Data Mining

HW#7: Tree-based Methods

학번	182STG18
이름	이하경
제출일	2019.05.08

Description

Tree-based Methods

■ Basic Decision Tree

회귀모형과 분류모형에 대해 트리를 기반으로 한 방법론들을 적용할 수 있다. 가장 기본적인 트리 모형은 설명변수의 공간을 구조화하고 세분화하기 위해 설명변수들을 사용한 적절한 splitting rule 들을 결정하여 관측치들을 여러 개의 구역으로 구분한 뒤, 주어진 관측치에 대해 해당 관측치가 분류되는 구역의 평균(regression) 또는 mode(classification)으로 반응변수의 값을 예측한다. 트리 모형은 간단하고 해석이 용이하다는 장점이 있지만 단일 트리로는 다른 예측 모형에 비해 정확도가 떨어진다. 이를 보완하기 위해 여러 개의 트리를 적합 후 이를 결합해 하나의 예측을 진행하는 방법으로 Bagging 과 Boosting 이 있다. 또한 단일 트리의 과대적합을 방지하기 위한 Pruning 방법이 존재한다.

■ Bagging & Random Forest

단일 트리의 단점은 분산이 크다는 것이다. **Bagging** (Bootstrap Aggregation)은 이를 해결하기 위한 방법으로, 여러 개(B)의 bootstrap 표본을 생성해 트리 모형을 적합하고 각각의 관측치에 대한 예측치 B 개를 생성한 뒤 이들의 평균 또는 mode 로 하나의 예측 값을 만드는 방식으로 예측치의 분산이 줄어들기를 기대한다.

$$\hat{f}_{\text{bag}}(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}^{*b}(x)$$

일반적으로 서로 독립적인 표본의 평균은 분산을 줄이는 것으로 알려져 있다. 그러나 하나의 표본 집단에서 생성한 bootstrap 표본들은 서로 상당히 많은 관측치들을 공통적으로 포함하므로 서로 상관관계가 높다. 이러한 문제점을 해결하기 위해 **Random Forest** 방법은 각각의 bootstrap 표본에 대해 트리 모형을 적합할 때 전체 p 개의 설명변수 중 일부 랜덤한 설명변수 m 개만을 splitting 변수 후보로 사용하여 트리 사이의 correlation 을 줄이는 방법이다. 일반적으로 regression 의 경우 m=p/3, classification 의 경우 m=$\sqrt{p}$를 사용한다.

■ Boosting

Boosting 역시 예측의 결정 트리의 정확도를 향상시키기 위해 만들어진 방법이다. Bagging 에서 각각의 트리는 서로 다른 Bootstrap 표본에서 서로 독립적으로 생성되지만 Boosting 은 bootstrap 표본이 아닌 original 표본으로부터 트리들이 순차적으로 생성되는 점에서 다르게 작용한다. Regression Tree 에서 Boosting 은 이전 step 에서 생성된 예측치와 실제 값의 차이인 잔차에 다시 트리 모형을 적합함으로써 이전 모형이 설명하지 못한 부분을 추가적으로 설명하고 이 트리들을 결합하여 하나의 추정치를 만든다. 따라서 Bagging 과 달리 Boosting 은 과대적합의 위험이 있다.

본 과제에서는 다양한 트리 기반의 방법론들을 사용하여 Chapter 8 의 Lab 및 예제의 데이터들에 대한 Regression 과 Classification 모형을 적합하고 서로 비교한다. 또한, Boosting 방법 중 하나로 속도가 빠르며 예측 성능이 매우 좋다고 알려진 XGBoost(Extreme Gradient Boosting)의 기초를 학습한다.

Results

XGBoost

XGBoost 는 그래디언트 부스팅을 기반으로 한 매우 효율적인 학습 알고리즘으로 이 역시 약한 모형의 추정치를 결합하여 하나의 정확한 예측치를 만드는 방법이다. 이전 모형에 의해 생성된 오차에 대해 새로운 예측 모형을 추가 적합함으로써 모형이 충분히 향상될 때까지 순차적으로 합친다. XGBoost 는 모형의 오버피팅을 방지하기 위해 학습 후반에 낮은 학습률을 적용하는 Regularization technique 과 행과 열에서 모두 subsampling 을 진행한다. 각 모형을 더할 때 오차를 최소화시키기 위한 gradient 를 사용한다. 그래디언트 부스팅 모형에 비해 속도가 매우 빠르고 대규모 데이터 셋을 처리하는 데에도 문제가 없으며 예측의 정확도가 뛰어나다. 다음은 XGBoost 의 주요 장점을 요약한 것이다.

- Parallel Computing: 병렬 연산 및 처리가 가능하며 연산에 모든 CPU core 을 사용할 수 있다.
- Regularization: 기본적인 Gradient Boosting 에서 고려되어 있지 않은 Regularization 을 진행하여 오버피팅을 방지한다.
- Cross Validation: 자체적으로 CV 기능을 제공한다.
- Missing Values: 내부적으로 결측치를 처리하는 기능을 제공한다.
- Flexity: 회귀와 분류 문제뿐만 아니라 사용자 정의 목적함수의 최적화 또한 지원한다.
- Availablity: R, Python, Java 등의 많은 프로그래밍 언어에서 사용할 수 있다.
- Save & Reload: 적합한 모형 또는 데이터 matrix 를 파일로 저장하고 재사용할 수 있다. 대규모 데이터 셋을 처리할 때 시간을 절약하는 방법이다.
- Tree Pruning: negative loss 가 발생하면 pruning 을 중단하는 GBM 과 달리 XGBoost 는 트리를 먼저 설정한 max_depth 까지 깊게 만든 뒤 loss 가 threshold 내에서 증가할 때까지 pruning 을 역으로 진행한다.

