2019 학년도 1학기
DATA MINING
HW7

과목명	데이터마이닝
담당교수명	송종우 교수님
제출일	2019.05.08
학번	182STG27
이름	임지연

Ⅰ. Description

이제까지 데이터에 linear 모형을 주로 공부했는데, 8장에서는 tree 기반의 몇몇 방법론에 대해서 소개하고 있다. 넓은 관점에서 Tree-Based Model에 대해 반응변수가 연속형일 때 사용할 수 있는 Regression Tree, 반응변수가 범주형일 때 사용할 수 있는 Classification Tree 방법이 있다. 더하여 예측력이 눈에띄게 좋아질 수 있는 방법인 Bagging, Random Forest, Boosting 방법론에 대해서 공부할 것이다. 각각의 모형에서 사용되는 방법론에 대해서 알아본 후 어떤 단계를 걸쳐 실행되고 장단점에 대해서 알아본 후 Lab에 설명되어 있는 코드를 실행해 보고 결과를 이해한 후 연습문제를 풀어볼 것이다.

Ⅱ. Implementation

Question 1.

Lab : Decision Tree 의 코드를 실행해보고 감상문을 써라

1. Classification Tree - Carseats dataset 적합

ISLR 패키지에서 제공되는 Carseats 데이터를 사용하여 반응변수로 범주형 변수인 'Sales'에 대해 Classification Tree 를 적합해 보았다. Tree 패키지에서 제공되는 tree()함수를 기존에 lm 을 사용했던 문법과 같게 적용할 수 있다. 모델 fitting 후, summary한 결과 터미널노드 수 , training set 에 대한 deviance, misclassification rate 를 알 수 있다. 그 후 plot 함수를 통해 어떤 변수를 가지고 어떤 split point 로 노드를 분할하는 지 알 수 있다. 앞서 살펴보았던 Cross-Validation 방법을 사용할 수도 있는데 cv.tree() 함수를 사용하면 된다. 옵션으로 FUN = prune.misclass 라고 설정해 준다면 default 값인 deviance가 아닌 misclassification rate 기준으로 cv와 pruning 과정이 수행될 수 있다. Cv.tree() 결과 터미널 노드 수, deviance, alpha(cost-complex parameter) 값을 알 수 있으며 terminal node size와 deviance 값에 대한 plot을 그려보면 어떤 terminal node값에서 가장 작은 deviance 값을 가지는지 알 수 있고 prune.misclass() 함수를 사용하여 옵션으로 best = 최적의 terminal node값을 지정해 주면 모형의 가지치기가 수행된다. 즉 모든 변수를 사용하지 않고 모든 변수값보다 더 작은 terminal node값을 사용해줌으로써 test MSE 값을 pruning 하기 전과 후를 함께 비교해 볼 수 있다. 많은 상황에서 해석력과 분류 정확도가 더 향상된 트리를 제공한다. Predict()함수를 사용해 예측값을 구한 후 misclassification rate를 가지고 분류 모형의 성능을 비교할 수 있다.

2. Regression Tree – Boston dataset 적합

MASS 패키지에서 제공되는 Boston 데이터를 사용하여 반응변수로 연속형 변수인 'medv'에 대해 Regression Tree를 적합해 보았다. Tree() 함수를 사용하여 같은 방법으로 적합이 가능하며, 마찬가지로 Cross-validation 방법을 사용하기 위해 cv.tree를 이용해 적합 후, plot()함수를 이용해 가장 적절한 terminal node 수를 찾아 pruning 할 수 있다. 최종모형을 가지고 predict()함수를 사용해 예측값을 찾은 수 test set 에서의 값과 비교해 test MSE 값을 계산할 수 있다.

3. Bagging, Random Forest – Boston dataset 적합

MASS 패키지에서 제공되는 Boston 데이터를 사용하여 반응변수로 연속형 변수인 'medv'에 대해 Bagging, Random Forest를 적합해 보았다. randomForest 패키지에서 제공되는 randomForest() 함수를 사용할 수 있는데 Bagging 은 Random Forest 에서 모든 변수를 사용하는 경우에 해당하기 때문에 이 함수를 사용가능하다. Mtry 옵션을 이용해 random forest에서 tree 수를 지정해 줄 수 있다. 그 후 predict() 함수를 이용해 testMSE 값을 구할 수 있다. 여기서 importance() 함수를 사용할 수 있는데 이 함수는 각 변수마다 IncMSE, IncNodePurity 값을 제공해 준다. varImpPlot() 함수를 사용하여 그래프로 시각화하면 어떤 변수가 가장 중요한 지 알 수 있다. 즉 Bagging 방법은 mtry 수를 데이터에서 변수만큼 설정해주고 RandomForest 에서는 mtry 수는 일반적으로 Classification Tree의 경우 $\sqrt{p}$개, Regression Tree의 경우 p/3 개만큼 설정해 준다.

