KNN

Park Ju ho

2022 4 17

class의 knn함수를 이용한 knn

모형식 기반이 아니기에 그냥 test,train을 넣음

```
library(class)

data(iris3)#data shape = 50*4*3 = number of data,feature,label

head(iris3)
```

```
## , , Setosa
##
##      Sepal L. Sepal W. Petal L. Petal W.
## [1,]      5.1      3.5      1.4      0.2
## [2,]      4.9      3.0      1.4      0.2
## [3,]      4.7      3.2      1.3      0.2
## [4,]      4.6      3.1      1.5      0.2
## [5,]      5.0      3.6      1.4      0.2
## [6,]      5.4      3.9      1.7      0.4
##
## , , Versicolor
##
##      Sepal L. Sepal W. Petal L. Petal W.
## [1,]      7.0      3.2      4.7      1.4
## [2,]      6.4      3.2      4.5      1.5
## [3,]      6.9      3.1      4.9      1.5
## [4,]      5.5      2.3      4.0      1.3
## [5,]      6.5      2.8      4.6      1.5
## [6,]      5.7      2.8      4.5      1.3
##
## , , Virginica
##
##      Sepal L. Sepal W. Petal L. Petal W.
## [1,]      6.3      3.3      6.0      2.5
## [2,]      5.8      2.7      5.1      1.9
## [3,]      7.1      3.0      5.9      2.1
## [4,]      6.3      2.9      5.6      1.8
## [5,]      6.5      3.0      5.8      2.2
## [6,]      7.6      3.0      6.6      2.1
```

```
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)))#setosa,versicolor,virginica

knn(train,test,cl,k=3,prob=T)
```

```
##  [1] s s s s s s s s s s s s s s s s s s s s s s s s s c c v c c c c c v c c c c
## [39] c c c c c c c c c c c c v c c v v v v v v v v v v c v v v v v v v v v v v v
## attr(,"prob")
##  [1] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
##  [8] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [15] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [22] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 0.6666667
## [29] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 0.6666667 1.0000000
## [36] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [43] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [50] 1.0000000 1.0000000 0.6666667 0.7500000 1.0000000 1.0000000 1.0000000
## [57] 1.0000000 1.0000000 0.5000000 1.0000000 1.0000000 1.0000000 1.0000000
## [64] 0.6666667 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [71] 1.0000000 0.6666667 1.0000000 1.0000000 0.6666667
## Levels: c s v
```

k=3일 때 test data를 분류한 결과
prob는 각 label이 분류된 그 확률을 의미
=> 3개 중 가장 많은 label의 비율
ex)K개 안에 setosa 2개 virgicolor 1개 => 0.6667의 확률로 setosa

DMwR의 knn함수

모형식 기반으로 수행, 정규화 옵션을 제공해 줌

```
library(DMwR)

data(iris)
idxs = sample(1:nrow(iris),as.integer(0.7*nrow(iris)))#Train = 150 * 0.7, test = 150*0.3
trainIris = iris[idxs,]
testIris = iris[-idxs,]

nn3 = kNN(Species~., trainIris,testIris,norm = F,k=3)

table(testIris$Species,nn3)
```

```
##             nn3
##              setosa versicolor virginica
##   setosa         19          0         0
##   versicolor      0         10         1
##   virginica       0          1        14
```

```
nn3_n = kNN(Species~., trainIris,testIris,norm = T,k=3)

table(testIris$Species,nn3_n)
```

```
##             nn3_n
##              setosa versicolor virginica
##   setosa         19          0         0
##   versicolor      0         10         1
##   virginica       0          2        13
```

```
nn5 = kNN(Species~., trainIris,testIris,norm = F,k=5)

table(testIris$Species,nn5)
```

```
##             nn5
##              setosa versicolor virginica
##   setosa         19          0         0
##   versicolor      0         10         1
##   virginica       0          0        15
```