XGBoost 에서 조정 가능한 파라미터는 크게 다음의 세 가지 종류로 나뉜다.

General Parameters

Booster (default=gbtree)	사용할 booster의 type을 결정한다. (**gbtree**, gblinear, dart) 분류 모형의 경우 gbtree, 회귀 모형의 경우 모든 옵션을 사용할 수 있다.
nthread (default=maximum cores)	병렬 연산을 활성화한다.
silent (default=0)	1로 설정할 경우 console에 학습 과정이 출력된다.

Booster Parameters

nrounds (default=100)	최대 반복 수를 결정하며 트리의 경우 생성할 트리의 개수와 동일하다.
eta (default=0.3)	학습률, 즉 최적값에 도달하기 위한 shrinkage parameter로 사용한다. 작게 설정할 경우 연산이 느리게 진행되므로 반복수를 충분히 크게 해야한다.
gamma (default=0)	값이 큰 계수에 대해 penalty를 적용해 Regularization 기능을 적용한다.
max_depth (default=6) min_child_weight (default=0)	각 트리 적합 시 최대 사이즈를 조정한다.
subsample (default=1)	각 트리 적합에 사용될 관측치 중 sampling할 비율을 설정한다.
colsample_bytree (default=1)	각 트리 적합에 사용될 설명변수의 개수를 설정한다. 1보다 작은 값일 경우 랜덤 포레스트처럼 일부 설명변수만을 splitting variable로 고려한다.
lambda (default=0)	L2 Regularization (Ridge)
alpha (default=1)	L1 Regularization (Lasso)

Learning Task Parameters

Objective (default=reg:linear)	reg:linear – linear regression binary:logistic – binary classification을 위한 logistic regression으로 예측된 class probability를 반환한다. multi:softmax – softmax를 적용한 multiclassification으로 예측된 class label을 반환한다. multi:softprob – softmax를 적용한 multiclassification으로 예측된 class probability를 반환한다.
eval_metric	mae (Regression), Logloss (Classification), AUC (Classification), RMSE (Regression), error (Binary Classification), mlogloss (Multi-Classification) 모형의 정확도를 검증하기 위해 사용된다.

본 과제에서는 XGBoost 의 Tutorial 과 Parameter Tuning 등을 직접 코드로 실습하여 XGBoost 라이브러리의 다양한 함수를 실제로 사용하고 정리하였다. 예제로 사용된 agaricus 데이터에 대해 위에서 나열한 여러가지 option 을 사용해 목적에 맞는 xgboost 모형을 기본적으로 적합하고 저장 및 예측을 진행해보았다. 또한 adult 데이터에 대해 파라미터 튜닝을 진행하였다. 먼저 default 파라미터로 xgboost 모형을 적합하고 다른 파라미터들은 고정한 상태에서 eta, nrounds 을 조정하거나 두 파라미터를 고정한 채 나머지 파라미터를 조정하는 grid search 를 진행할 수 있다. 이를 통해 모형의 정확도 향상을 기대할 수 있다.

Chapter 8 Lab

결정 트리는 회귀과 분류에 모두 적용이 가능하고 간단히 해석이 가능한 방법이다. Lab에서는 R의 'tree', 'randomForest', 'gbm' 등의 패키지에 내장되어 있는 함수들을 사용해 가장 기본적인 트리 모형부터 배깅 및 랜덤 포레스트, 부스팅과 같은 앙상블 방법론들을 이용한 모형을 적합하였다. 구체적으로 Carseats 데이터의 판매량(Sales)을 두 클래스로 분류하여 높고 낮음을 예측하는 단일 분류 트리 모형과 pruning을 거친 모형을 적합하고, Boston 데이터에는 주택 가격을 예측하는 회귀 모형에 트리 및 배깅, 랜덤 포레스트, 부스팅 모형을 적용하였다.

tree 함수를 사용한 단일 모형은 그래프 및 텍스트를 이용해 결정 트리의 구조 및 splitting rule을 시각적으로 확인할 수 있으며, cv.tree 함수를 이용해 tree의 size, 즉 terminal node의 개수에 따른 예측력을 평가하여 최적의 size를 찾고 pruning을 통해 이미 적합된 트리 모형을 가지치기하여 과대적합을 방지할 수 있다.

앙상블 모형인 배깅 및 랜덤 포레스트 모형의 적합에는 randomForest 함수를 사용할 수 있으며, 조정 가능한 파라미터는 크게 ntree(생성할 Bootstrap 표본 및 결정 트리의 개수)와 mtry(random splitting variable의 개수)이다. mtry에 모형에 사용하는 전체 설명변수의 개수 p를 지정할 경우 배깅, p보다 작은 수를 지정할 경우 랜덤 포레스트 모형이 적합된다. gbm 함수는 부스팅 모형 적합에 이용하는 함수이다.

여러가지 방법론을 사용한 Boston 회귀모형의 경우 부스팅 모형의 test MSE가 가장 작았다. 배깅, 랜덤포레스트, 부스팅 모두 단일 트리에서 예측력이 떨어지는 것을 향상시키기 위한 방법론이지만 이 중 나머지보다 항상 퍼포먼스가 뛰어난 방법론은 없다. 따라서 하나의 데이터에 대해 여러가지 트리 기반 방법론을 적용해보고 서로 비교해볼 수 있다.