4. Boosting – Boston dataset 적합

마찬가지로 Boston 데이터에 대해서 gbm() 함수를 사용하여 Boosting 할 수 있다. 옵션으로는 distribution 을 지정할 수 있는데 회귀문제의 경우 "gaussian" 으로 지정해주고 Classification 문제라면 "Bernoulli"로 지정해 줄 수 있다. N.trees 는 트리 수를 의미하며 interaction.depth 옵션으로 트리의 깊이를 지정해 줄 수 있다. 모형 적합 후, Partial Dependence Plot 을 생성할 수 있는데 plot(적합된모형, i = "관심있는 변수") 로 지정하면 관심있는 변수에 대해 y 값의 변화를 살펴볼 수 있다. 그 후 같은 방법으로 predict() 함수를 이용해 예측값을 살펴본 뒤 testMSE 값을 계산할 수 있다. Shirinkage parameter 인 lambda 값을 변화하면서 Boosting 해보며 성능을 비교할 수도 있다.

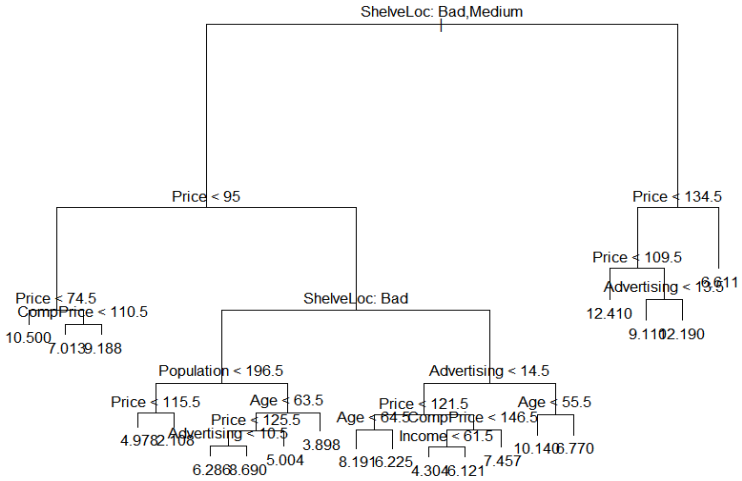
Question 2.

8.4 Excercises - Example 8, 9, 10 풀어라

[Example 8]

In the lab, a classification tree was applied to the Carseats data set after converting Sales into a qualitative response variable. Now we will seek to predict Sales using regression trees and related approaches, treating the response as a quantitative variable.

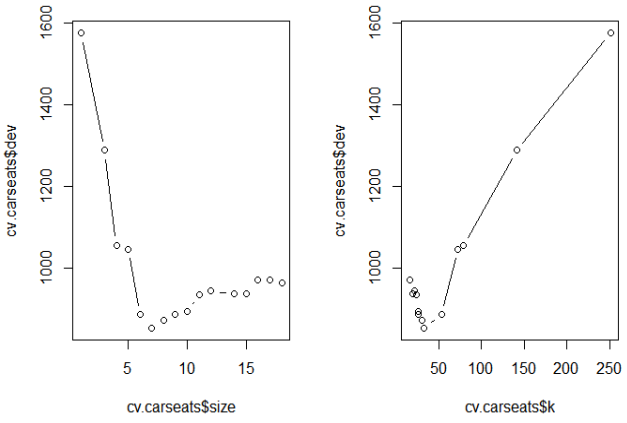
(a) Split the data set into a training set and a test set. (b) Fit a regression tree to the training set. Plot the tree, and interpret the results. What test error rate do you obtain



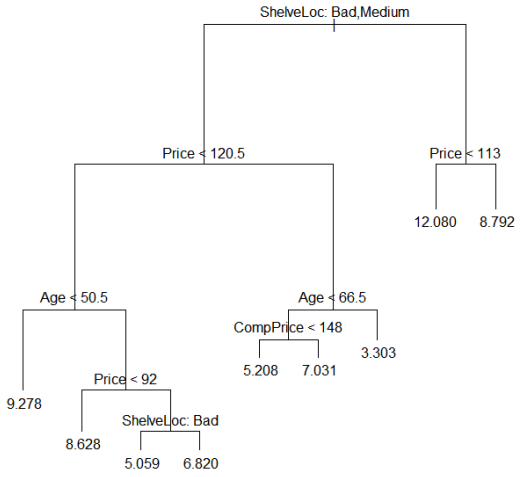
▶ 판매량에 가장 중요한 영향을 미치는 변수는 'ShelveLoc'이다. 즉, 각 지역에서 자동차 좌석에 대한 선반 위치가 판매량에 가장 중요한 영향을 미치는 것이라고 할 수 있다.

▶ 또한 Validation Method에 대한 test MSE 를 계산해 본 결과 4.12로 나타났다.

(c) Use cross-validation in order to determine the optimal level of tree complexity. Does pruning the tree improve the test error rate?