k=3이고 정규화는 하지 않은 것, k=3이고 정규화를 진행한 것, k=5인 것 3가지를 비교

kknn을 활용한 knn

kernel을 활용 할 수 있는 함수

```
library(kknn)

data(iris)

m = dim(iris)[1]
m
```

```
## [1] 150
```

```
val = sample(1:m, size = round(m/3), replace = F, prob = rep(1/m,m))
iris.learn = iris[-val,]
iris.valid = iris[val,]

iris.kknn = kknn(Species~.,iris.learn,iris.valid,distance=1,kernel="triangular")
summary(iris.kknn)
```

```
##
## Call:
## kknn(formula = Species ~ ., train = iris.learn, test = iris.valid,     distance = 1, kernel = "trian
##
## Response: "nominal"
##           fit prob.setosa prob.versicolor prob.virginica
## 1      setosa           1       0.0000000     0.00000000
```

```
## 2      setosa          1       0.0000000     0.00000000
## 3   virginica          0       0.0000000     1.00000000
## 4  versicolor          0       1.0000000     0.00000000
## 5   virginica          0       0.1271368     0.87286320
## 6   virginica          0       0.0000000     1.00000000
## 7   virginica          0       0.4508332     0.54916677
## 8      setosa          1       0.0000000     0.00000000
## 9   virginica          0       0.2007677     0.79923227
## 10  virginica          0       0.0000000     1.00000000
## 11 versicolor          0       1.0000000     0.00000000
## 12 versicolor          0       1.0000000     0.00000000
## 13  virginica          0       0.1079719     0.89202812
## 14     setosa          1       0.0000000     0.00000000
## 15 versicolor          0       0.6628696     0.33713043
## 16 versicolor          0       1.0000000     0.00000000
## 17     setosa          1       0.0000000     0.00000000
## 18  virginica          0       0.1180279     0.88197209
## 19 versicolor          0       0.8064634     0.19353657
## 20     setosa          1       0.0000000     0.00000000
## 21 versicolor          0       1.0000000     0.00000000
## 22 versicolor          0       1.0000000     0.00000000
## 23  virginica          0       0.0000000     1.00000000
## 24     setosa          1       0.0000000     0.00000000
## 25     setosa          1       0.0000000     0.00000000
## 26 versicolor          0       1.0000000     0.00000000
## 27     setosa          1       0.0000000     0.00000000
## 28     setosa          1       0.0000000     0.00000000
## 29     setosa          1       0.0000000     0.00000000
## 30     setosa          1       0.0000000     0.00000000
## 31  virginica          0       0.4703741     0.52962591
## 32 versicolor          0       1.0000000     0.00000000
## 33  virginica          0       0.0000000     1.00000000
## 34     setosa          1       0.0000000     0.00000000
## 35     setosa          1       0.0000000     0.00000000
## 36     setosa          1       0.0000000     0.00000000
## 37 versicolor          0       1.0000000     0.00000000
## 38 versicolor          0       0.9726431     0.02735686
## 39     setosa          1       0.0000000     0.00000000
## 40 versicolor          0       1.0000000     0.00000000
## 41  virginica          0       0.0000000     1.00000000
## 42 versicolor          0       0.5880777     0.41192232
## 43     setosa          1       0.0000000     0.00000000
## 44  virginica          0       0.0000000     1.00000000
## 45     setosa          1       0.0000000     0.00000000
## 46     setosa          1       0.0000000     0.00000000
## 47  virginica          0       0.0000000     1.00000000
## 48  virginica          0       0.0000000     1.00000000
## 49     setosa          1       0.0000000     0.00000000
## 50 versicolor          0       1.0000000     0.00000000
```