단일 트리 모형에서는 결과의 그래프 및 텍스트를 통해 예측의 기준을 해석할 수 있었으며, 앙상블 모형에서 개별 설명변수의 상대적 중요도를 그래프와 수치로 확인할 수 있었다. 이후 실제 데이터 분석에도 트리 기반 모형을 자유롭게 사용할 수 있도록 익숙해지는 계기가 되었다.

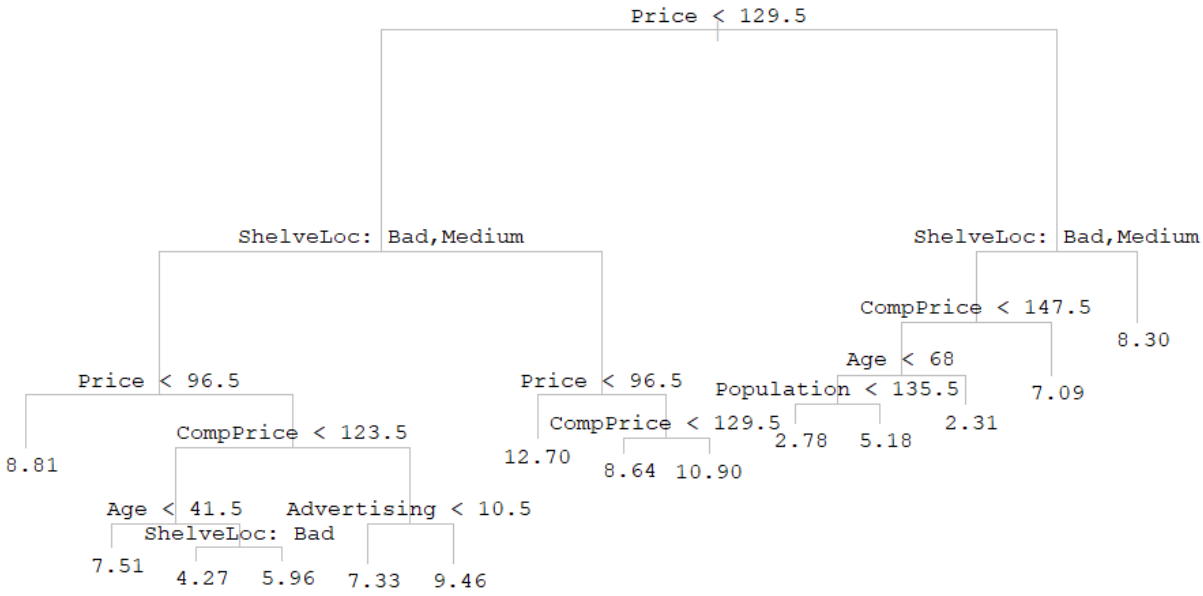
Exercise 8.8 Regression Trees using Carseats Data

(a) Split the Data Set

Carseats 데이터는 총 400개의 observation을 포함하고 있으므로 랜덤하게 training(50%)과 test(50%)으로 각각 200개씩 분리하였다.

(b) Fit a (single) Regression Tree to the training set

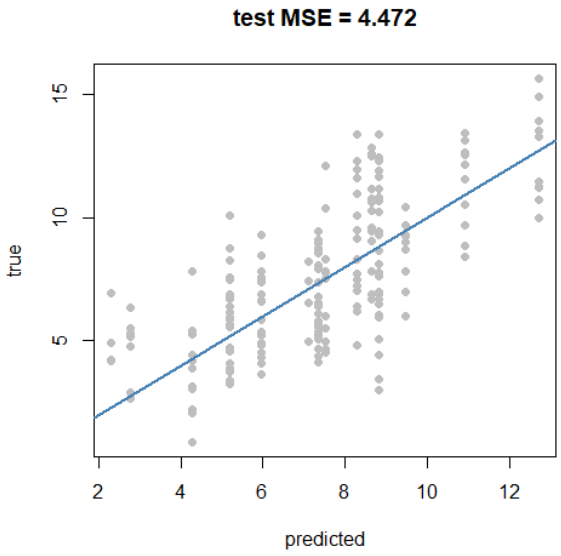
▷ **Plot of a Regression Tree**



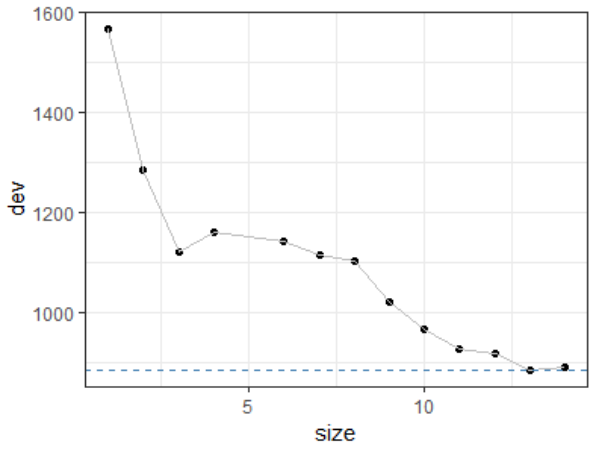
training set 에 대해 적합된 regression tree 는 총 14 개의 terminal node 를 가지고 있으며 **training MSE** 는 **2.602** 로 계산되었다. 기본적인 tree method 는 각각의 split 을 고려할 때 모든 변수를 고려한다. 이중 실제 모형에서 split 기준으로 사용된 변수는 'Price', 'ShelveLoc', 'ComPrice', 'Age', 'Advertising', 'Population'으로 7 개이다.