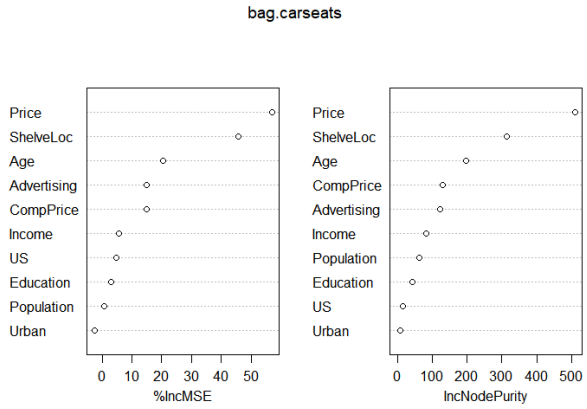
▶Terminal node수가 9일 때, deviance 값이 886.14로 가장 작은 것을 알 수 있다. 따라서, pruning 할 때 terminal node수를 9개로 두고 모델을 적합한다.



▶Terminal mode 수가 9일 때 모델을 적합한 결과 위와 같은 구조로 나타났고, test MSE값은 4.99로 나타났다.

(d) Use the bagging approach in order to analyze this data. What test error rate do you obtain? Use the importance() function to determine which variables are most important.

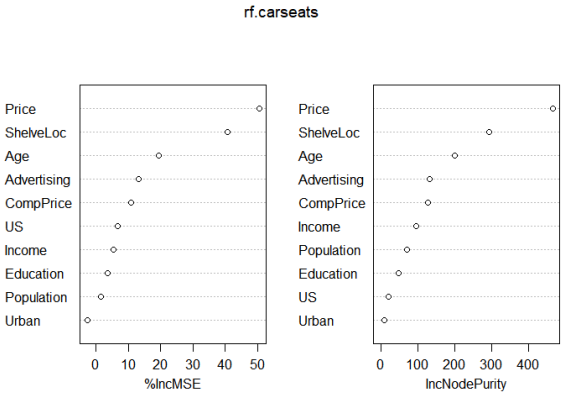
	IncMSE	IncNodePurity
CompPrice	14.87	130.34
Income	5.67	82.02
Advertising	15.05	122.24
Population	0.69	63.38
Price	57.07	509.78
ShelveLoc	45.63	313.09
Age	20.45	195.92
Education	3.07	42.92
Urban	-2.63	8.21
US	4.73	16.09



▶ Bagging 기법을 사용하여 데이터를 적합해 봤을 때, test MSE 값은 2.57로 나타났다. 또한 변수 중요도에 대한 IncMSE , IncNodePurity 값에 대한 표와 그에 대한 그래프는 위와 같다. 먼저 IncMSE 에 대한 의미는, 해당 변수가 모델에서 제외될 때 bagging 되지 않은 sample 에 대해서 predict accuracy에 대한 평균 감소량을 의미하며, IncModePurity 값은 해당 변수에 대해서 분할로 인한 노드 impurity 의 총 감소량에 대한 평균값을 의미한다. 따라서 Price, ShelveLoc, Age 순으로 모형에서 중요한 변수라고 할 수 있다.

(e) Use random forests to analyze this data. What test error rate do you obtain? Use the importance() function to determine which variables are most important. Describe the effect of m, the number of variables considered at each split, on the error rate obtained.

	X.IncMSE	IncNodePurity
CompPrice	10.93	128.21
Income	5.39	95.4
Advertising	13.26	132.9
Population	1.71	70.86
Price	50.56	466.84
ShelveLoc	40.7	293.63
Age	19.51	200.6
Education	3.59	49.71
Urban	-2.69	11.19
US	6.88	21.31



▶ Random Forest 기법을 사용하여 데이터를 적합해 봤을 때, test MSE 값은 2.74로 나타났다. 또한 변수 중요도에 대한 IncMSE , IncNodePurity 값에 대한 표와 그에 대한 그래프는 위와 같다. 위와 같은 이유로 모델에서 Price, ShelveLoc, Age 순으로 모형에서 중요한 변수라고 할 수 있다.

[Example 9]

This problem involves the OJ data set which is part of the ISLR package.
(a) Create a training set containing a random sample of 800 observations, and a test set containing the remaining observations. (b) Fit a tree to the training data, with Purchase as the response and the other variables except for Buy as predictors. Use the summary() function to produce summary statistics about the tree, and describe the results obtained. What is the training error rate? How many terminal nodes does the tree have?

