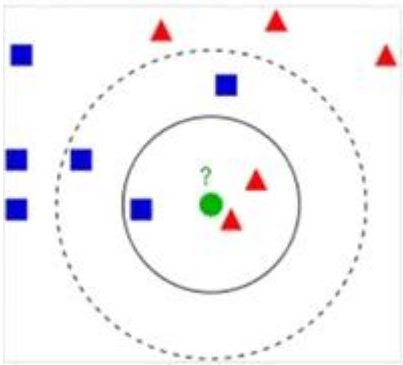
7장 k-인접이웃분류

7.1 서론
k-인접이웃(k-nearest neighbor, 이하k-NN로 사용)분류모형이란 새로운 데이터(설명변수값)에 대해
이와 가장 유사한(거리가 가까운) k-개의 과거자료(설명변수값)의 결과(반응변수: 집단)를 이용하여 다수결로 분류한다.
과거 자료를 이용하여 미리 분류모형을 수립하는 것이 아니라, 과거 데이터를 저장만 해두고 필요 시 비교를 수행하는 방식이다.
k 값의 선택에 따라 새로운 데이터에 대한 분류결과가 달라짐에 유의하여야 한다.
k-NN은 반응변수가 범주형인 경우에는 분류의 목적으로, 연속형인 경우에는 회귀의 목적으로 사용할 수 있다.

7-2 k-인접이웃 분류
k-NN은 기계학습 분야에서 가장 단순한 알고리즘이다.
이 알고리즘은 지역 정보만으로 근사되며, 모든 계산이 이루어진 후에 분류가 이루어지는 특징으로 인해
사례-기반 학습 또는 게으른 학습의 한 유형으로 볼 수 있다.
사례-기반 학습은 메모리에 저장되어 있는 과거의 train data로부터 직접 결과를 도출되므로 메모리-기반 학습이라고도 한다



k-NN 분류의 예는 옆의 그림과 같다.
이 그림에서 검증용 자료(중앙의 점)는 1그룹(사각형) 또는 2그룹(삼각형)으로 분류된다.
만약 k=3(실선의 원)이라면 이 자료는 2그룹으로 분류되며
k=5(점선의 원)이라면 1그룹으로 분류된다.
k-NN은 분류와 회귀 모두에서 주변 값들의 기여도에 가중을 부여할 수 있다.
즉, 더 가까운 주변일수록 더 큰 가중을 부여한다.
예를들어, 각 주변 점에 대해 새로운 점과의 거리의 역수(1/d)를 주변 점의 가중치로 하는 방법이다.
k-NN의 단점으로 데이터의 지역 구조에 민감한 점을 들 수 있다.

예제 1 : {class}knn() 함수를 사용하여 k-NN 분류를 수행한다.
데이터를 불러와서 검증용 데이터와 훈련용 데이터를 각각 75개씩 설정
```
library(class)
data(iris3)
train <- rbind(iris3[1:25,,1], iris3[1:25,,2], iris3[1:25,,3])
test <- rbind(iris3[26:50,,1], iris3[26:50,,2], iris3[26:50,,3])
cl <- factor(c(rep("s",25), rep("c",25), rep("v",25))) # 각 데이터에 요소를 대입한다. 예) Virginica는 v로 표시
# [1] s s s ... c c c ... v v v
```

{class}knn() 함수를 사용하여 k-NN 분류를 수행한다.
knn(훈련용데이터, 테스트데이터, class변수, k=분류의 수, prob=TRUE)
```
knn(train, test, cl, k = 3, prob=TRUE) # 마지막 값이 해당 class분류(v)로 분류될 확률이 0.6666667이라는 뜻이다.
# 결과
# [1] s s s s s ...v v v v v
# attr(,"prob")
# [1] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 ... 1.0000000 0.6666667 1.0000000 1.0000000 0.6666667
# Levels: c s v
```