▷ **Plot of Predicted & True Sales of the test set**

test set 에 대해 모형에서 예측된 sales 값과 실제 sales 를 비교한 그림이다. **test MSE** 는 **4.472** 로 계산되었다.



(c) Cross-Validation to determine the optimal level of tree complexity



Tree 의 Size 에 따른 devience (MSE)를 기준으로 10-folde CV 를 수행하였다. terminal node 가 13 개인 모형에서 total deviance 가 886.83 으로 가장 낮게 계산되었으나 기존의 14 개 모형과 크게 차이가 없다. 따라서 pruning 을 통한 test MSE 의 향상을 기대하기 어려울 것으로 보인다. 실제로 13 개의 terminal node 로 pruning 한 뒤 test MSE 를 계산해보았을 때 4.644 로 나타났다.

(d) Bagging approach

▷ **Summary**

Type of random forest:	**regression**
Number of trees:	**500**
No. of variables tried at each split:	**10**
Mean of squared residuals:	**2.937**
% Var explained:	**60.58**

Carseats 데이터에서 Sales 를 예측하기 위한 설명변수의 총 개수는 10 개이다. mtry=10, ntree=500 으로 하여 Bagging 방법을 사용한 모형을 적합하였다. train MSE 는 2.937 이다.

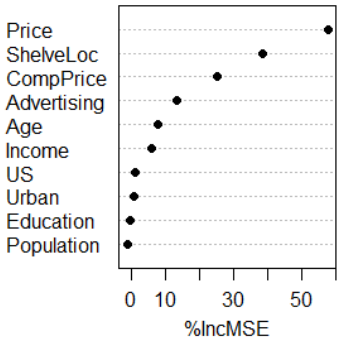
▷ **Test Error (MSE)**

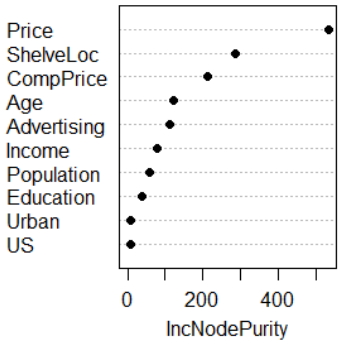
bagging 모형을 이용한 test set 에 대한 예측 결과 single tree 모형보다 정확도가 상당히 상승되었으며 **test MSE** 는 **2.598** 이다.

▷ **Variable Importance**

importance()와 **varImpplot**()을 사용해 중요도가 가장 높은 변수를 확인한 결과 다음과 같이 변수 제거의 따른 MSE 증가율과 노드 불순도의 증가 두 기준 모두에서 **Price** 가 가장 중요한 변수로 나타났다.

	%IncMSE	IncNodePurity
CompPrice	25.241	210.951
Income	5.913	78.449
Advertising	13.574	111.174
Population	-1.057	58.551
Price	**57.773**	**534.070**
ShelveLoc	38.498	285.813
Age	7.895	121.646
Education	-0.066	37.609
Urban	0.889	9.340
US	1.439	8.135





(e) Random Forest approach

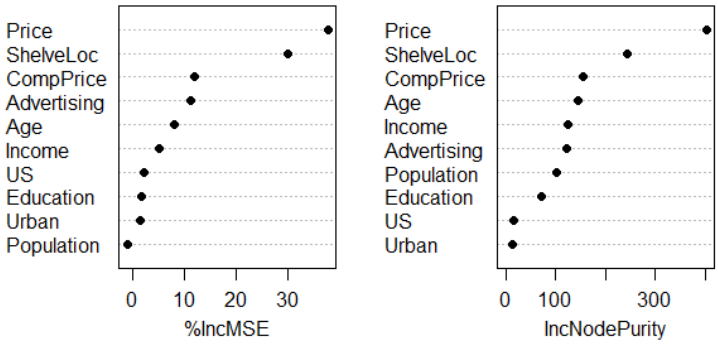
▷ Summary

Type of random forest:	**regression**
Number of trees:	**500**
No. of variables tried at each split:	**3**
Mean of squared residuals:	**3.385**
% Var explained:	**54.56**

Random Forest 는 각각의 split 에서 m 개의 설명변수만을 랜덤하게 선택해 split 기준으로 사용한다. Regression 모형에서 m=p/3=3.3 이므로 mtry=3, ntree=500 으로 랜덤포레스트 모형을 적합한 결과 train MSE 는 3.385 로 배깅에 비해 조금 높게 계산되었다.

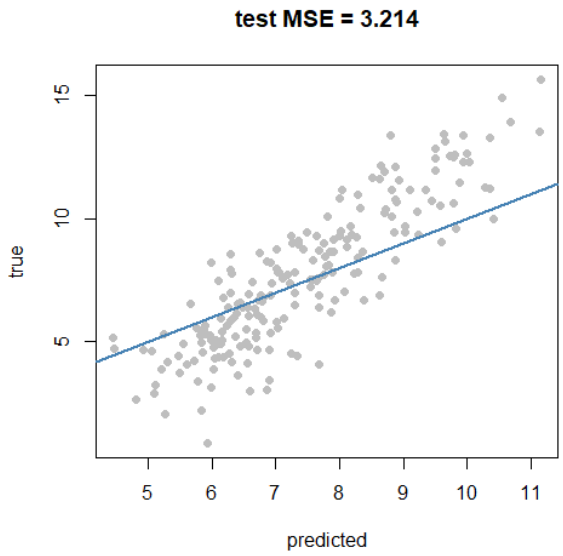
▷ Variable Importance

	%IncMSE	IncNodePurity
CompPrice	12.033	154.866
Income	5.162	124.795
Advertising	11.199	122.440
Population	-1.038	101.029
Price	**37.694**	**402.701**
ShelveLoc	29.942	244.630
Age	8.042	144.084
Education	1.705	71.644
Urban	1.521	13.909
US	2.302	15.933



변수의 중요도는 bagging 과 동일하게 **Price** 가 가장 높게 나타났다.

