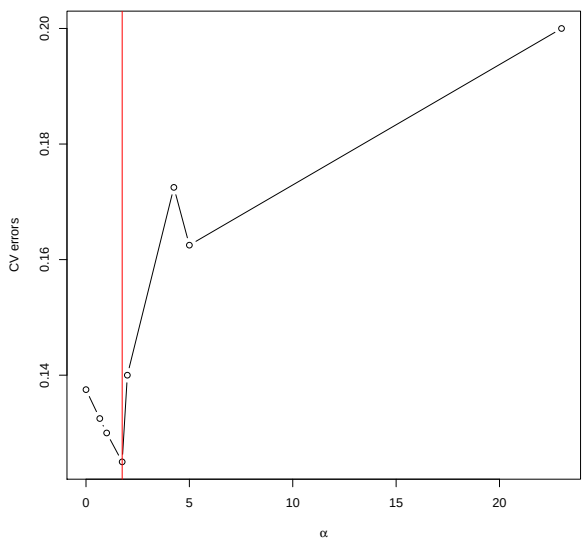
Figure: Full Classification Tree, T_0

```
> # train/test
> set.seed(2)
> n <- nrow(Carseats)
> id <- sample(n, 200)
>
> train.Carseats <- Carseats[id,]
> test.Carseats <- Carseats[-id,]
>
> train.obj <- tree(High ~. -Sales, train.Carseats)
> hat.test.High <- predict(train.obj, test.Carseats, type = "class")
> table(hat.test.High, test.Carseats$High)

hat.test.High No Yes
          No  86  27
          Yes 30  57
> mean(hat.test.High != test.Carseats$High)
[1] 0.285
```

예제: 05_tree_class_carseat.R

```
# Tree Pruning via Cross validation
set.seed(3)
cv.obj <- cv.tree(train.obj,
                  FUN = prune.misclass)

size  <- cv.obj$size
error <- cv.obj$dev
alpha <- cv.obj$k

best.size <- size[which.min(error)]
best.alpha <- alpha[which.min(error)]

plot(size, error, type = "b",
     xlab = "# of terminal nodes",
     ylab = "CV errors")
abline(v = best.size, col = 2)

plot(alpha, error, type = "b",
     xlab = expression(alpha),
     ylab = "CV errors")
abline(v = best.alpha, col = 2)
```





```
> train.prune.obj <- prune.misclass(obj, best = best.size)
> plot(train.prune.obj)
> text(train.prune.obj, pretty = 0)
>
> hat.test.prune.High <- predict(train.prune.obj, test.Carseats,
                                 type = "class")
>
> table(hat.test.prune.High, test.Carseats$High)

hat.test.prune.High  No Yes
                No  101  10
                Yes  15  74
>
> mean(hat.test.prune.High != test.Carseats$High)
[1] 0.125
```

예제: 05_tree_class_carseat.R

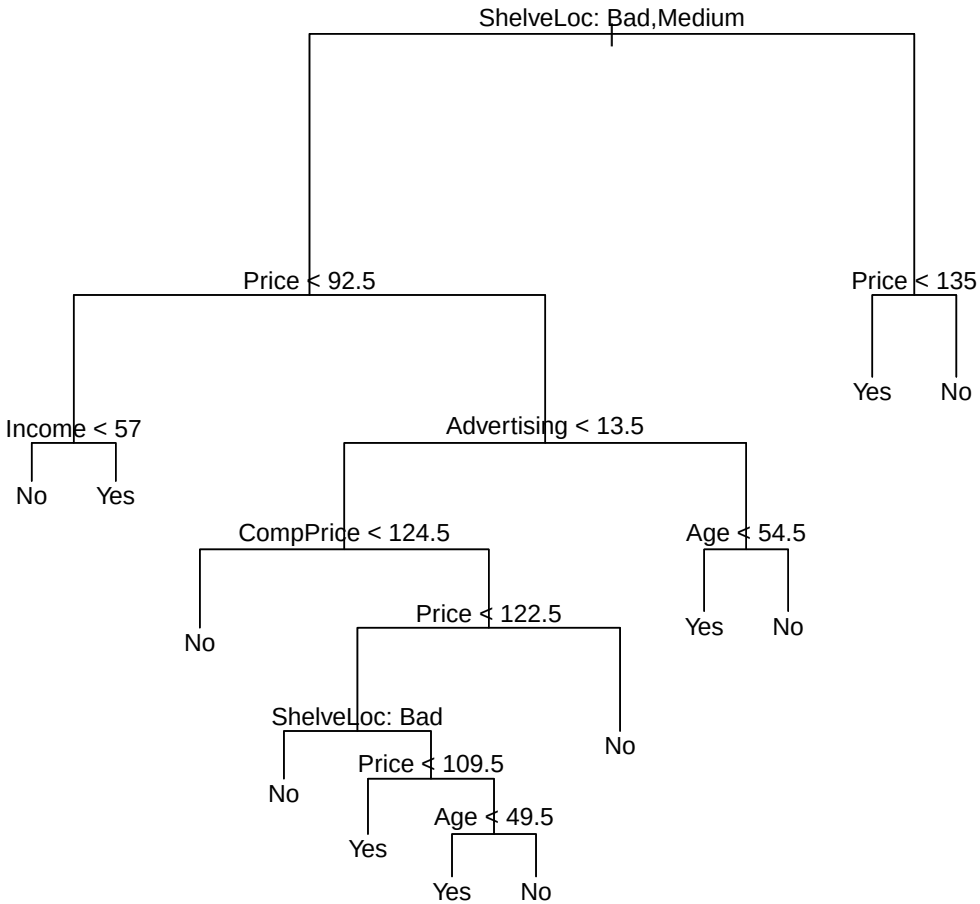


Figure: Final Model after pruning

나무 모형의 장단점

- 장점: 직관적이며, 모형의 해석이 쉬움.
- 단점: model variance가 상대적으로 높아 훈련자료에 민감. (weak learner)

목차

앙상블 방법: Motivation

- 나무 모형이 weak leaner인 이유: 훈련자료의 변이에 매우 민감 (why?)
- 모형의 분산이 지나치게 큼.
- Idea: 모형적합을 반복적으로 해서 결과를 합치면 분산이 줄어듬

$\Rightarrow$ $Var(X) = \sigma^2 \;\Rightarrow\; Var\left(\frac{1}{n}\sum_{i=1}^{n} X_i\right) = \frac{\sigma^2}{n}$.