▼Output

```
Classification tree: tree(formula = Purchase ~ ., data = OJ[train_index, ])
Variables actually used in tree construction: "LoyalCH", "PriceDiff", "SpecialCH" ,"ListPriceDiff"
Number of terminal nodes:  8
Residual mean deviance:  0.7305 = 578.6 / 792
Misclassification error rate: 0.165 = 132 / 800
```

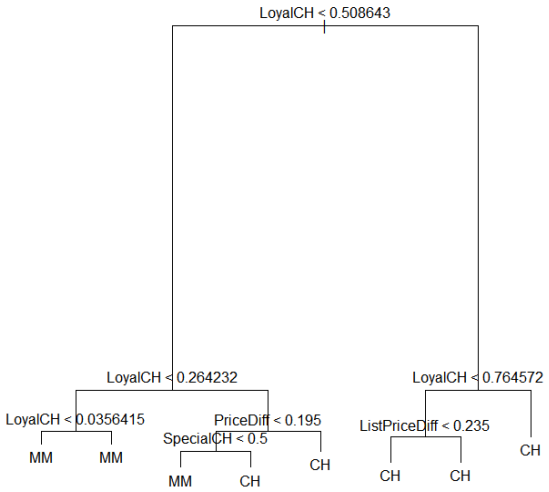
▶ OJ 데이터셋에 대해 Purchase를 반응변수 y로 하여 classification tree를 적합해 본 결과, terminal node 수는 8개, training set 에 대해서 오분류율은 0.16으로 나타났다.

(c) Type in the name of the tree object in order to get a detailed text output. Pick one of the terminal nodes, and interpret the information displayed.

```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##  1) root 800 1064.00 CH ( 0.61750 0.38250 )
##    2) LoyalCH < 0.508643 350  409.30 MM ( 0.27143 0.72857 )
##      4) LoyalCH < 0.264232 166  122.10 MM ( 0.12048 0.87952 )
##        8) LoyalCH < 0.0356415 57   10.07 MM ( 0.01754 0.98246 ) *
##        9) LoyalCH > 0.0356415 109  100.90 MM ( 0.17431 0.82569 ) *
##      5) LoyalCH > 0.264232 184  248.80 MM ( 0.40761 0.59239 )
##       10) PriceDiff < 0.195 83   91.66 MM ( 0.24096 0.75904 )
##         20) SpecialCH < 0.5 70   60.89 MM ( 0.15714 0.84286 ) *
##         21) SpecialCH > 0.5 13   16.05 CH ( 0.69231 0.30769 ) *
##       11) PriceDiff > 0.195 101  139.20 CH ( 0.54455 0.45545 ) *
##    3) LoyalCH > 0.508643 450  318.10 CH ( 0.88667 0.11333 )
##      6) LoyalCH < 0.764572 172  188.90 CH ( 0.76163 0.23837 )
##       12) ListPriceDiff < 0.235 70   95.61 CH ( 0.57143 0.42857 ) *
##       13) ListPriceDiff > 0.235 102   69.76 CH ( 0.89216 0.10784 ) *
##      7) LoyalCH > 0.764572 278   86.14 CH ( 0.96403 0.03597 ) *
```

▶ 위 결과로부터, " 9) LoyalCH > 0.0356415 109 100.90 MM (0.17431 0.82569) * " 의 결과를 해석해보겠다. 먼저 이 의미는 LoyalCH변수를 이용해서 split point 0.035를 이용하겠다는 의미이고, 이 해당 노드의 아래에 데이터는 109개가 있으며, 편차는 100.9임을 알 수 있다. 이 노드의 Predict y 값은 "MM"이며, 17%는 predict 값으로 CH를 가지고, 82%는 predict 값으로 MM 값을 가진다는 것을 알 수 있다. 또한, 마지막에 * 표시가 있는 노드에 대해서는 terminal node 임을 의미한다.

(d) Create a plot of the tree, and interpret the results.



▶ LoyalCH 변수가 가장 상단에서 노드를 구분하고 있기 때문에 가장 중요한 변수임을 알 수 있다. 그 아래 split 변수도 LoyalCH변수로 가장 중요한 변수임을 다시한번 보여준다. 왼쪽 아래에서 보면 LoyalCH <0.264일 때 terminal node값이 무조건 MM을 예측한다. 또한 LoyalCH> 0.76일 때는 CH값을 가진다는 것을 알 수 있다. 그 외의 범위에서는 PriceDiff, ListPriceDiff, SpecialCH값의 범위에 따라서 Class 값을 예측하는 것을 알 수 있다.

(e) Predict the response on the test data, and produce a confusion matrix comparing the test labels to the predicted test labels. What is the test error rate?

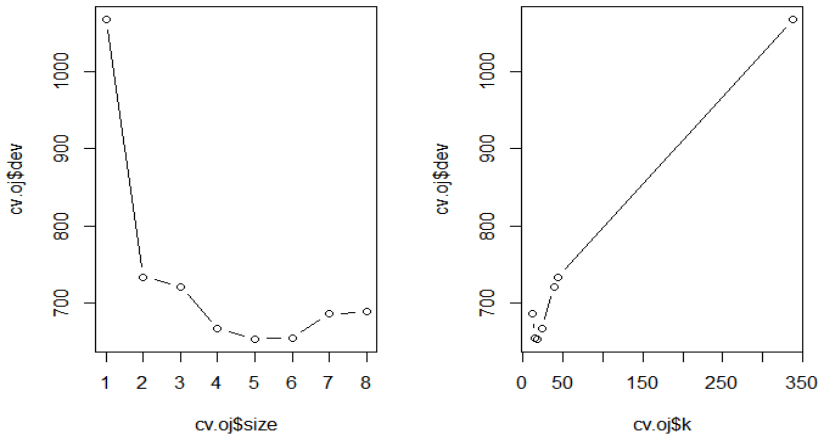
		True	
		CH	MM
Predict	CH	147	49
	MM	12	62

▶ Confusion Matrix 를 그려봤을 때, test error rate는 22%라는 것을 알 수 있다.