예제 2 : {DMwR2}kNN() 함수를 이용하여 k-인접이웃분류를 시행한다.
# KNN()함수는 Knn()함수와 유사하나, 모형식 기반으로 수행되는 차이점이 있으며, 자료에 대한 정규화(norm=) 옵션을 제공한다.
install.packages("DMwR2")
library(DMwR2)

데이터를 불러와서 훈련용 데이터와 검증용 데이터로 분류한다.
data(iris)
idxs <- sample(1:nrow(iris), as.integer(0.7*nrow(iris)))
trainIris <- iris[idxs,] # 훈련용 데이터 150 * 0.7 = 105개 할당
testIris <- iris[-idxs,] # 검증용 데이터 150 * 0.3 = 45개 할당

모형식 기반(Species ~ .)인 {DMwR2}kNN() 함수를 사용하여 k-인접이웃분류를 시행
nn3 <- kNN(Species ~ ., trainIris, testIris, k=3) # [1] setosa ... virginica
table(testIris[,'Species'], nn3) # 검증용데이터에 적용하여 테이블로 표시
결과 : nn3
setosa versicolor virginica
setosa 15 0 0
versicolor 0 13 0
virginica 0 0 17

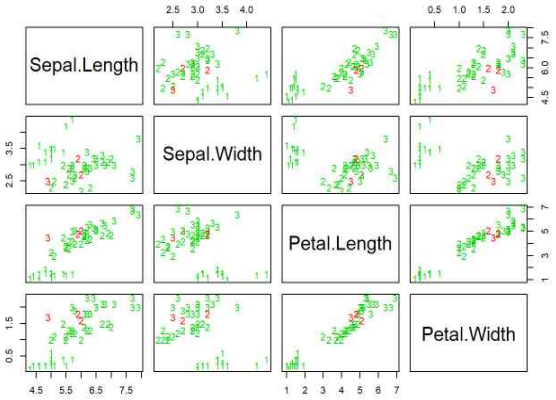
예제3 : {kknn}kknn()함수를 사용하여 k-인접이웃분류를 시행한다.
kknn()함수는 가중 k-NN분류를 제공한다. 검증용 자료의 각 행에 대해 k-인접이웃을 민코우스키 거리에 기반하여 구한다.
distance= 옵션은 민코우스키 거리의 모수(parameter)를 지정하며, distance=2는 유클리드 거리에 해당한다.
kernel= 옵션은 이웃점들의 가중치를 부여하는 방법을 지정하며, "rectangular"(가중을 고려하지 않은 k-NN과 동일), "triangular"
, "epanechnikov"(또는 beta(2,2)), "biweight"(또는 beta(3,3)), "triweight"(또는 beta(4,4)), "cos", "inv", "optimal"이 있다.
K-NN 분류는 커널밀도 함수의 합이 최대인 군집으로 분류를 수행한다.

데이터를 불러와서 러닝데이터와 유효데이터로 분류한다.
install.packages("kknn")
library(kknn)
data(iris)
m <- dim(iris)[1] # 데이터의 개수 저장
val <- sample(1:m, size=round(m/3), replace=FALSE, prob=rep(1/m, m))
iris.learn <- iris[-val,] # 러닝데이터에 150 * 2/3 = 100개 데이터 할당
iris.valid <- iris[val,] # 유효데이터에 150 * 1/3 = 50개 데이터 할당

kknn()함수를 사용하여 k-인접이웃분류를 시행한다.
iris.kknn <- kknn(Species~., iris.learn, iris.valid, distance=1, kernel="triangular") # k-인접이웃분류의 모수를 계산, 출력 없음
summary(iris.kknn) # summary를 통해 분류를 한 결과를 출력, 데이터마다 각각의 종류로 분류될 확률 알려줌
결과 : Response: "nominal"
fit prob.setosa prob.versicolor prob.virginica
1 versicolor 0 0.82767436 0.17232564
2 versicolor 0 1.00000000 0.00000000

새로운 데이터를 이용하여 만든 모형을 검증하는 과정
fit <- fitted(iris.kknn) # k-인접이웃분류의 모수를 적합한다.
table(iris.valid$Species, fit) # 유효데이터의 class와 적합한 결과를 table로 시각화
출력 : fit
setosa versicolor virginica
setosa 18 0 0
versicolor 0 13 1
virginica 0 1 17