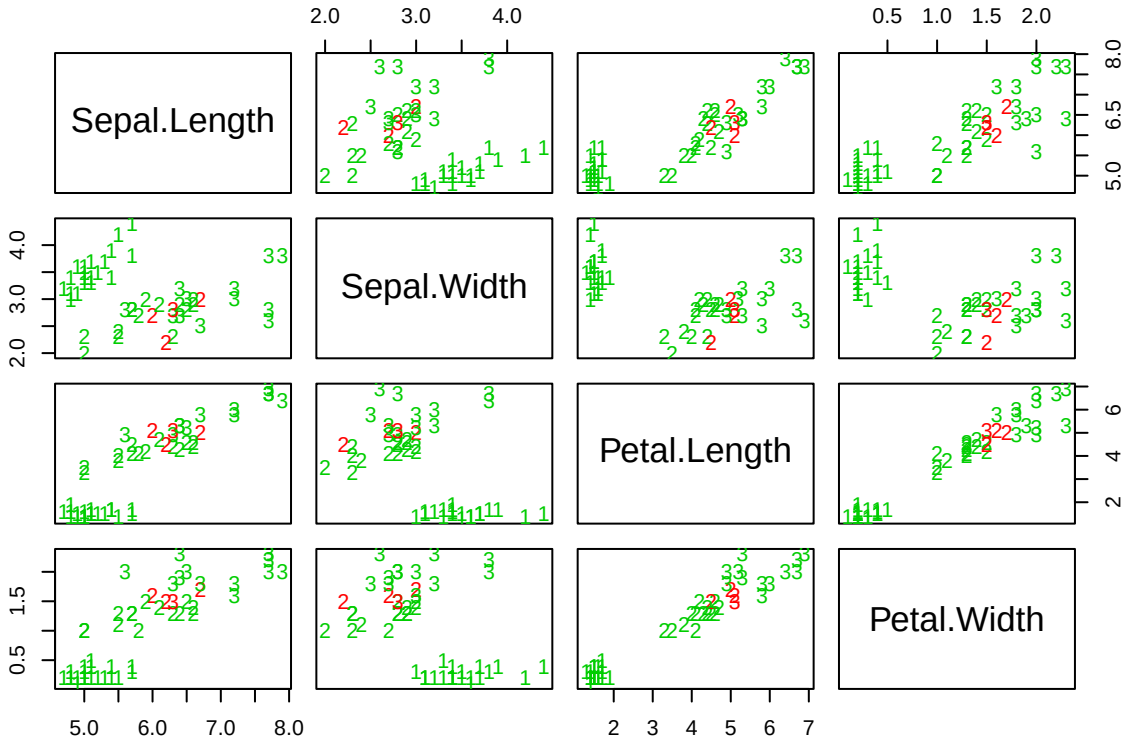
```
fit = fitted(iris.kknn)
table(iris.valid$Species,fit)
```

```
##             fit
```

```
##              setosa versicolor virginica
##    setosa         20          0         0
##    versicolor      0         14         3
##    virginica       0          1        12
```

triangular 커널을 적용하고 유클리드가 아닌 m=1거리를 적용시킨 knn

```
pcol = as.character(as.numeric(iris.valid$Species))
pairs(iris.valid[1:4],pch=pcol,col=c("green3","red")[(iris.valid$Species!=fit)+1])
```



틀린 데이터만 빨간색으로 하여 그래프를 그림
Sepal.Length와 Petal.Length를 보면 2,3이 구분되는 경계에 오분류가 많이 발생함
=> 분류에 사용시 조금 조심할 필요가 있음

k-NN회귀

```
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
```

```
##             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$name!="Bonds,Barry",]
test = full[full$name=="Bonds,Barry",]

k = kknn(Runs~lag1+lag2,trai = train,test = test, k=2, distance=1)
fit = fitted(k)
fit
```

```
## [1] 90.5
```

Bonds,Barry의 예측값으로 90.5를 예측

```
names(k)
```

```
##  [1] "fitted.values" "CL"            "W"             "D"
##  [5] "C"             "prob"          "response"      "distance"
##  [9] "call"          "terms"
```

```
k$fitted.values
```

```
## [1] 90.5
```

```
k$CL
```

```
##      [,1] [,2]
## [1,]   99   65
```

```
k$W
```

```
##      [,1] [,2]
## [1,] 0.75 0.25
```

Bonds의 인접값으론 99와 65가 나왔으며 이 둘의 가중치는 각각 0.75,0.25이다
Predcit = (99 * 0.75+65 * 0.25)/2 = 90.5

```
k$C
```

```
## [1] 3 4
```

```
train[c(k$C),]
```

```
##            name lag1 lag2 Runs
## 4  Walker,Larry   89   79   99
## 5 Pujols,Albert   95   92   65
```

인접한 두 선수론 3번과 4번 인덱스인 선수이다
이 둘은 Larry와 Alber이다

FNN을 활용한 KNN

Query를 활용하여 Knn이 가능함

```
library(FNN)

get.knnx(data = train[,c("lag1","lag2")], query = test[,c("lag1","lag2")],k=2)
```

```
## $nn.index
##      [,1] [,2]
## [1,]    3    4
##
## $nn.dist
##          [,1] [,2]
## [1,] 1.414214   13
```

train data의 lag1과 lag2를 이용하여 test data의 예측값을 구하는 것
위의 결과와 같이 인접한 이웃은 같으며 그 거리는 1.41,13을 거리를 가진다