▷ Test Error (MSE)



랜덤 포레스트 모형의 **test MSE** 는 **3.214** 로 m=3 개의 랜덤한 변수들 만을 split 기준으로 고려한 결과는 bagging 보다 향상되지 않았다.

Exercise 8.9 Classification Trees using OJ Data

(a) Split the data set

전체 1070개의 observation을 800개의 training sample과 나머지 270개의 test sample로 분리하였다.

(b) Fit a single Classification Tree

▷ **Summary Statistics of the Tree**

Classification tree:	
tree(formula = Purchase ~ ., data = OJ, subset = train)	
Variables actually used in tree construction:	**'LoyalCH', 'PriceDiff', 'ListPriceDiff'**
Number of terminal nodes:	**8**
Residual mean deviance	**0.7319 = 579.7 / 792**
Misclassification error rate:	**0.1512 = 121 / 800**

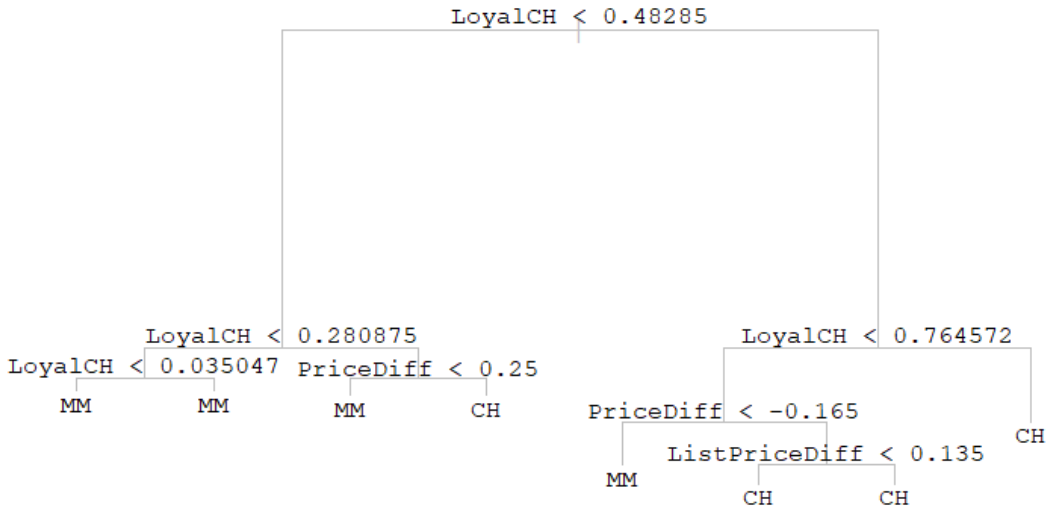
training **error rate**는 **15.12%**로, **terminal node의 개수**는 **8**개인 classification tree 모형이 적합되었다.

(c) Text Output of the Tree

	var	n	dev	yval	splits.cutleft	splits.cutright	yprob.CH	yprob.MM
1	LoyalCH	800	1053.259	CH	<0.48285	>0.48285	0.6313	0.3688
2	LoyalCH	288	321.6892	MM	<0.280875	>0.280875	0.2465	0.7535
4	LoyalCH	159	120.2985	MM	<0.035047	>0.035047	0.1258	0.8742
8	<leaf>	53	9.921596	MM			0.0189	0.9811
9	<leaf>	106	99.69239	MM			0.1792	0.8208
5	PriceDiff	129	173.1388	MM	<0.25	>0.25	0.3953	0.6047
10	<leaf>	84	98.61788	MM			0.2738	0.7262
11	<leaf>	45	59.66692	CH			0.6222	0.3778
3	LoyalCH	512	436.9952	CH	<0.764572	>0.764572	0.8477	0.1523
6	PriceDiff	252	297.7845	CH	<-0.165	>-0.165	0.7222	0.2778
12	<leaf>	39	46.40066	MM			0.2821	0.7179
13	ListPriceDiff	213	211.4973	CH	<0.135	>0.135	0.8028	0.1972
26	<leaf>	36	49.79543	CH			0.5278	0.4722
27	<leaf>	177	144.1535	CH			0.8588	0.1412
7	<leaf>	260	71.45112	CH			0.9692	0.0308

위의 frame에서 <leaf>는 terminal node를 나타낸다. 이 중 하나인 8번 노드의 split 기준은 '**LocalCH < 0.035**'이며, 노드에 포함된 관측치의 개수는 **57**개, deviance는 **9.92**이다. 약 **98%**의 관측치가 **MM**을 구매한 것으로 나타나며 CH를 구매한 관측치는 2%에 불과하다.

(d) Plot of the Tree



적합된 트리 모형의 첫번째, 두번째, 세번째 노드를 분류하는 기준에 모두 'LoyalCH'가 사용되므로 가장 중요한 변수라고 할 수 있다. LoyalCH, 즉 CH(Citrus Hill)에 대한 충성도가 높을수록 CH를 구매한다. 세번째 노드에서 분류 기준으로 PriceDiff가 사용되며 두 제품의 가격차이가 클수록 CH를 구매하였다는 것을 알 수 있다.