Q. 표본은 하나뿐인데??

A. 재표본(resampling)을 이용하자! Bootstrapping!!

($*$) 붓스트랩 (Bootstrap) 표본: 훈련자료로 부터 n개의 개체를 임의 복원 추출.

$$(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5) \;\Rightarrow\; (\mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_2, \mathbf{x}_5, \mathbf{x}_1)$$

Bagging (Bootstrap Aggregating)

1. 주어진 훈련자료 $(y_i, \mathbf{x}_i)$로으로 부터 resampling을 통해 bootstrap 표본 $(y_i^{*,b}, \mathbf{x}_i^{*,b})$을 얻는다.
2. bootstrap 표본 $(y_i^{*,b}, \mathbf{x}_i^{*,b})$을 바탕으로 나무 모형, T_b^*을 적합한다. No pruning!
3. 위의 과정 1-2를 B번 $(b = 1, \cdots, B)$ 독립적으로 반복하여 B개의 나무 모형 $T_1^*, \cdots, T_B^*$을 적합한다.
4. 이제 새로운 개체 $\mathbf{x}_0$에 대한 반응 변수 y를 예측하는 방법은 다음과 같다.
   - 회귀: B개 서로다른 나무 모형으로 부터 나온 예측값들의 평균
   - 분류: B개의 서로다른 나무 모형으로 부터 나온 분류결과에 대한 majority vote.

- Bagging과 동일.
- 단, 나무 모형을 적합할 때, 각 binary split 단계에서 p개의 모든 변수를 이용하는 것이 아니라 random하게 선택된 몇 개 ($\sqrt{p}$)의 변수만을 이용해 split을 한다.
- 이런 측면에서, Bagging은 Random Forest의 special case라 할 수 있다.

예제: 05_tree_ensemble_boston.R

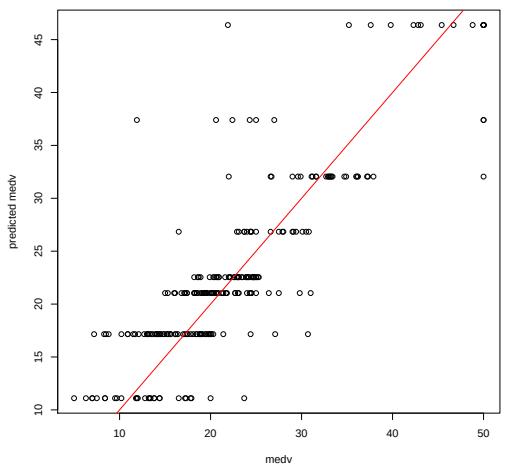
```
> # signle tree
> single.obj <- tree(medv ~ ., data = train.Boston)
>
> cv.obj <- cv.tree(single.obj)
> best <- cv.obj$size[which.min(cv.obj$dev)]
>
> prune.obj <- prune.tree(single.obj, best = best)
>
> hat.medv1 <- predict(prune.obj, newdata = test.Boston)
>
> plot(test.Boston$medv, hat.medv1, xlab = "medv", ylab = "predicted medv")
> abline(a = 0, b = 1, col = 2)
>
> mean((test.Boston$medv - hat.medv1)^2)
[1] 25.04559
```

```
> # Bagging
> bagging.obj <- randomForest(medv ~ ., data = train.Boston, mtry = p)
> bagging.obj

Call:
 randomForest(formula = medv ~ ., data = train.Boston, mtry = p)
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 13

          Mean of squared residuals: 10.90139
                    % Var explained: 86.8
>
> hat.medv2 <- predict(bagging.obj, newdata = test.Boston)
> plot(test.Boston$medv, hat.medv2, xlab = "medv", ylab = "predicted medv")
> abline(a = 0, b = 1, col = 2)
>
> mean((test.Boston$medv - hat.medv2)^2)
[1] 13.1939
```

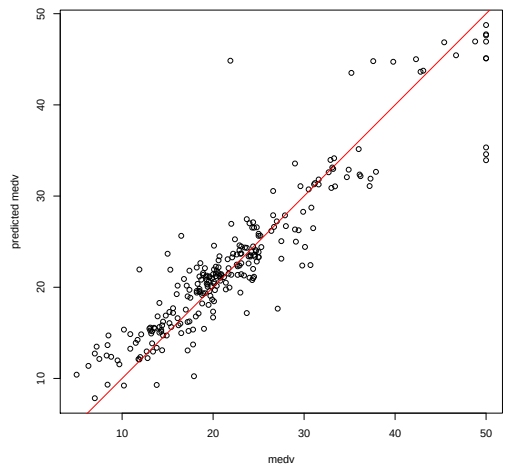
예제: 05_tree_ensemble_boston.R

```
> # RandomForest
> rforest.obj <- randomForest(medv ~ ., data = train.Boston,
                              mtry = floor(sqrt(p)))
> rforest.obj

Call:
 randomForest(formula = medv ~ ., data = train.Boston, mtry = floor(sqrt(p)))
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 3

          Mean of squared residuals: 14.72969
                    % Var explained: 82.16
>
> hat.medv3 <- predict(rforest.obj, newdata = test.Boston)
> plot(test.Boston$medv, hat.medv3, xlab = "medv", ylab = "predicted medv")
> abline(a = 0, b = 1, col = 2)
>
> mean((test.Boston$medv - hat.medv3)^2)
[1] 12.44349
```