caret data를 활용한 k-NN 분석

```
library(ISLR)
library(caret)

set.seed(100)
indxTrain = createDataPartition(y = Smarket$Direction, p = 0.75, list = F)#train_test_split
training = Smarket[indxTrain,]
testing = Smarket[-indxTrain,]

prop.table(table(training$Direction))*100
```

```
##
##     Down       Up
## 48.18763 51.81237
```

```
prop.table(table(testing$Direction))*100
```

```
##
##     Down       Up
## 48.07692 51.92308
```

```
prop.table(table(Smarket$Direction))*100
```

```
##
##  Down    Up
## 48.16 51.84
```

```r
#normalization
trainX = training[,names(training)!="Direction"]
preProcValues = preProcess(x = trainX,method = c("center","scale"))
preProcValues
```

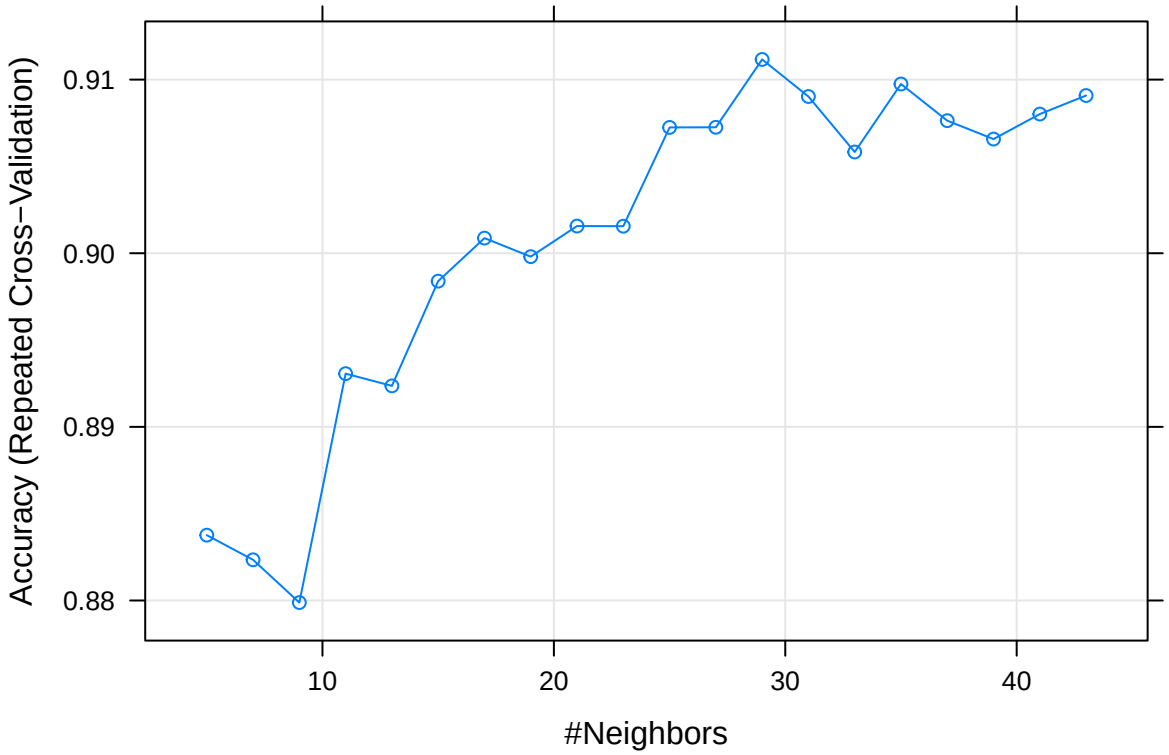
```
## Created from 938 samples and 8 variables
##
## Pre-processing:
##   - centered (8)
##   - ignored (0)
##   - scaled (8)
```

```r
set.seed(200)
ctrl = trainControl(method = "repeatedcv",repeats = 3)#cross Validate
knnFit = train(Direction ~ ., data = training, method = "knn", trControl=ctrl, preProcess = c("center",

knnFit
```

```
## k-Nearest Neighbors
##
## 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, 843, 844, 845, 845, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    5  0.8837625  0.7668667
##    7  0.8823439  0.7641736
##    9  0.8798806  0.7591151
##   11  0.8930660  0.7855027
##   13  0.8923603  0.7840046
##   15  0.8983926  0.7959911
##   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
##   35  0.9097479  0.8186335
##   37  0.9076353  0.8143704
##   39  0.9065752  0.8122363
##   41  0.9080164  0.8150995
##   43  0.9090842  0.8172527
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 29.
```

