5. 의사결정나무와 앙상블

5.1. 의사결정나무

'스무고개'는 예/아니오로 대답할 수 있는 질문을 최대 20개까지 던져 상대방이 생각하고 있는 답을 맞추는 놀이다. 흔히 가장 먼저 던지는 질문은 이런 것이다.

> 그것은 생물입니까?

만약 '예'라고 한다면 이런 질문을 던질 수 있을 것이다.

> 그것은 동물입니까?

처음의 질문에 '아니오'라고 한다면 이런 질문을 할 수 있다.

> 자연물입니까?

스무고개의 질문들을 그림으로 그려보면, 질문 하나마다 가지를 치면서 마치 나무를 뒤집어 놓은 것과 같은 모양이 된다.

이러한 스무고개 방식의 기계학습 모형을 의사결정 나무(decision tree)라고 한다. 의사결정 나무는 회귀의 경우 평균값을, 분류의 경우에는 가장 확률이 높은 클래스를 예측한다.

5.1.1. 학습

스무고개에서 "생물입니까?"와 같은 질문을 먼저 하는 이유는 가능한 많은 경우의 수를 제외할 수 있기 때문이다. "이 방에 있는 것입니까?" 같은 질문을 처음부터 했다가 '아니오'라고 하면 별로 도움이 되지 않는다.

의사결정나무의 학습의 원리도 비슷하다. 의사결정나무에서 각각의 질문은 예/아니오에 따라 데이터를 나누게 된다. 이때 Gini impurity, Information gain, Variance reduction 등 다양한 지표로 데이터가 잘 나눠졌는지를 따진다. 그리고 현 단계에서 데이터를 가장 잘 나누는 질문을 선택한다. 이를 반복함으로써 의사결정나무 전체가 만들어진다. 의사결정나무를 만드는 구체적인 알고리즘으로는 ID3, C4.5, CART 등등이 있다.

5.1.2. 장단점

의사결정 나무는 여러 가지 장점이 있다. 복잡한 통계적 가정이 없기 때문에 모형을 이해하기도 쉽고, 전처리도 단순하다. 속도도 빠르고 다양한 종류의 변수를 다룰 수 있다. 그러면서도 복잡한 패턴을 나타낼 수 있다.

하지만 데이터가 조금만 달라져도 학습된 결과가 크게 바뀌고, 과적합이 일어나기 쉽다는 문제가 있다. 또한 여러 변수를 동시에 고려해야는 문제(예: XOR 문제)에 약하고 비대칭 자료(imbalanced data)를 잘 다루지 못한다.