(e) Prediction on the Test data

▷ **Confusion Matrix**

오분류율 = 19.63%

	PREDICTED	
TRUE	**CH**	**MM**
CH	132	16
MM	37	85

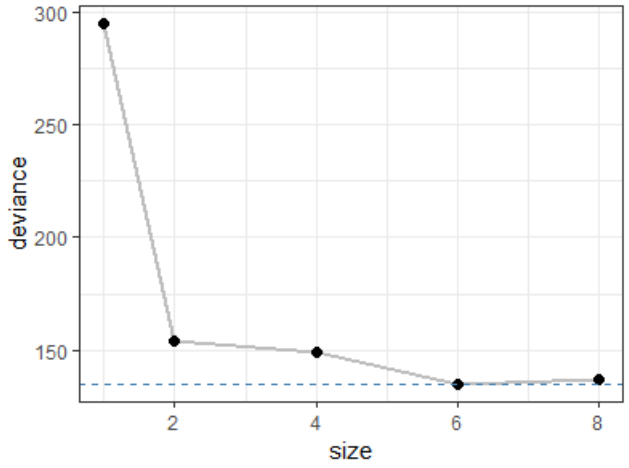
총 270개의 test sample 중 오분류가 발생한 개수는 37+16=53개로 test set에 대한 오분류율은 약 **19.63%**로 나타났다.

(f)-(h) Cross-Validation

▷ **Summary**

size	**8**	**6**	**4**	**2**	**1**
dev	**137**	**135**	**149**	**154**	**295**
k	**-Inf**	**0.0**	**5.5**	**8.5**	**146.0**
method	**'missclass'**				

▷ **Plot of CV Error Rate vs Tree Size**



10-fold CV를 이용해 misclassification rate를 가장 작게 하는 optimal tree size를 찾아보았다. size에 따른 deviance를 그래프로 확인하였을 때 terminal node의 개수가 **6**개일 때의 deviance가 135로 이 때의 CV 오분류율이 135/800=**16.88%**로 가장 작다.

(i)-(j) Pruned tree corresponding to the optimal tree size

(h)에서 최적 트리의 사이즈가 6으로 결정되었으므로 6개의 terminal node를 가지도록 pruning을 진행하였다.

▷ **Summary**

Classification tree:	
tree(tree.oj, nodes = c(4, 13))	
Variables actually used in tree construction:	**'LoyalCH', 'PriceDiff'**
Number of terminal nodes:	**6**
Residual mean deviance	**0.7657 = 607.9 / 794**
Misclassification error rate:	**0.1512 = 121 / 800**

Mean Deviance는 기존의 8개 terminal node를 가진 트리에 비해 약간 증가하여 차이를 보였으나 training error rate는 **15.12%**로 동일하였다.

(k) Test Error Rate using Pruned vs Unpruned Tree

▷ **Confusion Matrix**

오분류율 = 19.63%

	PREDICTED	
TRUE	**CH**	**MM**
CH	132	16
MM	37	85

Pruned Tree를 이용한 Test set 예측 결과 역시 기존의 모형을 이용한 예측과 동일했으며 오분류율은 19.63%이다.

Exercise 8.10 Boosting using Hitters Data

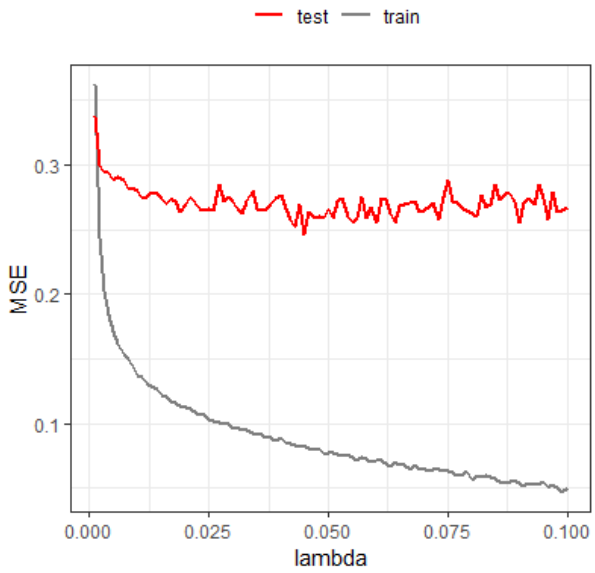
(a) Remove the observations for whom the salary information is unknown, and then log-transform the salaries.

전체 Hitters 데이터는 322 개의 관측치를 가지고 있다. 이중 반응변수 Salary 가 입력되지 않은 관측치는 총 59 개로 이를 제거하고 양수 값만 존재하는 Salary 를 로그 변환하였다.

(b) Create a training set consisting of first 200 observations and a test set consisting of the remaining observations.

결측치를 제거한 데이터 셋은 총 263 개의 관측치를 가지고 있으므로 처음 200 개의 관측치를 training set 으로, 나머지 63 개의 관측치를 test set 으로 분리하였다.

(c)-(d) Perform a Boosting on the training set with 1000 trees for a range of shrinkage parameter λ. Produce a Plot with different shrinkage values and the corresponding (c)training set MSE, and (d)test set MSE.