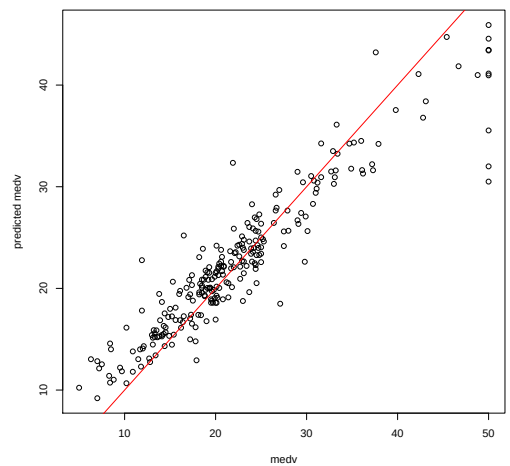
예제: 05_tree_ensemble_boston.R



(a) Tree



(b) Bagging



(c) Random Forest

목차

- 데이터: $(y_i, \mathbf{x}_i) = (y_i, x_{i1}, \cdots, x_{ip}), i = 1, \cdots, n.$
- 다중 선형 회귀 모형:

$$y_i = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip} + \epsilon_i, \quad i = 1, \cdots, n$$

오차항은 $\epsilon_1, \cdots, \epsilon_n \overset{iid}{\sim} N(0, \sigma^2)$.

- 최소제곱 (LS) 추정량

$$\hat{\boldsymbol{\beta}}_{LS} = \underset{\beta_0, \beta_1, \cdots, \beta_p}{\operatorname{argmin}} \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_{i1} - \cdots - \beta_p x_{ip})^2$$

- 훈련자료의 개체수 n이 고려하고 있는 독립변수의 갯수 p 보다 충분히 크다면 최소제곱 추정량 $\hat{\boldsymbol{\beta}}_{LS}$은 매우 좋은 추정량이다.

- p: 모형의 complexity - p가 클수록 모형이 복잡해짐.
- n: 정보의 양
- 만약, p에 비해 n이 충분히 크지 않다면, 모형의 예측력이 감소한다.
- $p > n$인 경우에는 최소제곱추정량이 유일하게 구해지지 않는다.
- p가 커질수록, 모형의 해석이 어렵다.

Three approahces

- 변수선택 (subset selection): p개의 변수중 중요한 것들만 선택

  (ex. 전진선택법, 후진제거법, 단계적선택법, 최적조합 선택법)

- 차원축소 (dimension reduction): p차원상에 정의된 독립변수 $\mathbf{x}$를 저차원 공간으로 투영시킨뒤 회귀모형을 적합.

  (ex. 주성분 회귀)

- 벌점화 모형 (penalization, regularization, shrinkage method): 최소제곱추정에 적절한 벌점항을 추가하므로써, 회귀계수 추정량을 0의 방향으로 밀어내는 방법. 편향을 발생시키지만, 분산을 줄여줌으로써 더나은 예측력을 보일 수 있음.

  (ex. Ridge regression, Lasso)

- Ridge regression 추정량:

$$\boldsymbol{\beta}_{Ridge} = \underset{\boldsymbol{\beta}}{\operatorname{argmin}} \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

- λ는 조율모수로 penalty의 효과를 조절
  - $\lambda = 0$: No penalty $\Rightarrow$ $\boldsymbol{\beta}_{Ridge} = \boldsymbol{\beta}_{LS}$
  - $\lambda = \infty$: $\sum_{j=1}^{p} \beta_j^2 = 0 \Leftrightarrow \boldsymbol{\beta}_{Ridge} = \mathbf{0}$
- λ가 커질수록 모형의 분산은 작아지고 편향은 커진다. (편향-분산 트레이드오프)
- 조율모수 λ는 교차타당성 검증을 통해 정할 수 있다.