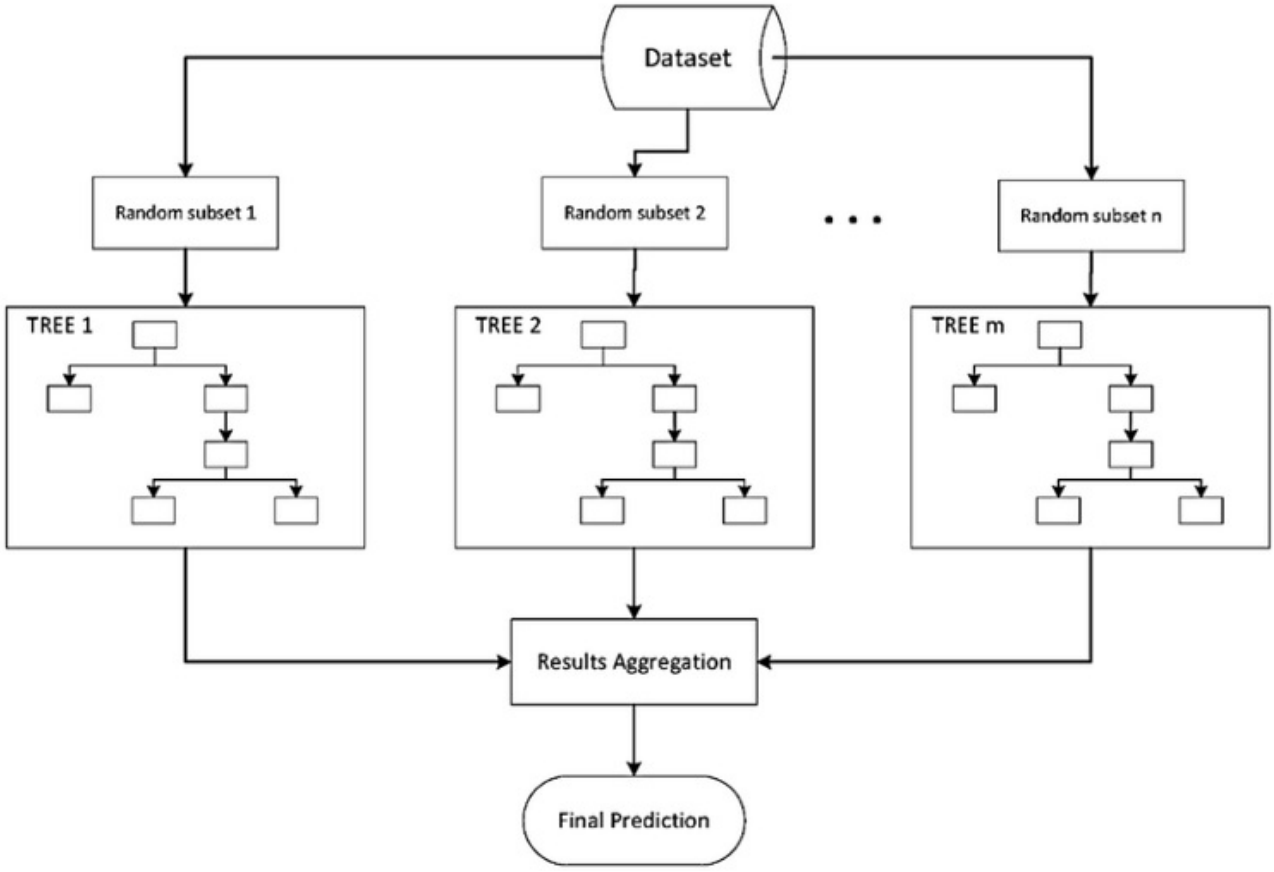
5.2. 앙상블

앙상블(ensemble)은 여러 개의 모형을 만들어 각 모형의 예측을 다수결이나 평균하는 방법이다. 하나의 모형만을 학습시키는 것보다 대체로 나은 결과를 보여준다.

앙상블에는 배깅(bagging), 부스팅(boosting), 스태킹(stacking) 등의 방법이 있다.

5.2.1. 배깅

배깅은 다음과 같이 한다. 먼저 데이터에서 일부를 샘플링 하는 부트스트랩(bootstrap)을 실시한다. 이렇게 만든 각각의 부트스트랩 샘플에 모형을 각각 학습시킨다. 그러면 하나의 데이터셋에서 여러 개의 샘플을 만들어 모형을 여러 번 학습시킬 수 있다. 이들의 예측을 합쳐 최종 예측을 한다. 배깅이라는 이름은 Bootstrap AGGregation에서 온 것이다.