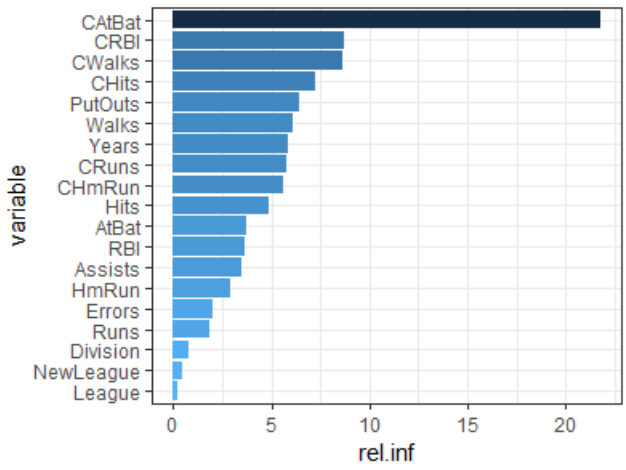
0.001 부터 0.1 사이의 λ의 값들을 이용해 부스팅 모형을 적합하고 training MSE 와 test MSE 를 각각 계산하였다. λ가 커질수록 training MSE 는 계속해서 감소하나 test MSE 는 일정 값 이후부터 거의 감소하지 않는다.

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

Boosting	OLS	Lasso
0.2675	0.4918	0.4548

(d)의 CV 에서 test MSE 가 가장 작은 $\lambda = \mathbf{0.045}$이므로 이 값을 사용해 training set 에 대해 부스팅 모형을 다시 적합하였다. Chapter 3 의 OLS 모형, Chapter 6 의 Model Selection and Regularization 방법론 중 성능이 좋은 편이었던 Lasso 모형을 이용해 log-Salary 에 대한 모형을 적합하여 test MSE 를 비교하였다. Lasso 모형을 이용한 추정 시에는 최적 λ을 찾기 위해 10-fold cross-validation 을 진행한 뒤 test MSE 가 가장 작은 0.035 를 선택하여 적합하였다. 같은 test set 에 대해 계산된 모형 별 MSE 는 위의 표와 같이 부스팅에서 가장 작았다.

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



'CatBat' 변수의 중요도가 가장 높은 것으로 나타난다.

(g) Apply Bagging to the training set. What is the test set MSE?

Hitters 데이터에서 반응변수 Salary 를 제외하고 존재하는 설명변수의 개수 p 는 19 이다. 따라서 mtry=19 로 모든 변수를 split 기준으로 사용한 Bagging 모형을 적합한 결과는 다음과 같다. 해당 모형을 사용한 **test MSE** 계산 결과 **0.2301** 으로 위에서 적합한 다른 모형에 비해 제일 낮았다.

▷ **Summary**

Type of random forest:	**regression**
Number of trees:	**500**
No. of variables tried at each split:	**19**
Mean of squared residuals:	**0.2174**
% Var explained:	**73.87**

Discussion

다양한 데이터 셋에 단일 결정 트리 및 이를 기반으로 한 앙상블 모형을 적합함으로써 회귀와 분류 모형에 쉽고 빠르게 적용 가능하면서도 선형 모형 및 정규화된 선형 모형 등의 기존 방법론에 비해 예측력이 우수한 것을 직접 확인할 수 있었다. 또한 배깅 및 랜덤 포레스트, 부스팅 모형에서 각 설명변수의 상대적 중요도를 계산하고 중요도가 높은 일부 변수들의 partial dependence plot 을 그려 다른 설명변수들이 불변할 때 해당 설명변수의 크기에 따른 반응변수 값의 변화를 확인함으로써 영향력을 판단할 수 있다는 것을 알게 되었다. 본 과제를 통해 현재 Kaggle 을 비롯한 데이터 분석 문제에서 인기가 높으며 성능이 뛰어나기로 유명한 XGBoost 를 포함하여 다양한 트리 기반의 모형을 실제 데이터에 적용하는 데 큰 도움을 얻을 것이다.

[Appendix] R code

Exercise 8.8

```
# Quick Functions
regFitPlot <- function(pred, true) {
  mse <- mean((true - pred)^2)
  plot(pred, true, pch = 19, col = 'gray', xlab = 'predicted',
       main = glue('test MSE = {round(mse, 3)}'))
  abline(0, 1, col = 'steelblue', lwd = 2)
  return(mse)
}

data(Carseats) ;dim(Carseats)

# (a) split the dataset
set.seed(2)
train <- sample(nrow(Carseats), nrow(Carseats)/2)
carseats.test <- Carseats[-train,]
sales.test <- Carseats$Sales[-train]

# (b) fit a regression tree: plot, test MSE
summary(tree.carseats <- tree(Sales ~ ., Carseats, subset = train))
plot(tree.carseats, col = 'gray')
text(tree.carseats, digits = 3, pretty = 0, font = 10)

pred.tree <- predict(tree.carseats, carseats.test)
(mse.tree <- regFitPlot(pred.tree, sales.test))

# (c) cv.tree
set.seed(20)
(cv.tree.carseats <- cv.tree(tree.carseats, FUN = prune.tree))

cv.table <- data.frame(cv.tree.carseats[1:3])
cv.table[which.min(cv.table$dev),]
plot(cv.tree.carseats)
ggplot(cv.table, aes(size, dev)) + theme_bw() +
  geom_point() + geom_line(col = 'gray') +
  geom_hline(aes(yintercept = min(dev)), linetype = 2, col = 'steelblue')

pred.tree <- predict(prune.tree(tree.carseats, best = 13), carseats.test)
(mse.tree <- regFitPlot(pred.tree, sales.test))

# (d) bagging
p <- ncol(Carseats) -1
set.seed(2)
(bag.carseats <- randomForest(Sales ~ ., Carseats, subset = train,
                              mtry = p, importance = TRUE))
importance(bag.carseats)
varImpPlot(bag.carseats, pch = 19, main = 'Variable Importance')

pred.bag <- predict(bag.carseats, carseats.test)
(mse.bag <- regFitPlot(pred.bag, sales.test))

# (e) random forest (mtry = 3 = p/3)
set.seed(2)
(rf.carseats <- randomForest(Sales ~ ., Carseats, subset = train,
                             importance = TRUE))
varImpPlot(rf.carseats, pch = 19, main = 'Variable Importance')

pred.rf <- predict(rf.carseats, carseats.test)
(mse.rf <- regFitPlot(pred.rf, sales.test))

c(mse.tree, mse.bag, mse.rf)
```