(f) Apply the cv.tree() function to the training set in order to determine the optimal tree size.

▶ cv.tree 함수를 사용하여 적합해 봤을 때 tree size가 5일 때 deviance 값이 653.7774로 가장 작은 것을 알 수 있다.

(g) Produce a plot with tree size on the x-axis and cross-validated classification error rate on the y-axis.



▶(f)에서의 결과로도 알 수 있듯이, tree size가 5일 때 가장 작은 test MSE 값을 가진다는 것을 d라 수 있다.

(h) Which tree size corresponds to the lowest cross-validated classification error rate?

▶ tree size가 5일 때 가장 작은 CV error 갖는다.

(i) Produce a pruned tree corresponding to the optimal tree size obtained using cross-validation. If cross-validation does not lead to selection of a pruned tree, then create a pruned tree with five terminal nodes.

(j) Compare the training error rates between the pruned and unpruned trees. Which is higher?

▶ pruned 되지 않은 tree 의 training error rate는 0.165, pruned 된 tree의 training error rate는 0.1713로, pruned 된 tree 값이 더 큰 것을 알 수 있다.

(k) Compare the test error rates between the pruned and unpruned trees. Which is higher?

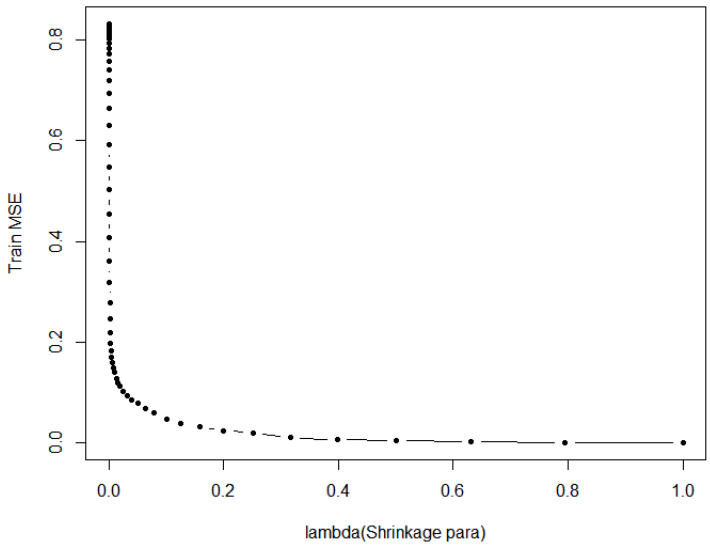
▶ pruned 되지 않은 tree 의 test error 는 0.2259, pruned 된 tree의 test error rate는 0.21로, pruned 되지 않은 tree 값이 더 큰 것을 알 수 있다.

[Example 10]

We now use boosting to predict Salary in the Hitters data set.

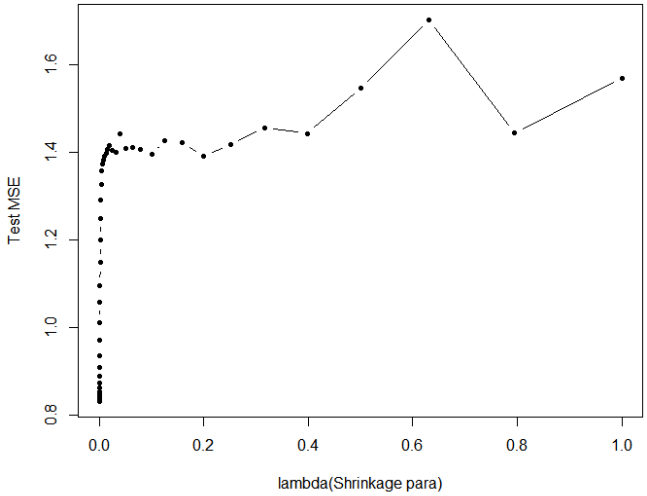
(a) Remove the observations for whom the salary information is unknown, and then log-transform the salaries. (b) Create a training set consisting of the first 200 observations, and a test set consisting of the remaining observations.

(c) Perform boosting on the training set with 1,000 trees for a range of values of the shrinkage parameter λ. Produce a plot with different shrinkage values on the x-axis and the corresponding training set MSE on the y-axis.



▶ Shirinkage parameter = 1일 때 train error 는 0.0007로 가장 작은 값을 가진다.

(d) Produce a plot with different shrinkage values on the x-axis and the corresponding test set MSE on the y-axis.