교차 산점도로 표현
# 유효데이터의 class를 숫자 "1","2","3"으로 표현
pcol <- as.character(as.numeric(iris.valid$Species))
# 교차 산점도로 표현
pairs(iris.valid[1:4], pch=pcol, col=c("green3", "red")[(iris.valid$Species != fit)+1])



예제4 : {kknn}kknn()함수와 {FNN}get.knnx()함수를 사용하여 k-인접이웃분류를 시행
# {kknn}kknn()함수를 이용
library(kknn)

프로야구 선수 6명에 대해 두 시즌 간의 기록(lag1, lag2)이 다음 해의 득점(runs)에 미친 영향을 알아보기 위해 k-NN회귀 수행
full <- data.frame(name=c("McGwire,Mark", "Bonds,Barry", "Helton,Todd", "Walker,Larry", "Pujols,Albert", "Pedroia,Dustin"),
lag1=c(100,90,75,89,95,70), lag2=c(120,80,95,79,92,90), Runs=c(65,120,105,99,65,100))
full
결과
```
          name lag1 lag2 Runs
1    McGwire,Mark  100  120   65
2      Bonds,Barry   90   80  120
3      Helton,Todd   75   95  105
4    Walker,Larry   89   79   99
5   Pujols,Albert   95   92   65
6 Pedroia,Dustin   70   90  100
```
train <- full[full$name!="Bonds,Barry",] # 검증용데이터에 "Bonds,Barry"를 제외한 5명의 선수 데이터 할당
test <- full[full$name=="Bonds,Barry",] # 훈련용데이터에 1명, "Bonds,Barry" 선수 데이터 할당

{kknn}kknn()함수를 이용
# distance= 옵션은 민코우스키 거리의 모수(parameter)를 지정하며, distance=2는 유클리드 거리에 해당한다.
# 모형식 기반으로 분석을 진행
k <- kknn(Runs~lag1+lag2, train=train, test=test, k=2, distance=1) # k-인접이웃분류의 모수를 계산, 출력 없음
fit <- fitted(k) # k-인접이웃분류의 모수를 적합
fit
결과 [1] 90.5 -> "Bonds,Barry"는 다음년도에 90.5점 정도 얻을 것으로 예측된다.

예측된 값 분석
names(k) # k에 어떤 값들이 있는지 name()함수로 확인
k$fitted.values # 앞에서 확인한 fitted값이랑 같음
k$CL # k=2 이므로 "Bonds,Barry"와 가장 가까운 2개의 인접값(득점)으로 99, 65를 얻었다.
k$W # k=2 이므로 2개의 인접값에 대한 가중치인 0.75, 0.25를 구한다.
예측 점수는 예측 점수와 가중치를 사용한 가중평균을 계산하여 결과를 나타낸다. 식 : (99*3+65*1)/4 = 90.5
k$C # 65점이 여려명인 상태에서 인접값이 누구인지 확인하기 위해서 사용, "Bonds,Barry" 빠진 훈련용데이터의 3번째 4번째를 말한다.
train[c(k$C),] # 따라서 실제론 전체데이터의 4,5번째 데이터를 출력하는 것이다.
출력
```
# name lag1 lag2 Runs
# 4   Walker,Larry    89    79    99
# 5 Pujols,Albert    95    92    65
```

{FNN}get.knnx()함수를 사용하여 k-인접이웃분류를 시행
# {FNN}패키지는 훈련용자료에 대해 원하는 질의를 통해 필요한 결과를 얻게 해준다.
# {FNN}은 Fast Nearest Neighbor Search Algorithms and Applications을 의미한다.
# get.knnx(data= 훈련용 데이터, query= 검증용데이터, k= k의수)
install.packages("FNN")
library(FNN)
get.knnx(data=train[,c("lag1","lag2")], query=test[,c("lag1","lag2")], k=2) # 인접이웃의 훈련용데이터에서의 인덱스와 유클리드 거리가 출력되었다.
train[c(3,4), "name"] # 인접이웃의 이름을 출력한다. 출력 : [1] "Walker,Larry" "Pujols,Albert"