편향-분산 트레이드 오프

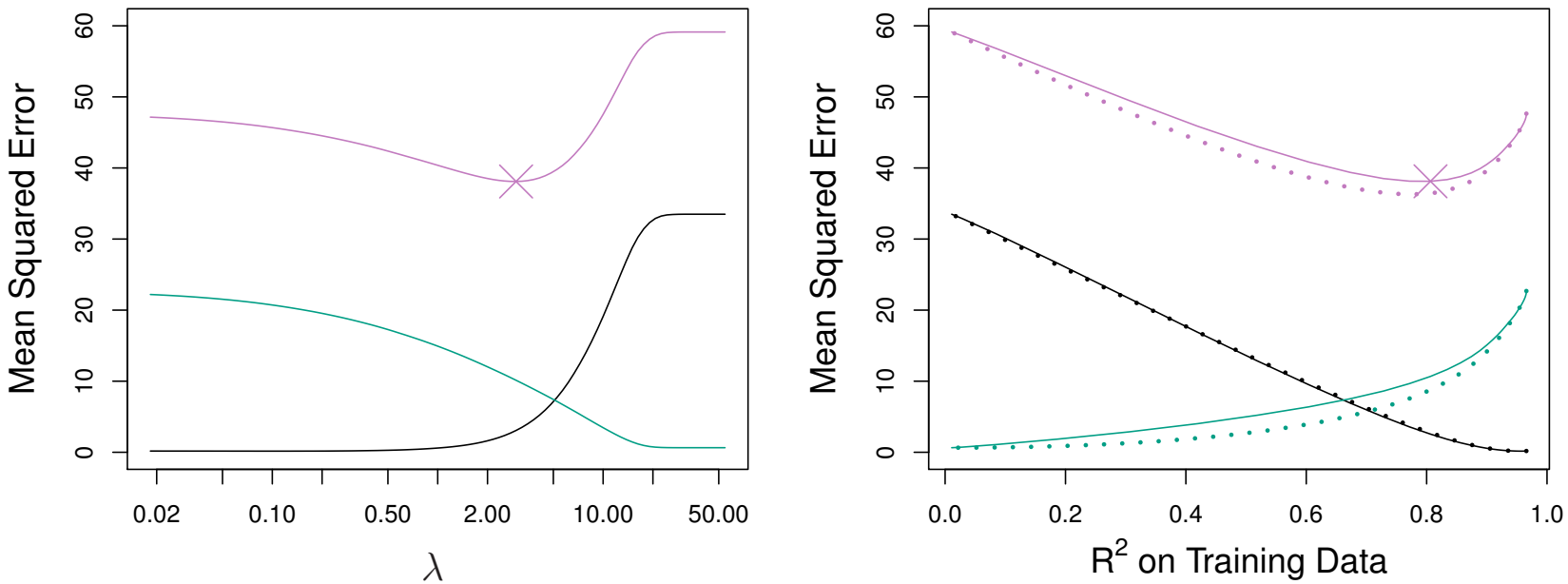


Figure: Bias-variance tradeoff

```
> # ridge regression
> grid <- 10^seq(5, -2, length = 100)
> ridge.obj <- glmnet(x, y, alpha = 0, lambda = grid)
>
> ridge.obj$lambda[25]
[1] 2009.233
> coef(ridge.obj)[,25]
  (Intercept)         AtBat          Hits         HmRun          Runs
171.749537815   0.099881529   0.429043890   1.269655963   0.678613196
          RBI         Walks         Years        CAtBat         CHits
  0.666517894   0.890125758   2.669738831   0.008492743   0.033191424
       CHmRun         CRuns          CRBI        CWalks       LeagueN
  0.245335576   0.066538307   0.068890197   0.064573864   4.845653564
    DivisionW       PutOuts       Assists        Errors    NewLeagueN
-27.287378943   0.064061995   0.008778405  -0.211348049   4.087192329
>
> Beta <- coef(ridge.obj)
> plot(0, 0, type = "n",
+      xlim = quantile(log(grid), c(0,1)),
+      ylim = quantile(as.numeric(Beta[-1,]), c(0,1)),
+      xlab = "lambda", ylab = "coefficient")
> for (j in 1:p) {
+   lines(log(grid), Beta[j+1,])
+   points(min(log(grid)), beta.ols[j+1], pch = 4, cex = 2, col = 2)
+ }
```

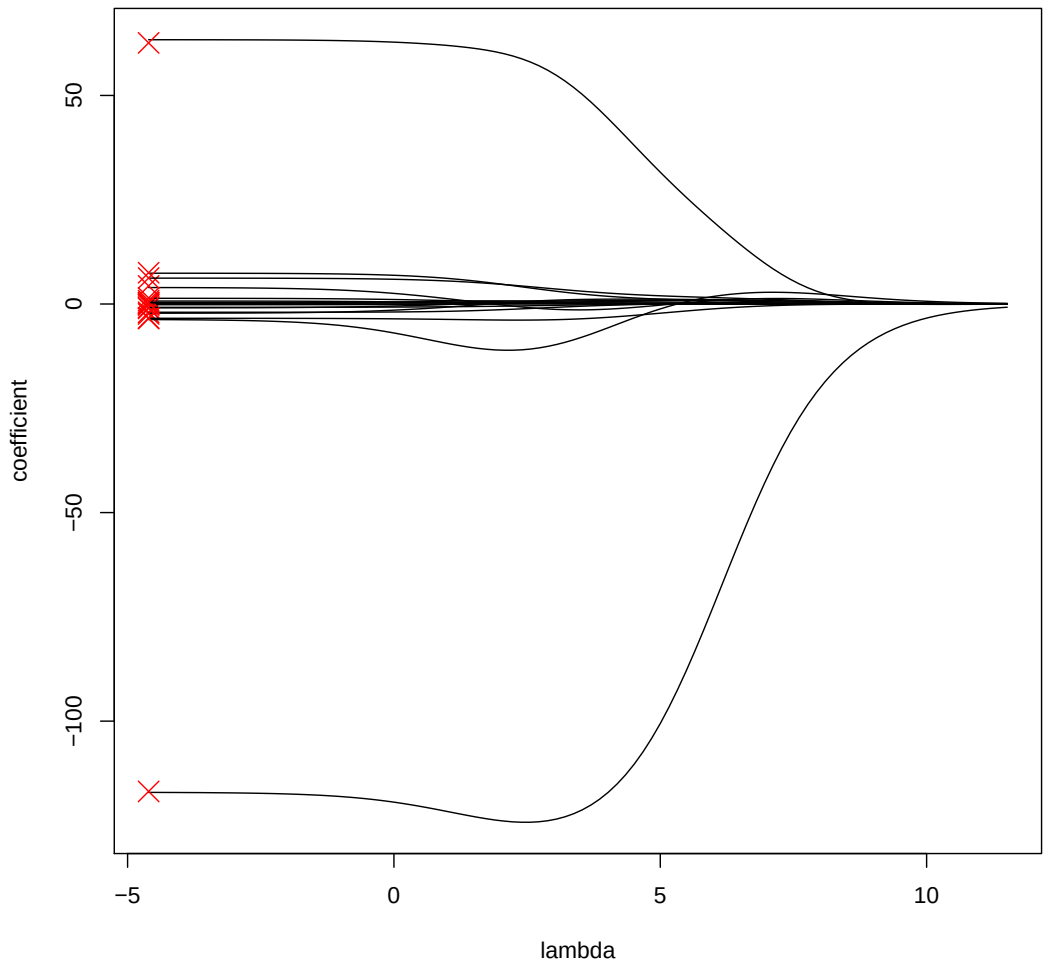


Figure: Ridge regression estimates as a function of λ.