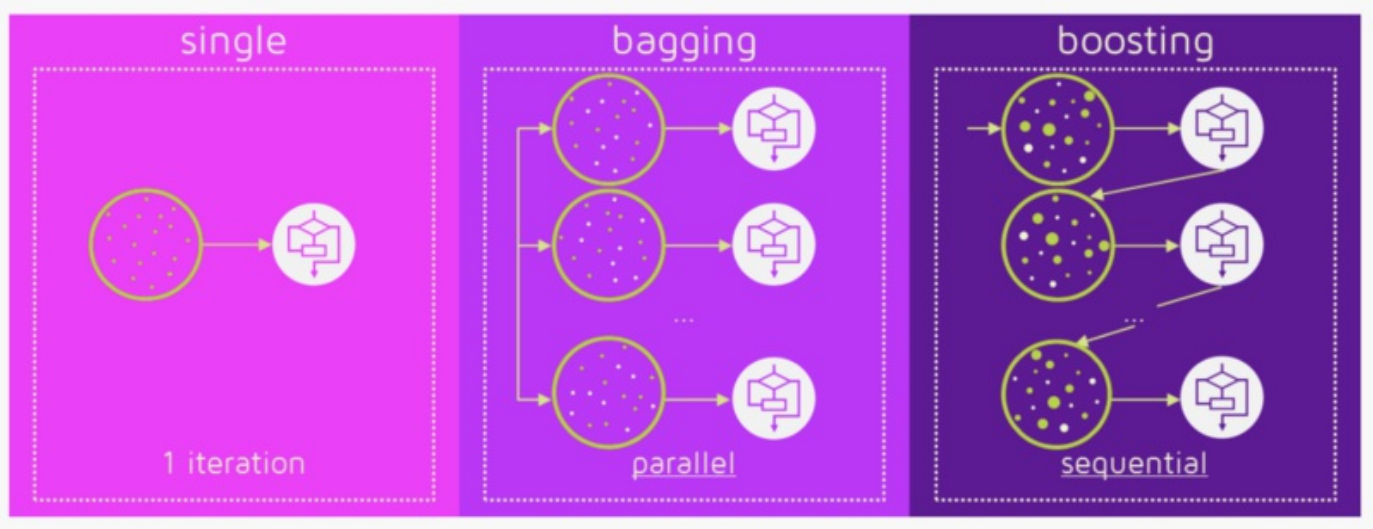


배깅은 의사결정나무와 같이 불안정한(unstable) 모형에 좋다. 반대로 안정한 모형의 경우에는 인위적으로 여러 샘플을 만들어 결과를 부정확하게 만들 수 있다.

랜덤 포레스트

랜덤 포레스트(random forest)는 의사결정나무에 배깅을 적용한 모형이다. 랜덤 포레스트에서 각각의 나무는 표준적인 배깅과 달리 무작위로 선택된 특성을 이용한다. 이를 특성 배깅(feature bagging)이라고 한다. 부트스트랩을 해서 샘플마다 차이가 있어도 예측력이 높은 특성이 있으면 모든 나무들이 같은 특성을 사용하기 때문에 앙상블을 하는 의의가 없다. 그래서 나무에서 사용할 특성을 무작위로 선택해서 다양한 나무를 만드는 것이다.

5.2.2. 부스팅

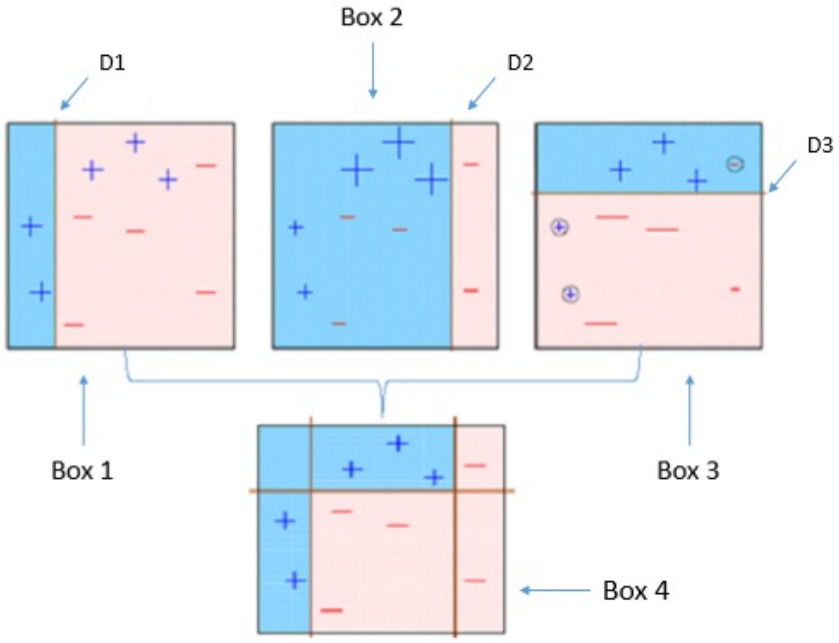


부스팅은 약한 학습 결과를 합쳐 강한 학습 결과를 만드는 방법이다.

AdaBoost

먼저 AdaBoost 알고리즘을 알아보자.

AdaBoost 알고리즘은 일단 데이터셋에 하나의 모형을 학습시킨다(Box 1). 그 학습 결과로 예측하여 틀린 사례에 큰 가중치를 주고 다시 학습을 시킨다(Box 2). 이 과정을 반복하여(Box 3) 학습 결과들을 모두 합치는 것(Box 4)이다.



경사 부스팅

최근에 각광을 받는 방법은 경사 부스팅(Gradient Boosting)이다. 경사 부스팅은 이전 학습 결과와 실제의 차이를 학습하는 방식이다. 예를 들어 첫번째 학습 결과 f_1이 실제값 Y와 다음과 같은 관계가 있다고 해보자.

$$Y = f_1(X) + \epsilon_1$$

여기서 ϵ_1는 예측 오차이다. 이제 다시 두번째 학습 결과 f_2가 이 예측 오차를 학습하게 한다. 그러면 다음과 같은 관계가 생긴다.