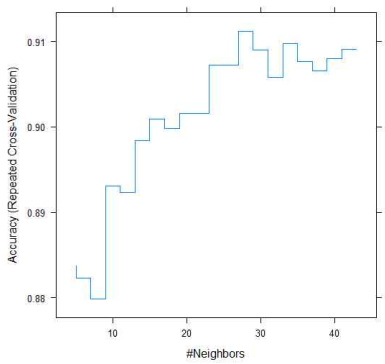
7.3 {caret}을 이용한 k-NN 분석
{caret}패키지을 이용하여 k-NN분석 수행

(a) 표본추출 : {caret}createDataPartition()
{caret}createDataPartition()함수를 사용하여 훈련용 데이터 셋(training set)과 검증용 데이터 셋(test set)으로 나눈다.
자료분할을 위해 이전에 사용되었던 방식보다 매우 편리하게 자료를 분할 함을 알 수 있다.
#(확인 필요) 복잡한 회귀와 분류 문제에 대한 모형 훈련과 조절 과정을 간소화하는 함수를 포함하고 있다.
#(확인 필요) resampling을 사용하여 모형의 모형의 조절모수가 성능에 미치는 영향을 평가한다.
install.packages("ISLR") # 데이터 셋을 위해서
library(ISLR)
install.packages("caret")
library(caret)
set.seed(100)
indxTrain <- createDataPartition(y = Smarket$Direction, p = 0.75, list = FALSE) # $Direction함수의 비율을 유지한채로 0.75 데이터 추출
training <- Smarket[indxTrain,] # 원자료의 0.75를 훈련용 데이터로 할당
testing <- Smarket[-indxTrain,] # 원자료의 0.25를 검증용 데이터로 할당
head(Smarket$Direction) # 출력 : [1] Up   Up   Down Up   Up   Up
prop.table(table(Smarket$Direction)) * 100 # 원자료의 업 다운 비율이 48.16 : 51.84 정도이다.
prop.table(table(training$Direction)) * 100 # train셋의 업 다운 비율이 48.18763 51.81237으로 원자료와 비슷하게 0.75비율로 뽑는다.
prop.table(table(testing$Direction)) * 100 # test셋의 업 다운 비율이 48.07692 51.92308으로 원자료와 비슷하게 0.25비율로 뽑는다.

(b) 전처리 : {caret}preProcess()
# {caret}preProcess()함수를 이용하여 전처리를 진행한다.
trainX <- training[,names(training) != "Direction"] # 반응변수를 제외한 것을 저장
preProcValues <- preProcess(x = trainX, method = c("center", "scale")) # 반응 변수를 제외한 데이터셋에 전처리(중심화와 척도화)를 한다.
preProcValues

(c) 훈련과 훈련 조율 : {caret}train()

```
library(e1071)
set.seed(200)
ctrl <- trainControl(method="repeatedcv", repeats = 3)
# train()함수의 옵션
# method = "knn" : knn방법을 이용, trControl : 3번 할거라고 표시, preProcess : 전처리(중심화와 척도화)가 필요하므로 선언
# tuneLength : 각 모수에 대해 값을 지정하는 대신 각 모수별로 고려해야 할 모수 값의 개수(길이)를 지정한다. 디폴트는 3개로 지정
knnFit <- train(Direction ~ ., data = training, method = "knn", trControl = ctrl, preProcess = c("center","scale"), tuneLength = 20)
# k의 적합결과로, Accuracy를 기준으로 k를 늘려나가면서 확인을 하여 기준값이 가장 큰 k를 선택한다.
# k가 27일때 Accuracy(정확도)가 가장 크므로 k가 27인 모델을 사용하도록 한다. kappa 값은 별다른 의미가 없으며 그냥 계산되는 값이다.
knnFit
# 출력
# ...
# 17   0.9008712   0.8008697
# 19   0.8998034   0.7987302
# 21   0.9015651   0.8021944
# 23   0.9015574   0.8021663
# 25   0.9072465   0.8135966
# 27   0.9072542   0.8135649
# 29   0.9111626   0.8214699
# 31   0.9090274   0.8171458
# 33   0.9058319   0.8107612
# ...
## Accuracy was used to select the optimal model using the largest value.
# The final value used for the model was k = 29.
```