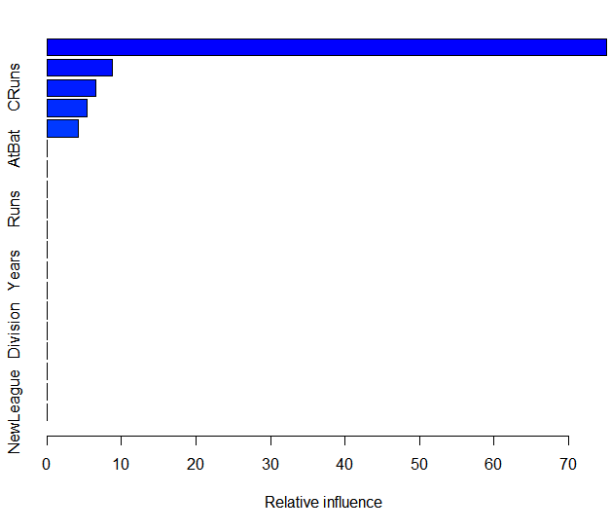


▶ Shirinkage = 1e-10 일 때 test error 는 0.83로 가장 작은 값을 가진다.

(e) Compare the test MSE of boosting to the test MSE that results from applying two of the regression approaches seen in Chapters 3 and 6.

Model	Linear model	Ridge Model	Lasso Model
Test MSE	0.49	0.45	0.47

(f) Which variables appear to be the most important predictors in the boosted model?



▶ 변수 중요도 plot 을 그려본 결과 CAtBat , Chits, Cruns, Cwalks, CRBI 순으로 중요도가 나타났다.

(g) Now apply bagging to the training set. What is the test set MSE for this approach?

▶ training set 에서 bagging 적용 후 구한 test MSE 는 0.229 이다.

Ⅲ. Discussion

이번 과제를 하면서 트리 기반의 몇 가지 방법을 알 수 있었다. 마찬가지로 test set 에서의 예측력을 높게 해주는 모형을 찾기 위한 방법의 특징과 장단점을 알 수 있었다.

먼저 Regression Tree 방법은 설명변수의 특정 값 기준으로 범위를 나눌 수 있다. 그 후 같은 범위에 해당하는 데이터에 대해서 고차원의 box로 분할이 될 수 있는데 이 box내에 있는 데이터에 대해 y값들의 평균값을 사용한다. 이 Tree를 만드는 방법은 top-down greedy 방식을 사용한다. 트리의 맨 위부터 분할해가기 때문에 top-down 이라고 하며, "greedy"라는 것은 트리를 만드는 과정의 각 단계에서 미리 앞을 내다보고 나중에 나오는 어떤 단계에서 더 나은 트리가 될 수 있도록 분할을 선택하는 것이 아니라, 그 특정 단계에서 가장 좋은 분할을 선택하기 때문이다. 어떠한 Stopping Point를 설정해 준 후 split 변수가 되는 설명변수 Xj 를 선택하고 cutting point(s)를 선택하며 나아간다.

"Tree Prunning" 에 대한 이해도 필요하다. 트리가 너무 복잡할 경우 overfitting 문제가 발생할 수 있다. 따라서 분할 수가 더 작은 트리는 Bias가 증가할 수 있지만 Variance는 더 낮아져서 해석력의 관점에서 더 좋아질 수 있다. 이 방법은 매우 큰 트리 (To)를 만든 후, prune 하여 sub-tree를 만들 수 있다. CV 또는 Validation Approach 를 사용하여 sub-tree 의 testMSE를 계산하는 방법이다. 앞서 Lasso에서와 같이 모형의 복잡성에 대한 penalty alpha를 이용하여 가장 적절한 tree를 만들 수 있다.

Classification Tree 방법은 Regression Tree와 매우 유사한데 반응변수 값을 예측할 때 평균값이 아니라 어떤 class에 속하는 관측치가 가장 많은지에 대해 voting 한다. 즉 오분류율을 이용하여 모형의 성능을 판단한다. Classification 에서 모형의 성능을 판단하기 위해 Node Impurity 를 최소화할 수 있는 측도인 Gini index, Cross-Entropy 값을 이용할 수 있다. 이 측도들은 노드 각각의 Purity 를 가장 높이고 싶어 하는데, 쉽게 말하면 오분류율을 조금 더 높이더라도 한 노드 안에 같은 y값을 더 많이 갖도록 할 수 있다.

기본적인 위의 Regression Tree, Classification Tree방법의 장점은 크게 네 가지로 요약된다. ①설명하기 매우 쉽다 ②인간의 의사결정 과정을 더 밀접하게 반영한다 ③그래프로 나타내기 쉽고 쉽게해석할 수 있다. ④범주형 변수에 대해서 가변수를 만들지 않고도 사용 가능하다. 하지만 이런 장점에도 불구하고 두 가지 단점이 있는데 ①'예측 정확도' 관점에서 회귀보다도 더 좋지 않은 값을 가질 수 있다. ②높은 분산을 가질 수 있다. 따라서 트리 기반의 Bagging , Random Forest, Boosting 방법을 사용하면 이 문제점을 어느 정도 해결 가능하다.