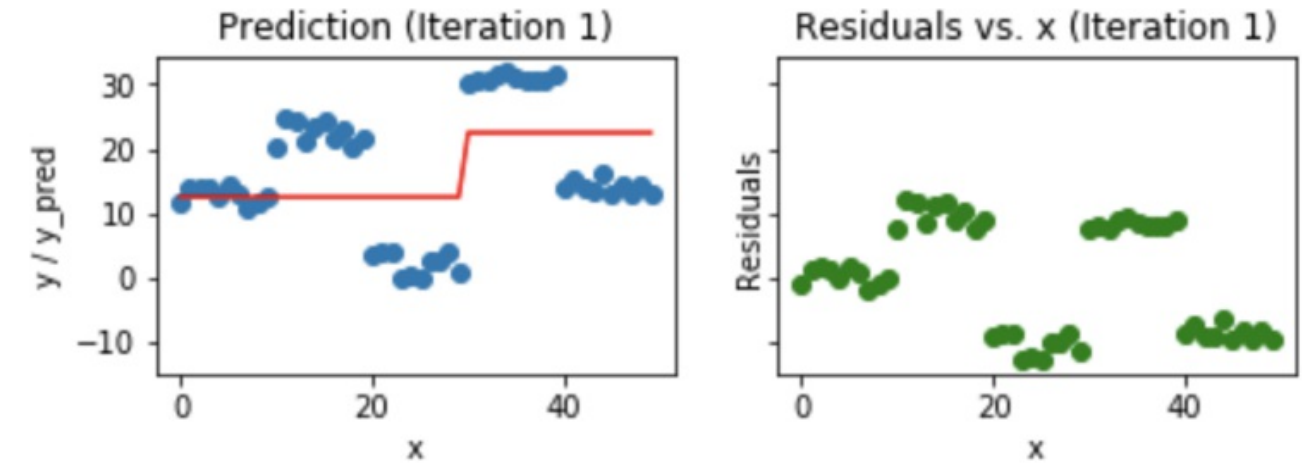
$$\epsilon_1 = f_2(X) + \epsilon_2$$

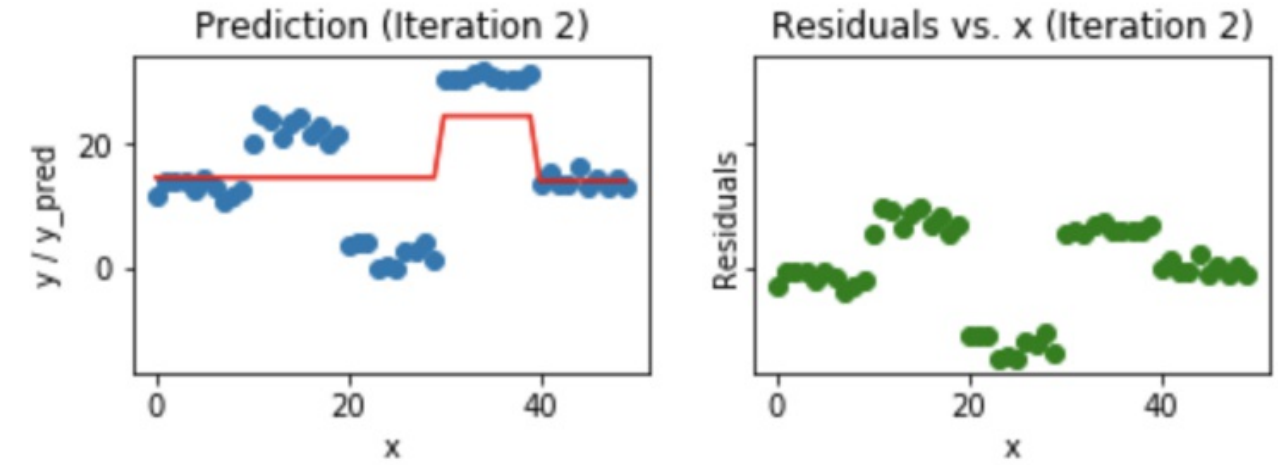
학습이 잘되었다면 첫번째 학습의 오차 분산 $Var(\epsilon_1)$보다 두번째 학습의 오차 분산 $Var(\epsilon_2)$가 더 줄어들게 된다.(결정계수 R^2의 논리)