인접이웃 크기에 따라서 교차 타당법에 기초하여 정확도를 구하면 다음과 같다.

```
plot(knnFit) # k의 적합결과를 꺾은선 그래프로 표현한다.
```

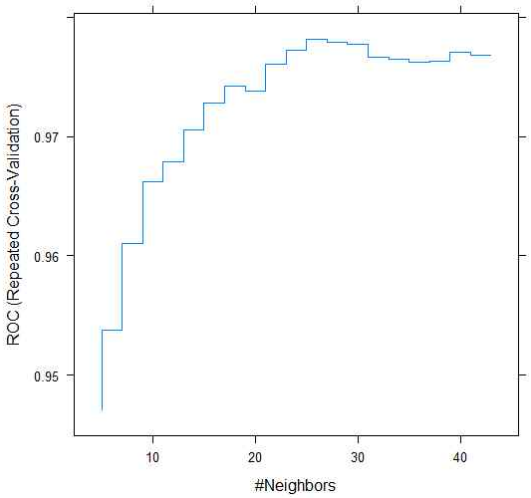
적합된 모형을 이용하여 검증용 자료에 대해 예측을 수행

```
knnPredict <- predict(knnFit, newdata = testing ) # 검증용 자료에 대한 예측, 결과 : [1] Up   Up   Down Down Up
confusionMatrix(knnPredict, testing$Direction ) # 예측한 값들(prediction)이 22+8=30개가 잘못 예측되었다.
# 출력 : Reference
# Prediction Down  Up
# Down   128   8
# Up      22 154
# Accuracy : 0.9038 -> 정분류율로 (128+154)/(128+8+22+154) = 0.9038이고, 검증용 자료에 대한 정확도가 90.38%라고 할수 있다.
# 95% CI : (0.8656, 0.9342)
# No Information Rate : 0.5192  -> 자료에서 1(Positive Class)과 0의 비율 중 큰 값으로 (8+154)/312 = 0.5192 이다.
# Kappa : 0.8067  -> (Accuracy-기대정확도)/(1-기대정확도)=0.8067, 기대정확도=( (128+22)*(128+8)/312+(22+154)*(8+154)/312 )/312 = 0.5025
#             Sensitivity : 0.8533 -> 민감도로 128/(128+22) = 0.8533
#             Specificity : 0.9506 -> 특이도로 154/(8+154) = 0.9506
#          Pos Pred Value : 0.9412 -> 예측 1 가운데 바르게 예측한(TP의) 비율로 128/(128+8) = 0.9412
#          Neg Pred Value : 0.8750 -> 예측 0 가운데 바르게 예측한(TN의) 비율로 154/(22+154) = 0.8750
#              Prevalence : 0.4808 -> 자료에서 더 큰 범주(1)의 비율 (128+22)/(128+8+22+154) = 0.4808
#          Detection Rate : 0.4103 -> 전체에서 TP의 비율로 128/(128+8+22+154) = 0.4103
#    Detection Prevalence : 0.4359 -> 전체에서 더 큰 범주(1)로 예측할 비율로 (128+8)/(128+8+22+154) = 0.4359
#       Balanced Accuracy : 0.9020 -> (민감도 + 특이도) / 2 또는 (TP/P + TN/N)/2으로 (128/(128+22)+154/(8+154))/2 = 0.90195
```

검증용 자료에 대한 정확도로 Accuracy의 값을 나타낸다.

```
mean(knnPredict == testing$Direction)
# 결과 : [1] 0.8910256
```