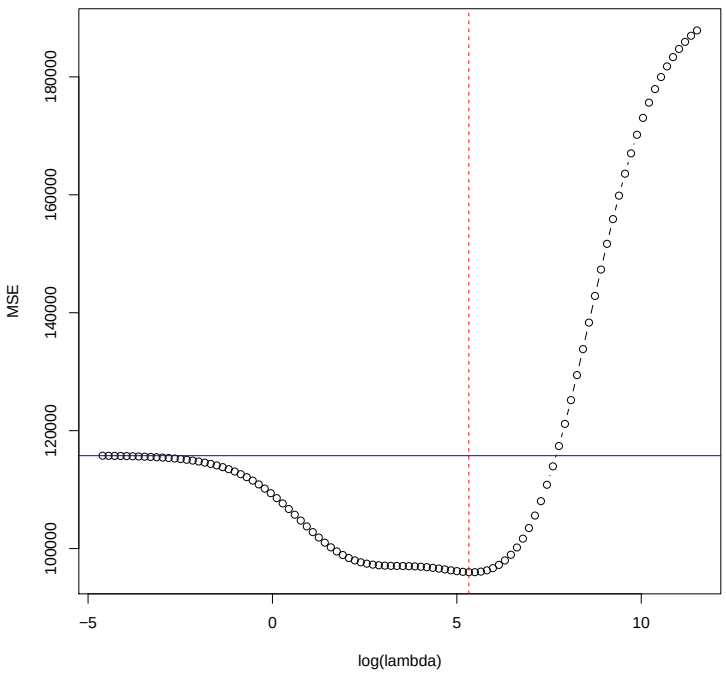
예제: 06_preg_ridge_hitter.R

```
> train.ridge.obj <- glmnet(x.train, y.train, alpha = 0, lambda = grid)
> hat.y.test <- predict(train.ridge.obj, s = 4, newx = x.test)
> mean((hat.y.test - y.test)^2)
[1] 101182.2
>
> test.mse <- NULL
> for (i in 1:length(grid)){
+   hat.y.test <- predict(train.ridge.obj, s = grid[i], newx = x.test)
+   test.mse[i] <- mean((hat.y.test - y.test)^2)
+ }
>
> plot(log(grid), test.mse, type = "b",
+      xlab = "log(lambda)", ylab = "MSE")
>
> opt.lambda.test <- grid[which.min(test.mse)]
> opt.lambda.test
[1] 205.6512
>
> abline(v = log(opt.lambda.test), col = 2, lty = 2)
> hat.y.test.ols <- predict(train.ridge.obj, s = 0, newx = x.test)
> test.mse.ols <- mean((hat.y.test.ols - y.test)^2)
> abline(h = test.mse.ols, col = 4)
```

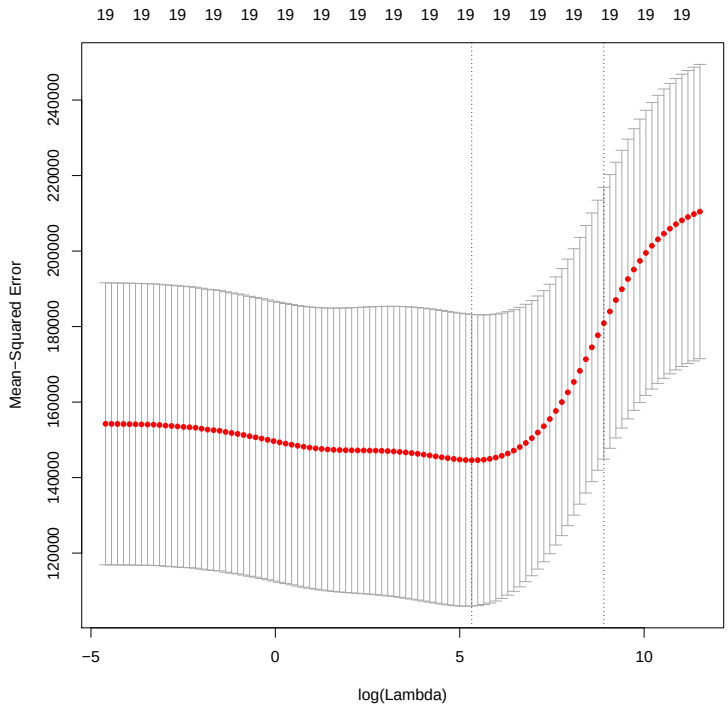
예제: 06_preg_ridge_hitter.R

```
> # cross validation
> set.seed(1)
> cv.obj <- cv.glmnet(x.train, y.train, alpha = 0, lambda = grid)
> plot(cv.obj)
> opt.lambda.cv <- cv.obj$lambda.min
> opt.lambda.cv
[1] 205.6512

> # final model
> train.ridge.obj <- glmnet(x.train, y.train, alpha = 0, lambda = grid)
> hat.y.test.opt <- predict(train.ridge.obj, s = opt.lambda.cv, newx = x.test)
> mean((hat.y.test.opt - y.test)^2)
[1] 95999.56
```

예제: 06_preg_ridge_hitter.R



(a) test error rate



(b) CV error rate

LASSO: Least Absolute Shrinkage and Selection Operator

- LASSO 추정량:

$$\boldsymbol{\beta}_{LASSO} = \underset{\boldsymbol{\beta}}{\operatorname{argmin}} \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

- Ridge: $\|\boldsymbol{\beta}\|_2^2 = \sum_{j=1}^{p} |\beta_j|^2$ (L_2-penalty)
- LASSO: $\|\boldsymbol{\beta}\|_1 = \sum_{j=1}^{p} |\beta_j|$ (L_1-penalty)
- LASSO는 변수 선택의 기능을 한다. (i.e., $\hat{\boldsymbol{\beta}}_{LASSO}$ is sparse.)