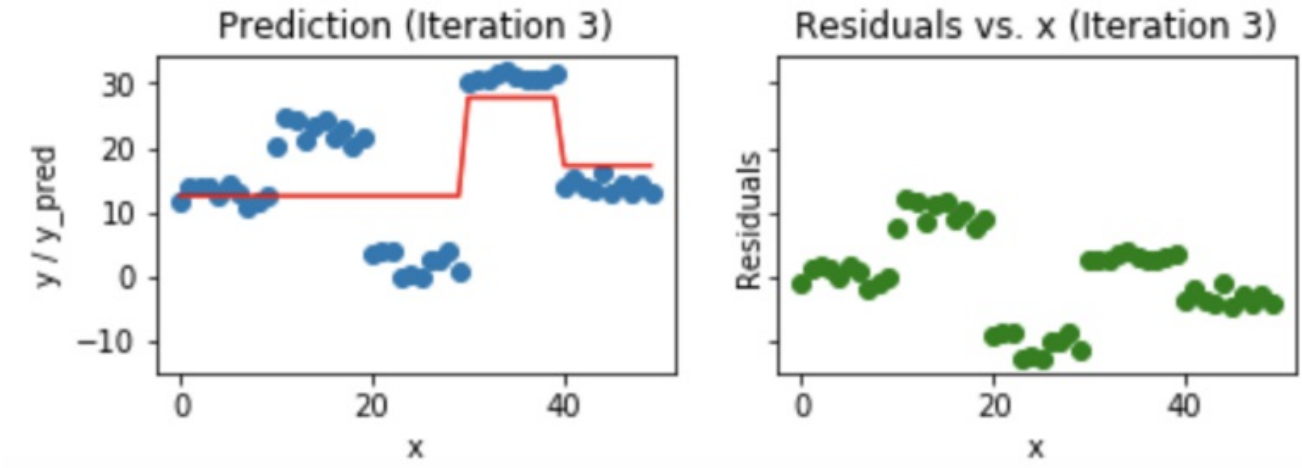
이제 두 번째 식을 첫 번째 식에 대입하면 다음과 같이 된다.

$$Y = f_1(X) + f_2(X) + \epsilon_2$$

이와 같은 과정을 반복하여 오차를 점점 줄여가는 방식이다. 아래 그림은 이를 도식화한 것이다. 왼쪽의 파란점은 실제값, 빨간 선은 예측값, 오른쪽의 초록색 점은 예측 오차를 나타낸다.







5.2.3. 스태킹

배깅과 부스팅은 동일한 모형을 여러 번 학습시켜 그 결과를 결합하는 방법이다. 반면 스태킹은 서로 다른 모형들을 학습시킨다.

$$\hat{y}_f = f(X) \hat{y}_g = g(X)$$

또한 단순한 평균이나 다수결 대신 새로운 모형(예: 선형 모형)으로 실제값을 예측한다.

$$\hat{y} = h(\hat{y}_f, \hat{y}_g)$$

5.3. 의사결정 나무 실습

5.3.1. 데이터 불러오기

UCI Machine Learning Repository의 Wine Quality 데이터를 연다

```
red.url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-red.csv'
white.url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-white.csv'

red.wine = read.csv(url(red.url), sep = ';')
white.wine = read.csv(url(white.url), sep = ';')
```

데이터에 `color`라는 컬럼을 추가하고 와인 색상 값을 넣는다.

```
red.wine$color = 'red'
white.wine$color = 'white'
```

레드 와인 데이터와 화이트 와인 데이터를 합쳐 `wine`이라는 하나의 데이터로 만든다.

```
wine = rbind(red.wine, white.wine)
```

5.3.2. 데이터 분할

수업에서 결과를 일정하게 만들기 위해 난수의 시드 값을 고정한다.

```
set.seed(1234)
```

caret 패키지를 불러들인다.

```
library(caret)
```

```
Loading required package: lattice
Loading required package: ggplot2
Warning message:
: package 'ggplot2' was built under R version 3.3.3
```

`wine` 데이터의 색상(`color`)를 기준으로 80%는 훈련용, 20%는 테스트용으로 데이터를 분할한다..

```
idx = createDataPartition(wine$color, p=.8, list=F)
```

`idx`에 해당하는 행의 데이터는 `data.train`으로, 나머지 행의 데이터는 `data.test`로 분할한다.

```
data.train = wine[idx, ]
data.test = wine[-idx, ]
```

5.3.3. 교차 검증 설정

훈련시 교차검증은 5-fold CV로 한다. 즉, 훈련용 데이터를 무작위로 5개로 나누어 4개로 학습 후 1개로 검증하는 과정을 총 5회 반복한다. 이를 통해 성능이 가장 좋은 하이퍼 파라미터를 선택한다.

```
control = trainControl(method='cv', search='grid', number=5)
```

5.3.4. 랜덤 포레스트

랜덤 포레스트로 학습한다. 랜덤 포레스트의 하이퍼파라미터는 다음과 같다.