{caret}train()함수의 trainControl() 옵션 : summaryFunction = twoClassSummary 으로 성능 측도(ROC, 민감도 측이도) 표현
# summaryFunction = twoClassSummary를 지정하면 디폴트 측도(Accuracy, Kappa)가 이닌 성능 측도(ROC, 민감도 측이도)를 제공해준다.
set.seed(200)
ctrl <- trainControl(method="repeatedcv", repeats = 3, classProbs=TRUE, summaryFunction = twoClassSummary)
knnFit <- train(Direction ~ ., data = training, method = "knn", trControl = ctrl, preProcess = c("center","scale"), tuneLength = 20)
knnFit
아까와 처음 부분은 같게(비슷하게) 출력될 것이지만
성능 측도를 바꿔서 출력이 됬으므로 정확도 카파 값이 아닌 ROC, Sens, Spac 측도로 계산이 되었다.
앞에서는 Accuracy를 기준으로 k=25일때 가장 컸지만, ROC 측도를 기준으로 비교했을때 k=43일때 값이 가장 크므로 k=43으로 적합. 하지만
k=25일때의 43일때의 값과 거의 비슷하고, 계산을 43개보단 25개 하는 것이 계산을 덜할 수 있으므로 K=25가 좋은 대안이라고 할 수 있다.
정분류 관점에서 25일때보다 43일때 오분류가 적으므로 43이 더 적합하다.
결과
...
k   ROC   Sens   Spec
...
19 0.9742213 0.8533011 0.9431831
21 0.9738209 0.8518035 0.9479308
23 0.9760444 0.8532689 0.9465136
25 0.9772831 0.8606280 0.9506378
27 0.9781590 0.8576973 0.9533588
29 0.9779005 0.8628986 0.9560941
31 0.9777936 0.8591787 0.9553997
...
#
ROC was used to select the optimal model using the largest value.
The final value used for the model was k = 27.



이웃의 수에 대한 정확도 그림(반복된 교차타당법에 의한)
plot(knnFit, print.thres = 0.5, type="S")

predict()함수를 이용하여 적합된 모형을 가지고 검증용 자료에 대해 예측을 수행
knnPredict <- predict(knnFit, newdata = testing)
confusionMatrix(knnPredict, testing$Direction) # Accuracy : 0.9103
결과 : Confusion Matrix and Statistics
#
Reference
Prediction Down Up
Down 122   7
Up   28 155
#
Accuracy : 0.8878
95% CI : (0.8474, 0.9206)
No Information Rate : 0.5192
P-Value [Acc > NIR] : < 2.2e-16
#
Kappa : 0.7741
#
Mcnemar's Test P-Value : 0.0007232
#
Sensitivity : 0.8133
Specificity : 0.9568
Pos Pred Value : 0.9457
Neg Pred Value : 0.8470
Prevalence : 0.4808
Detection Rate : 0.3910
Detection Prevalence : 0.4135
Balanced Accuracy : 0.8851

검증용 자료에 대한 정확도로 Accuracy의 값을 나타낸다.
# 정분류율 관점에서 모형의 성능이 다소 향상되었음을 확인할 수 있다.(0.9038->0.9103)
mean(knnPredict == testing$Direction)
# 결과 : [1] 0.8878205