- Ridge:

$$\boldsymbol{\beta}_{Ridge} = \underset{\boldsymbol{\beta}}{\operatorname{argmin}} \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 \quad \text{subject to } \sum_{j=1}^{p} \beta_j^2 < s$$

- LASSO:

$$\boldsymbol{\beta}_{LASSO} = \underset{\boldsymbol{\beta}}{\operatorname{argmin}} \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 \quad \text{subject to } \sum_{j=1}^{p} |\beta_j| < s$$

비교: Ridge vs LASSO

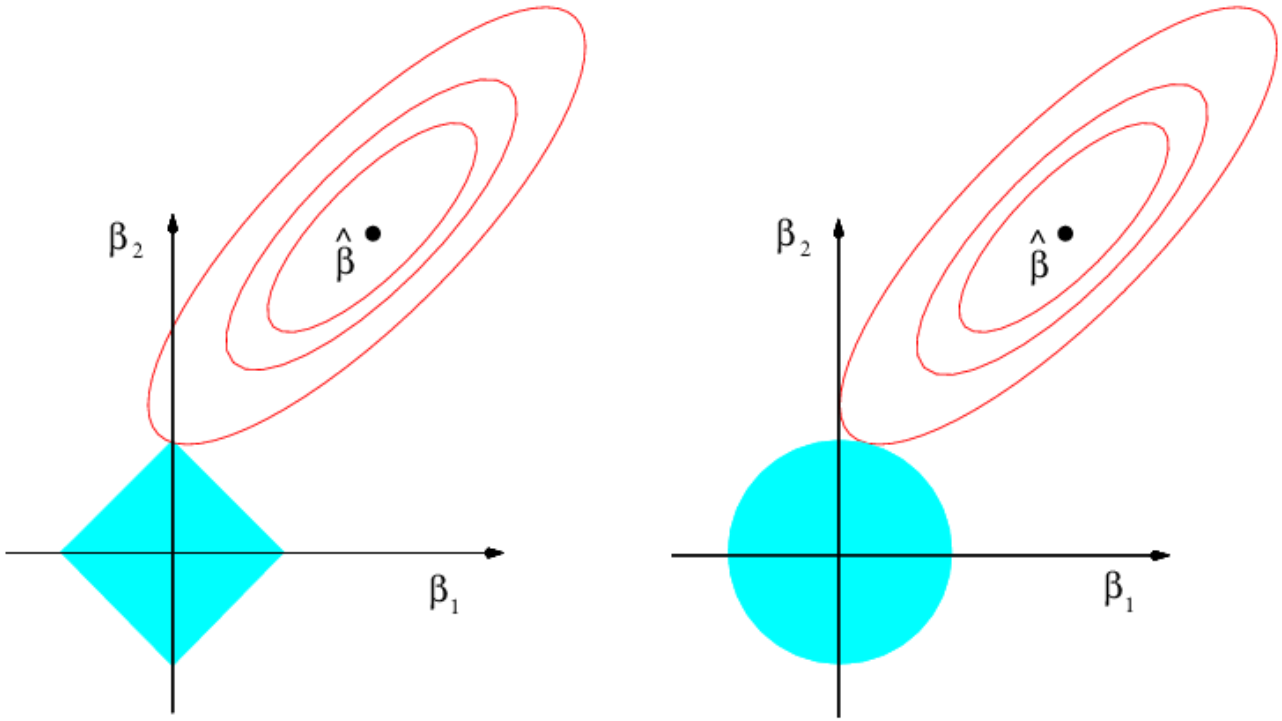


Figure: Geometrical comparison: 변수선택의 원리

```
> # LASSO
> grid <- 10^seq(5, -2, length = 100)
> lasso.obj <- glmnet(x, y, alpha = 1, lambda = grid)
>
> lasso.obj$lambda[25]
[1] 2009.233
> coef(lasso.obj)[,25]
(Intercept)       AtBat        Hits       HmRun        Runs         RBI
   535.9259      0.0000      0.0000      0.0000      0.0000      0.0000
      Walks       Years      CAtBat       CHits      CHmRun       CRuns
     0.0000      0.0000      0.0000      0.0000      0.0000      0.0000
       CRBI      CWalks     LeagueN   DivisionW     PutOuts     Assists
     0.0000      0.0000      0.0000      0.0000      0.0000      0.0000
     Errors  NewLeagueN
     0.0000      0.0000
>
> Beta <- coef(lasso.obj)
> plot(0, 0, type = "n",
+      xlim = quantile(log(grid), c(0,1)),
+      ylim = quantile(as.numeric(Beta[-1,]), c(0,1)),
+      xlab = "lambda", ylab = "coefficient")
> for (j in 1:p) {
+   lines(log(grid), Beta[j+1,])
+   points(min(log(grid)), beta.ols[j+1], pch = 4, cex = 2, col = 2)
+ }
```