먼저 Bagging 방법은 Bootstrap sample 방법을 이용할 수 있는데, Bootstrap Sample들의 수를 B라고 하면 B개 만큼 각각의 모형을 적합할 수 있고 그 모형으로부터 값의 평균 값을 취할 수 있다. 앞서 설명했던 매우 큰 트리(To)는 prune되지 않은 모형인데 각각의 트리는 분산이 크지만, bias는 작다. 따라서 B개의 트리의 평균을 취하는 것은 분산을 줄일 수 있다. 붓스트랩 표본은 기존 표본의 관측치의

2/3 만큼을 가지고 있다. 따라서, 어떤 관측치 obs.k가 주어져 있을 때 이 관측치를 포함한 표본에 대해서 training set 으로 이용하여 model fitting 을 한 후 이 관측치를 포함하지 않은 1/3 만큼의 표본을 test set으로 하여 예측값을 구할 수 있다. 이 때 1/3 만큼의 표본을 가르켜 OOB(Out-Of Bag) 관측치라고 한다. B가 충분히 큰 경우 OOB오차는 LOOCV(Leave-One-Out Cross-Validation) 오차와 동일하다. Bagging은 결과적으로 예측력은 증가하지만, 해석력이 떨어진다. 하지만 Gini-index를 이용해 변수 중요도는 알 수 있는데, 주어진 설명변수의 분할로 인해서 지니지수가 감소하는 총량을 비교해 볼 수 있다.

Random Forest 방법은 트리들 간의 uncorrelate 방법으로 bagging 보다 더 좋은 성능을 가질 수 있는데, 트리 내에서 split variable 을 기존에는 설명변수만큼 고려했다면 여기서는 m개만의 설명변수를 사용할 수 있다. 만약 하나의 매우 강한 설명변수가 있을 때 항상 이 변수가 split variable 이 될 것이다. 따라서 트리들은 서로 매우 높은 상관을 갖게 되고, single tree와 비교했을 때 분산을 크게 줄이지 못할 것이다. 이런 문제점을 극복할 수 있는데 트리들의 상관성을 제거하여 트리의 평균은 분산이 줄어들고 더 안정적이게 된다. 따라서 B값을 충분히 증가시켜 overfitting 을 없애고 error rate 가 안정화되도록 한다.

Boosting 방법은 앞서 Bagging이 독립적인 모형을 만들어 낸다는 것과는 다르게 이전에 만들어진 트리로부터 정보를 사용하여 만들어진다는 것에 대한 차이가 있다. 이 방법는 "천천히 학습한다"는 개념을 가지고 있다. Y에 대한 것이 아니라 잔차를 반응변수로 하여 이 잔차를 업데이트한다. Shrinkage Parameter lambda는 step size를 의미하며 이 lambda를 이용하여 잔차를 업데이트 해 나간다. Boosting 에서는 3가지 tuning parameter 값을 가지는데, ①트리수(B) – 만약B값이 너무크면 overfitting 발생할 수 있다. ②shrinkage para(lambda) – boosting 이 학습하는 속도를 의미한다. 매우 작은 값을 가진다면, 좋은 성능을 갖기 위해 B값을 키워야 한다. ③트리에서 split수(d) – 복잡도를 제어한다. 상황에 맞게 조절해서 모델의 성능을 가장 높게 할 수 있다. 주로, 부스팅에서는 다른 모형에서보다 더 작은 트리로 성능을 낼 수 있으며, 작은 트리를 사용하기 때문에 모델 해석력에 도움이 될 수 있다.

Ⅳ. Appendix – R code

EX 8
```{r}

library(ISLR)
#a
data(Carseats)
set.seed(1)
train_index= sample(1:nrow(Carseats), nrow(Carseats)/2)
test = Carseats[-train_index,]

#b
tree.carseats = tree(Sales~., data= Carseats,subset =train_index)
pred.tree = predict(tree.carseats, test)
summary(tree.carseats)
# tree plot
plot(tree.carseats)
text(tree.carseats, pretty=0)
# testMSE
mean((pred.tree - test$Sales)^2)

#c
#model fitting
cv.carseats = cv.tree(tree.carseats, FUN = prune.tree)
cv.carseats
#plot -> best size =9
par(mfrow = c(1, 2))
plot(cv.carseats$size, cv.carseats$dev, type = "b")
plot(cv.carseats$k, cv.carseats$dev, type = "b")
par(mfrow = c(1, 1))
# pruning
pruned.carseats = prune.tree(tree.carseats, best = 9)
plot(pruned.carseats)
text(pruned.carseats, pretty = 0)
# predict
pred.prune = predict(pruned.carseats, test)
mean((pred.prune-test$Sales)^2)

#d
library(randomForest)
## bagging model fitting
bag.carseats = randomForest(Sales~.,data =Carseats, subset=train_index, mtry=ncol(Carseats)-1,ntree=500, importance =TRUE)
bag.carseats
#predict
pred.bag = predict(bag.carseats, test)
plot(pred.bag, test$Sales)
abline(0,1)
#testMSE
mean((pred.bag-test$Sales)^2)
#importance PLOT
importance(bag.carseats)
varImpPlot(bag.carseats)

#e
## Random Forest
set.seed(1)
rf.carseats = randomForest(Sales~., data = Carseats, subset = train_index, mtry= 6,ntree = 500,importance=TRUE)
                # 일반적으로 regression tree 만들 때는 p/3개 사용
rf.carseats
#predict
pred.rf = predict(rf.carseats, newdata= test)
#testMSE
mean((pred.rf-test$Sales)^2) # test MSE -> rf가 bagging 보다 더 good!
#importance PLOT
```

```
importance(rf.carseats) # 변수중요도에 대해서 %IncMSE , IncNodePurity값 제공
varImpPlot(rf.carseats)
```