- mtry: 각 트리를 만들 때 사용할 변수의 수

```
rf.model = train(
    color ~ .,
    data = data.train,
    tuneGrid = data.frame(.mtry=c(4, 6, 8)),
    trControl = control,
    method = 'rf')
```

성능평가

학습된 모형을 테스트 데이터에 적용해 색상을 예측한다.

```
color.rf = predict(rf.model, data.test)
```

예측된 색상과 실제 색상을 비교하여 혼돈 행렬을 만들고 여러 가지 지표로 성능을 확인한다.

```
confusionMatrix(color.rf, data.test$color)
```

```
Confusion Matrix and Statistics

          Reference
Prediction red white
     red   312     1
     white   7   978

               Accuracy : 0.9938
                 95% CI : (0.9879, 0.9973)
    No Information Rate : 0.7542
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.9833
 Mcnemar's Test P-Value : 0.0771

            Sensitivity : 0.9781
            Specificity : 0.9990
         Pos Pred Value : 0.9968
         Neg Pred Value : 0.9929
             Prevalence : 0.2458
         Detection Rate : 0.2404
   Detection Prevalence : 0.2411
      Balanced Accuracy : 0.9885

       'Positive' Class : red
```

```
varImp(rf.model)
```

```
rf variable importance

                     Overall
chlorides            100.000
total.sulfur.dioxide  98.530
volatile.acidity      34.705
sulphates             14.836
density               13.650
free.sulfur.dioxide   12.931
fixed.acidity         10.677
residual.sugar         9.468
pH                     4.710
citric.acid            2.775
alcohol                1.882
quality                0.000
```

5.3.5. XGBoost

XGBoost의 하이퍼파라미터는 아래와 같다.

- nrounds: 최대 반복 횟수 (> 0)
- eta 학습률. eta를 낮추면 모형의 과적합 가능성은 낮아지지만 학습 속도가 느려진다.(0 ~ 1)
- gamma 트리에서 가지를 추가로 치기 위해 필요한 최소한의 손실 감소 기준. 기준값이 클 수록 모형이 더 단순해진다.(> 0)
- max_depth 트리의 최대 깊이.(> 0)
- min_child_weight 트리에서 가지를 추가로 치기 위해 필요한 최소한의 사례 수.(> 0)
- colsample_bytree 각각의 트리를 만들 때 데이터에서 사용할 열(column)의 비율(0 ~ 1)

```
xgb.grid = expand.grid(
    .nrounds = 100,
    .eta = c(0.1, 0.3, 0.5),
    .gamma = 1,
    .max_depth = c(3, 5),
    .min_child_weight = 1,
    .colsample_bytree = 1
)
```

경사 부스팅 모형을 학습시킨다.

```
xgb.model <- train(
    color ~ .,
    data = data.train,
    tuneGrid = xgb.grid,
    trControl = control,
    method = 'xgbTree'
)
```

```
Loading required package: xgboost
Warning message:
: package 'xgboost' was built under R version 3.3.3Loading required package: plyr
```

성능평가

학습된 모형을 테스트 데이터에 적용해 색상을 예측한다.

```
color.xgb = predict(xgb.model, data.test)
```

예측된 색상과 실제 색상을 비교하여 혼돈 행렬을 만들고 여러 가지 지표로 성능을 확인한다.

```
confusionMatrix(color.xgb, data.test$color)
```

```
Confusion Matrix and Statistics

          Reference
Prediction red white
     red   315     1
     white   4   978

               Accuracy : 0.9961
                 95% CI : (0.991, 0.9987)
    No Information Rate : 0.7542
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.9896
 Mcnemar's Test P-Value : 0.3711

            Sensitivity : 0.9875
            Specificity : 0.9990
         Pos Pred Value : 0.9968
         Neg Pred Value : 0.9959
             Prevalence : 0.2458
         Detection Rate : 0.2427
   Detection Prevalence : 0.2435
      Balanced Accuracy : 0.9932

       'Positive' Class : red
```

```
varImp(xgb.model)
```

```
xgbTree variable importance

                     Overall
chlorides            100.0000
total.sulfur.dioxide  85.6656
volatile.acidity      11.5869
density                6.5538
sulphates              5.0458
fixed.acidity          3.3353
pH                     3.0052
residual.sugar         2.8373
alcohol                1.9034
citric.acid            0.9937
free.sulfur.dioxide    0.5186
```