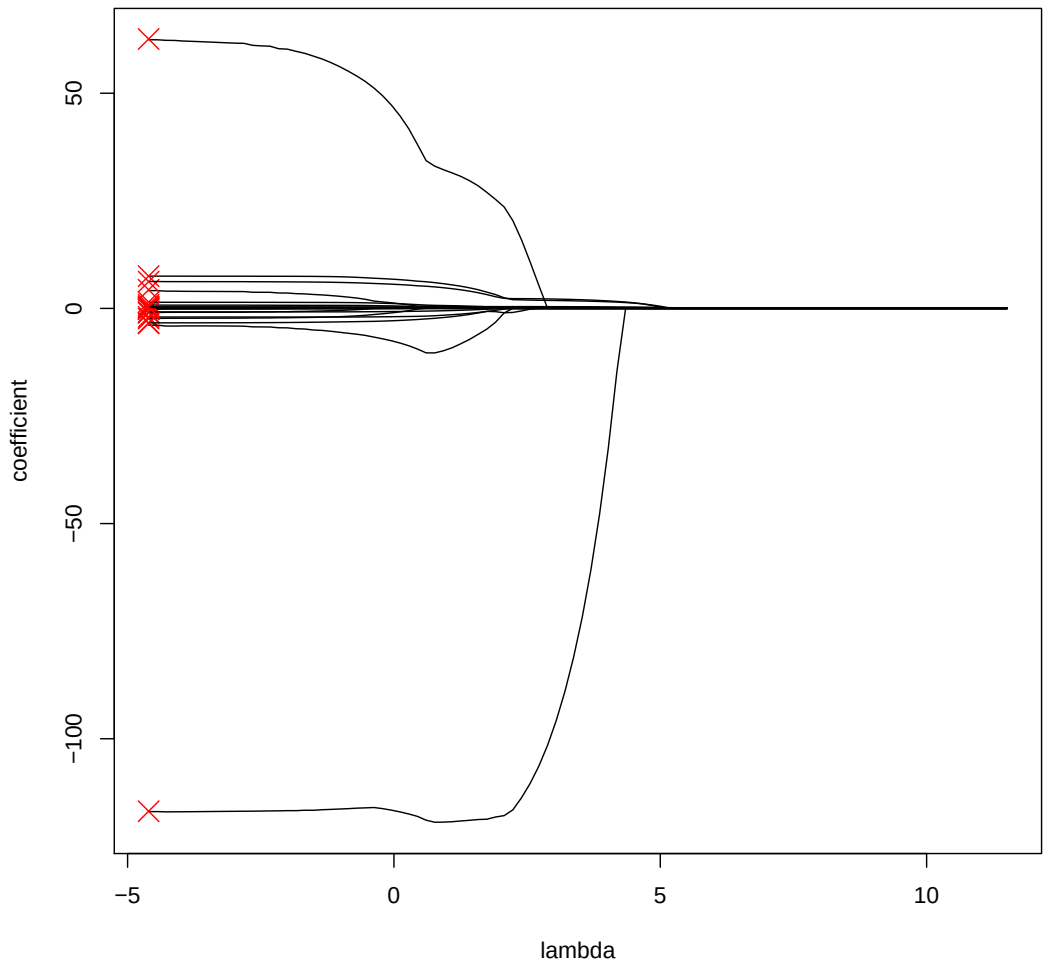
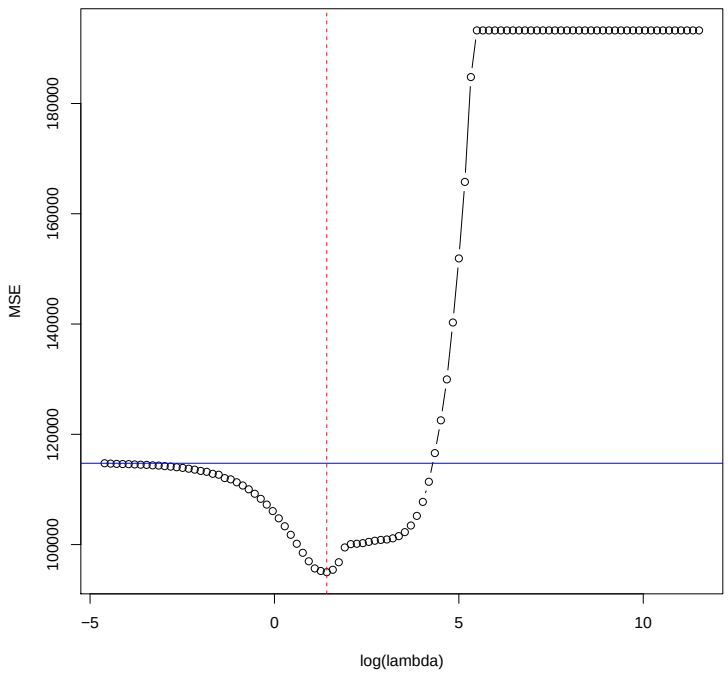


Figure: Ridge regression estimates as a function of λ.