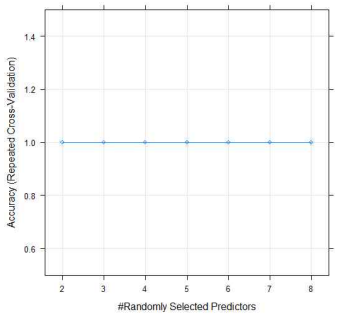
```
## ROC 곡선
# AUC 기준을 사용하여 모형의 성능을 파악한다.
# 기준값을 0.5를 기준으로 그리는 것이다. (민감도, 특이도)
library(pROC)
knnPredict <- predict(knnFit, newdata = testing , type="prob") # 적합한 모형을 가지고 새로운 데이터에 대해 예측을 수행
head(knnPredict) # 예측된 데이터
knnROC <- roc(testing$Direction, knnPredict[,"Down"], levels = levels(testing$Direction)) # ROC 그림을 그리기 위해 데이터를 변환
knnROC
# 결과
# Data: knnPredict[, "Down"] in 150 controls (testing$Direction Down) > 162 cases (testing$Direction Up).
# Area under the curve: 0.9698 (AUC 값)

## print.thres(민감도, 특이도) 이 기준점으로 그래프에 표시된다.
plot(knnROC, type="S", print.thres= 0.5)
```

```
### (d) 랜덤포리스트를 적용 {caret}train() method="rf"
# 랜덤포리스트 방법을 적용하여 모형을 구축하고, 앞서 다룬 k-NN방법과의 성능을 비교한다.
# 램덤포리스트는 일종의 앙상블 모형으로, 대체로 성능이 뛰어난 방법으로 알려져 있다.
# 랜덤포리스트는 10장에서 자세하게 다룰 예정이다.

## {caret}train()함수의 옵션 method = "rf"를 통해서 랜덤 포래스트를 사용
# method = "rf" : 랜덤포리스트(rf) 이용
# trControl : 3번 할거라고 표시
# preProcess : 전처리(중심화와 척도화)가 필요하므로 선언
# tuneLength : 각 모수에 대해 값을 지정하는 대신 각 모수별로 고려해야 할 모수 값의 개수(길이)를 지정한다. 디폴트는 3개로 지정
library(e1071)
set.seed(300)
ctrl <- trainControl(method="repeatedcv", repeats = 3)
rfFit <- train(Direction ~ ., data = training, method = "rf", trControl = ctrl, preProcess = c("center","scale"), tuneLength = 20)
rfFit
# 결과 Random Forest
#
# 938 samples
# 8 predictor
# 2 classes: 'Down', 'Up'
#
# Pre-processing: centered (8), scaled (8)
# Resampling: Cross-Validated (10 fold, repeated 3 times)
# Summary of sample sizes: 844, 844, 844, 845, 844, 845, ...
# Resampling results across tuning parameters:
#
#  mtry  Accuracy   Kappa
# 2     0.9989323  0.9978609
# 3     0.9989323  0.9978609
# 4     0.9989323  0.9978609
# 5     0.9989323  0.9978609
# 6     0.9989323  0.9978609
# 7     0.9989323  0.9978609
# 8     0.9989323  0.9978609
#
# Accuracy was used to select the optimal model using the largest value.
# The final value used for the model was mtry = 2.
```

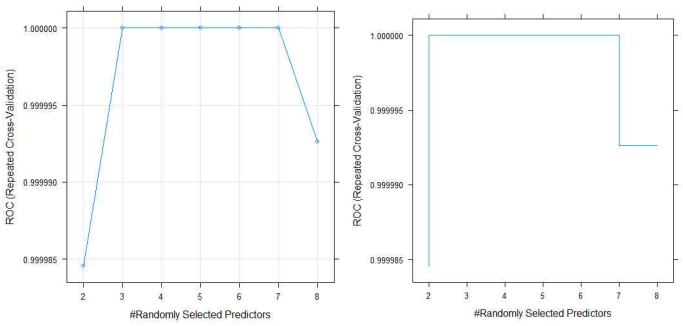
mtry에 대한 Accuracy값을 시각화로 표현
# 변화가 없이 일정하므로 k=2를 선택
plot(rfFit)



적합된 모형을 이용하여 검증용 자료에 대해 예측을 수행
# 랜덤포리스트 모형이 검증용 자료를 거의 완벽하게 분류함을 확인할 수 있다.(Accuracy : 0.9968)
rfPredict <- predict(rfFit, newdata = testing)
confusionMatrix(rfPredict, testing$Direction)
# 결과 :
# Reference
# Prediction Down Up
# Down 150 1
# Up 0 161
# ...
# Accuracy : 0.9968
# ...
# Kappa : 0.9936
# ...