EX 9
```{r}
#a
data(OJ)
head(OJ)
set.seed(1)
train_index= sample(1:nrow(OJ), 800)
test = OJ[-train_index,]
#b
tree.oj = tree(Purchase~., data=OJ[train_index,])
summary(tree.oj)
#c
tree.oj
#d
plot(tree.oj)
text(tree.oj, pretty=0)

#e
pred.tree = predict(tree.oj, newdata =test, type = "class") # type = "class"이용해서 실제 예측값 반환
tree.table = table(pred.tree, test$Purchase)
tree.table

#f
cv.oj = cv.tree(tree.oj, FUN=prune.tree)
cv.oj

#g
par(mfrow = c(1,2))
plot(cv.oj$size, cv.oj$dev, type = "b")
plot(cv.oj$k, cv.oj$dev, type = "b")
par(mfrow = c(1,1))

#h

#i
prune.oj = prune.tree(tree.oj, best=6)

#j
summary(tree.oj)
summary(prune.oj)

#k
#predict
pred.test = predict(tree.oj, newdata =test,type = "class")
prune.pred.test = predict(prune.oj, newdata =test,type="class")
# confusion matrix
unprune.table=table(pred.test, test$Purchase)
prune.table = table(prune.pred.test, test$Purchase)
# misclassification rate
(unprune.table[1,2]+ unprune.table[2,1])/sum(unprune.table)
(prune.table[1,2]+ prune.table[2,1])/sum(prune.table)
```


EX 10 - boosting
```{r, warning=FALSE, error = FALSE}
#a
library(ISLR)
library(dplyr)
data(Hitters)
dim(Hitters)
sum(is.na(Hitters$Salary))
Hitters = Hitters %>% filter(!is.na(Salary)) %>% mutate(Salary = log(Salary))
```

```
dim(Hitters)
sum(is.na(Hitters$Salary))
#b
train_index = c(1:200)
train = Hitters[train_index,]
test = Hitters[-train_index,]

#c
library(gbm)
set.seed(5)
lambdas = 10^seq(-10,0, by = 0.1)
train.errors = rep(0, length(lambdas))
test.errors = rep(0, length(lambdas))
for (i in 1:length(lambdas)) {
    boost.hitters = gbm(Salary ~ .,data=train,distribution="gaussian",n.trees=1000, shrinkage=lambdas[i])
    train.pred = predict(boost.hitters, train, n.trees = 1000)
    test.pred = predict(boost.hitters, test, n.trees = 1000)
    train.errors[i] = mean((train$Salary - train.pred)^2)
    test.errors[i] = mean((train$Salary - test.pred)^2)
}
plot(lambdas, train.errors, type = "b", xlab = "lambda(Shrinkage para)", ylab = "Train MSE", pch = 20)
min(train.errors)
lambdas[which.min(train.errors)]

#d
plot(lambdas, test.errors, type = "b", xlab = "lambda(Shrinkage para)", ylab = "Test MSE", pch = 20)
min(test.errors)
lambdas[which.min(test.errors)]

#e
# lm Fit
lm.fit = lm(Salary ~ ., data = train)
lm.pred = predict(lm.fit,test)
mean((test$Salary - lm.pred)^2) # testMSE in lm
# Lasso Fit
library(glmnet)
set.seed(134)
x = model.matrix(Salary ~ ., data = train)
y = train$Salary
x.test = model.matrix(Salary ~ ., data = test)
ridge.fit = glmnet(x, y, alpha = 0) # Lidge
lasso.fit = glmnet(x, y, alpha = 1) # Lasso
ridge.pred = predict(ridge.fit, s = 0.01, newx = x.test)
lasso.pred = predict(lasso.fit, s = 0.01, newx = x.test)

mean((test$Salary - ridge.pred)^2)
mean((test$Salary - lasso.pred)^2)

#f
boost.best = gbm(Salary ~ ., data = train, distribution = "gaussian",
    n.trees = 1000, shrinkage = lambdas[which.min(test.errors)])
summary(boost.best)

#g

set.seed(1)
rf.hitters = randomForest(Salary ~ ., data = train, ntree = 500, mtry = 19)
rf.pred = predict(rf.hitters, test)
mean((test$Salary - rf.pred)^2)


```