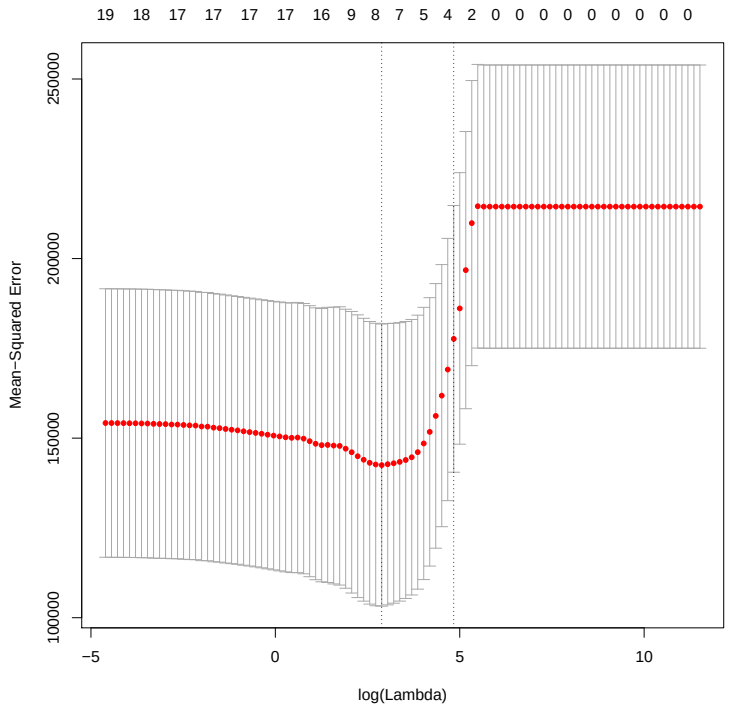
```
> train.lasso.obj <- glmnet(x.train, y.train, alpha = 1, lambda = grid)
> hat.y.test <- predict(train.lasso.obj, s = 0, newx = x.test)
> mean((hat.y.test - y.test)^2)
[1] 114752.2
>
> test.mse <- NULL
> for (i in 1:length(grid)){
+   hat.y.test <- predict(train.lasso.obj, s = grid[i], newx = x.test)
+   test.mse[i] <- mean((hat.y.test - y.test)^2)
+ }
>
> plot(log(grid), test.mse, type = "b",
+      xlab = "log(lambda)", ylab = "MSE")
>
> opt.lambda.test <- grid[which.min(test.mse)]
> opt.lambda.test
[1] 4.132012
>
> abline(v = log(opt.lambda.test), col = 2, lty = 2)
```

```
> # cross validation
> set.seed(1)
> cv.obj <- cv.glmnet(x.train, y.train, alpha = 1, lambda = grid)
> plot(cv.obj)
> opt.lambda.cv <- cv.obj$lambda.min
> opt.lambda.cv
[1] 17.8865
>
> # final model
> train.lasso.obj <- glmnet(x.train, y.train, alpha = 1, lambda = grid)
> hat.y.test.opt <- predict(train.lasso.obj, s = opt.lambda.cv, newx = x.test)
> mean((hat.y.test.opt - y.test)^2)
[1] 100850.1
```

예제: 06_preg_lasso_hitter.R



(a) test error rate



(b) CV error rate

예제: 비교 (Ridge vs OLS vs LASSO)

Variable	Ridge	OLS	Lasso
AtBat	-0.031	-1.980	.
Hits	0.514	7.501	.
HmRun	2.683	4.331	1.506
Runs	0.654	-2.376	.
RBI	0.833	-1.045	1.044
Walks	3.210	6.231	4.864
Years	-3.676	-3.489	.
CAtBat	0.010	-0.171	.
CHits	0.060	0.134	.
CHmRun	0.137	-0.173	.
CRuns	0.157	1.454	0.461
CRBI	0.063	0.808	.
CWalks	0.095	-0.812	.
LeagueN	38.598	62.599	28.208
DivisionW	-113.529	-116.849	-127.245
PutOuts	0.257	0.282	0.316
Assists	0.059	0.371	.
Errors	-0.477	-3.361	.
NewLeagueN	22.065	-24.762	.
Test MSE	95999.56	115756.3	100850.1

Table: Ridge vs OLS vs Lasso

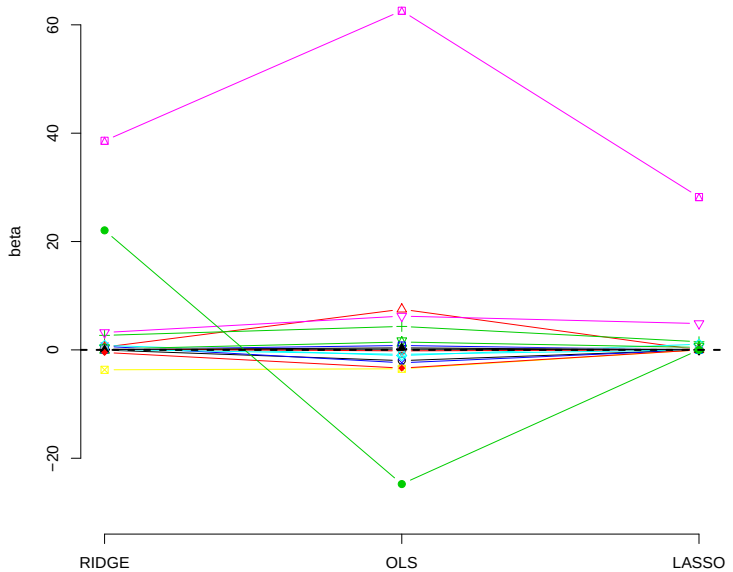


Figure: Ridge vs OLS vs Lasso

- Elastic net:

$$\sum_{i=1}^{n}\left(y_i-\beta_0-\sum_{j=1}^{p}\beta_j x_{ij}\right)^2+\lambda\left[\alpha\sum_{j=1}^{p}|\beta_j|+(1-\alpha)\sum_{j=1}^{p}\|\beta_j\|_2^2\right]$$

  - $\alpha=0 \Rightarrow$ Ridge
  - $\alpha=1 \Rightarrow$ LASSO
  - $0<\alpha<1 \Rightarrow$ Mixture

- Support vector machine:

$$\sum_{i=1}^{n}\left[1-y_i(\beta_0+\sum_{j=1}^{p}\beta_j x_{ij})\right]_+ + \frac{\lambda}{2}\|\boldsymbol{\beta}\|_2^2$$

  - $[u]_+ = u$ if $u \geq 0$ and 0 otherwise.

- Pruning tree

$$\sum_{m=1}^{|T|}\sum_{\mathbf{x}_i\in R_m}(y_i-\bar{y}_{R_m})^2+\alpha|T|$$