검증용 자료에 대한 정확도로 Accuracy의 값을 나타낸다.
# 정확도 기준에서 K-NN 방법보다 랜덤포리스트 방법이 더 잘 분류함을 알 수 있다.
mean(rfPredict == testing$Direction)
# 결과 : [1] 0.9967949

{caret}train() -> trainControl() 옵션 : summaryFunction = twoClassSummary
# summaryFunction = twoClassSummary를 지정하면 디폴트 측도(Accuracy, Kappa)가 이닌 성능 측도(ROC, 민감도 측이도)를 제공해준다.
# 랜덤 포리스트 방법이므로 method = "rf"를 사용한다.
set.seed(300)
ctrl <- trainControl(method="repeatedcv", repeats = 3, classProbs=TRUE, summaryFunction=twoClassSummary)
rfFit <- train(Direction ~ ., data = training, method = "rf", trControl = ctrl, preProcess = c("center","scale"), tuneLength = 20)
# ROC기준으로 했을때 k=3인 경우 1이 되므로 k=3으로 적합하는것이 좋다고 할 수 있다.
rfFit
결과
Random Forest
#
938 samples
8 predictor
2 classes: 'Down', 'Up'
#
Pre-processing: centered (8), scaled (8)
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 844, 844, 844, 845, 844, 845, ...
Resampling results across tuning parameters:
#
mtry ROC Sens Spec
2 0.9999846 0.9977939 1
3 1.0000000 0.9977939 1
4 1.0000000 0.9977939 1
5 1.0000000 0.9977939 1
6 1.0000000 0.9977939 1
7 1.0000000 0.9977939 1
8 0.9999926 0.9977939 1
ROC was used to select the optimal model using the largest value.
The final value used for the model was mtry = 3.



mtry에 대한 Accuracy값을 시각화로 표현
# k=2일때 ROC가 가장 높으므로 k=3을 선택를 선택
plot(rfFit)
# 다른 형식으로 그래프 작성
plot(rfFit, print.thres = 0.5, type="S")

predict()함수를 이용하여 적합된 모형을 가지고 검증용 자료에 대해 예측을 수행
```
rfPredict <- predict(rfFit, newdata = testing )
confusionMatrix(rfPredict, testing$Direction )
# 결과
# Confusion Matrix and Statistics
# ...
# Prediction Down  Up
# Down  150   1
# Up      0 161
#
# Accuracy : 0.9968
# ...
# Kappa : 0.9936
# ...
#              Sensitivity : 1.0000
#              Specificity : 0.9938
```

검증용 자료에 대한 정확도로 Accuracy의 값을 나타낸다.
```
# 이 자료에 경우 랜덤포리스트 방법이 더 잘 분류함을 알 수 있다.
mean(rfPredict == testing$Direction)
# 결과 : [1] 0.9967949
```

ROC 곡선
```
# AUC 기준을 사용하여 모형의 성능을 파악한다.
# K-NN 모형의 경우 AUC=0.9698, 랜덤포리스트 모형의 경우 AUC=1
# AUC 기준에 의해, 구축된 랜덤포리스트 모형의 성능이 매우 우수함을 알 수 있다.(정확도 기준에 의한 결과와 동일)
library(pROC)
rfPredict <- predict(rfFit, newdata = testing , type="prob")
rfROC <- roc(testing$Direction,rfPredict[,"Down"], levels = rev(testing$Direction))
rfROC
# 기준값을 0.5를 기준으로 그리는 것이다. (민감도, 특이도)
plot(rfROC, type="S", print.thres= 0.5)
```

```
> # ROC 곡선 그리기
> library(pROC)
> rfPredict <- predict(rfFit,newdata = testing , type="prob")
> rfROC <- roc(testing$Direction,rfPredict[,"Down"],
               levels = rev(testing$Direction))
> rfROC

 Call:
 roc.default(response = testing$Direction, predictor = rfPredict[,
"Down"], levels =   levels(testing$Direction))

 Data: rfPredict[, "Down"] in 150 controls (testing$Direction
Down) > 162 cases  (testing$Direction Up).
 Area under the curve: 1
```

```
> plot(rfROC, type="S", print.thres= 0.5)
```