Exercise 8.9

```
data(OJ) ; dim(OJ)

# (a) split the data set
set.seed(9)
train <- sample(nrow(OJ), 800)
oj.test <- OJ[-train,]
purchase.test <- OJ$Purchase[-train]

# (b) fit a single tree
summary(tree.oj <- tree(Purchase ~ ., OJ, subset = train))

# (c) text output
tree.oj$frame

# (d) plot
plot(tree.oj, col = 'gray')
text(tree.oj, digits = 3, pretty = 0, font = 10)

# (e) prediction
pred.tree <- predict(tree.oj, oj.test, type = 'class')
table(purchase.test, pred.tree)
mean(purchase.test != pred.tree)

# (f) cv.tree
set.seed(9)
(cv.tree.oj <- cv.tree(tree.oj, FUN = prune.misclass))

# (g) plot
cv.table <- data.frame(cv.tree.oj[1:3])
ggplot(cv.table, aes(size, dev)) + theme_bw() +
  geom_line(col = 'gray', size = 1) + geom_point(size = 2) +
  geom_hline(aes(yintercept = min(dev)), linetype = 2, color = 'steelblue') +
  labs(y = 'deviance')

# (h) min error rate (deviance)
cv.table[which.min(cv.table$dev),]

# (i)-(j)
summary(prune.oj <- prune.misclass(tree.oj, best = 6))

# (k) test error rate
pred.prune <- predict(prune.oj, oj.test, type = 'class')
table(purchase.test, pred.prune)
mean(purchase.test != pred.prune)
```

Exercise 8.10

```
data(Hitters) ; dim(Hitters)

# (a)
sapply(Hitters, function(x) sum(is.na(x)))
hitters <- Hitters %>% na.omit %>%
                mutate(log.Salary = log(Salary)) %>% select(-Salary)

# (b)
train <- 1:200
hitters.train <- hitters[train,]
hitters.test <- hitters[-train,]
y.train <- hitters.train$log.Salary
y.test <- hitters.test$log.Salary

# (c)-(d)
lambda <- seq(0.001, 0.1, length = 100)
mse.lambda <- NULL
for (i in lambda) {
  bst.hitters <- gbm(
    log.Salary ~ ., data = hitters.train,
    distribution = 'gaussian', n.trees = 1000, shrinkage = i
  )
  pred.train <- predict(bst.hitters, hitters.train, n.trees = 1000)
  pred.test <- predict(bst.hitters, hitters.test, n.trees = 1000)
  mse.train <- mean((y.train - pred.train)^2)
  mse.test <- mean((y.test - pred.test)^2)
  mse.lambda <- rbind(mse.lambda, c(mse.train, mse.test))
}
mse.lambda <- data.frame(lambda, mse.lambda)
colnames(mse.lambda) <- c('lambda', 'train', 'test')
mse.lambda %>% filter(test == min(test))
best.lmabda <- mse.lambda$lambda[which.min(mse.lambda$test)]

ggplot(mse.lambda) + theme_bw() +
  geom_line(aes(lambda, train, col = 'train'), size = 1) +
  geom_line(aes(lambda, test, col = 'test'), size = 1) +
  scale_color_manual('', values = c(train = 'gray50', test = 'red')) +
  labs(y = 'MSE') + theme(legend.position = 'top') +
  geom_vline(aes(xintercept = best.lmabda), linetype = 2)

# (e)
set.seed(10)
bst.hitters <- gbm(
  log.Salary ~ ., data = hitters.train,
  distribution = 'gaussian', n.trees = 1000, shrinkage = best.lambda
)
pred.bst <- predict(bst.hitters, hitters.test, n.trees = 1000)

# linear regression (CH3)
summary(lm.hitters <- lm(log.Salary ~ ., hitters.train))
pred.lm <- predict(lm.hitters, hitters.test) %>% as.numeric

# lasso regression (CH6)
X.train <- model.matrix(lm.hitters)
X.test <- model.matrix(log.Salary ~ ., hitters.test)
lasso.hitters <- glmnet(X.train, y.train, alpha = 1)
set.seed(10)
cv.lasso <- cv.glmnet(X.train, y.train, alpha = 1)
pred.lasso <- predict(lasso.hitters, X.test, s = cv.lasso$lambda.min,
                      exact = TRUE) %>% as.numeric

result <- data.frame(pred.bst, pred.lm, pred.lasso)
sapply(result, function(x) mean((y.test - x)^2))

# (f)
bst.tb <- summary(bst.hitters)
ggplot(bst.tb) + theme_bw() + coord_flip() +
  geom_col(aes(fct_reorder(var, (rel.inf)), rel.inf, fill = desc(rel.inf))) +
  labs(x = 'variable') + theme(legend.position = '')

plot(bst.hitters, i = 'CAtBat', lwd = 2) # partial dependence plot

# (g) bagging
p <- ncol(hitters) -1
set.seed(10)
(bag.hitters <- randomForest(log.Salary ~ ., hitters.train,
                             mtry = p, importance = TRUE))
pred.bag <- predict(bag.hitters, hitters.test)
mean((y.test - pred.bag)^2)
```