10 fold validate를 3번 반복하여 각 K별 정확도를 측정한 결과 K가 43일 때 정확도가 가장 높다

```
plot(knnFit)
```



K의 수가 증가할수록 정확도가 증가하고 있고 그 값이 43일 때 가장 높은 것을 그래프로 확인 할 수 있다

```
knnPredict = predict(knnFit, newdata = testing)
confusionMatrix(knnPredict,testing$Direction)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Down  Up
##       Down  124   8
##       Up     26 154
##
##                Accuracy : 0.891
##                  95% CI : (0.8511, 0.9233)
##     No Information Rate : 0.5192
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7808
##
##  Mcnemar's Test P-Value : 0.003551
##
##             Sensitivity : 0.8267
```

```
##             Specificity : 0.9506
##          Pos Pred Value : 0.9394
##          Neg Pred Value : 0.8556
##              Prevalence : 0.4808
##          Detection Rate : 0.3974
##    Detection Prevalence : 0.4231
##       Balanced Accuracy : 0.8886
##
##        'Positive' Class : Down
##
```

검증용 자료에 대한 정확도가 95%
=> 상당히 잘 예측하는 것을 확인 할 수 있음 => 과적합 역시 발생하지 않음

Accuracy = 정분류율 95% CI= 정분류율에 대한 신뢰구간 No Information Rate = 최빈값의 비율

Kappa와 Mcnemar's = 일치도를 나타내는 계수 Mcnemar는 0.2단위로 구간을 나누며 값이 클 수록 일치한다는 의미임

예측값(Prediction) \ 실제값(Reference)	Y	N
Y	True Positive(TP)	False Positive(FP)
N	False Negative(FN)	True Negative(TN)

지표	계산식	의미
Precision	$\frac{TP}{TP+FP}$	Y로 예측된 것 중 실제로도 Y인 경우의 비율
Accuracy	$\frac{TP+TN}{TP+FP+FN+TN}$	전체 예측에서(예측이 Y이든 N이든 무관하게) 옳은 예측의 비율
Recall (Sensitivity, TP Rate, Hit Rate)	$\frac{TP}{TP+FN}$	실제로 Y인 것들 중 예측이 Y로 된 경우의 비율
Specificity	$\frac{TN}{FP+TN}$	실제로 N인 것들 중 예측이 N으로 된 경우의 비율
FP Rate (False Alarm Rate)	$\frac{FP}{FP+TN}$	실제로 N인데 Y로 예측된 비율, 1-Specificity와 같은 값
Pos Pred Value	$\frac{TP}{TP+FP}$	Y로 예측한 것들 중에 실제 Y인 비율
Neg Pred Value	$\frac{TN}{TN+FN}$	N로 예측한 것들 중에 실제 N인 비율
Prevalence	$\frac{FN+TP}{TP+FP+FN+TN}$	전체 중 실제 Y가 나타난 비율
Detection Rate	$\frac{TP}{TP+FP+FN+TN}$	전체 중 모델이 실제 Y를 예측한 비율
Detection Prevalence	$\frac{TP+FP}{TP+FP+FN+TN}$	전체 중 모델이 Y로 예측한 비율
Balanced Accuracy	(Recall+Specificity)/2	(Recall+Specificity)/2
F-1 Score	$2\times\frac{1}{\frac{1}{Precision}+\frac{1}{recall}}$	Precision과 Recall의 조화평균 (Precision과 Recall 중 한쪽만 클 때보다 두 값이 골고루 클 때 큰 값을 가짐. 0~1 사이의 값을 가짐)
Kappa	$\frac{Accuracy-P(e)}{1-P(e)}$	일반적으로 코헨의 카파를 말하며 두 평가자의 평가가 얼마나 일치하는 지를 의미함. 0~1사이의 값을 가짐. P(e)는 두 평가자가 우연히 일치할 확률을 의미함.

Confusion Matrix와 관련하여 분류의 성능 척도에 쓰이는 지표들

AUC,민감도,특이도 정보 summaryFunction = twoClassSummary 이것을 추가함으로써 위의 정보를 제공

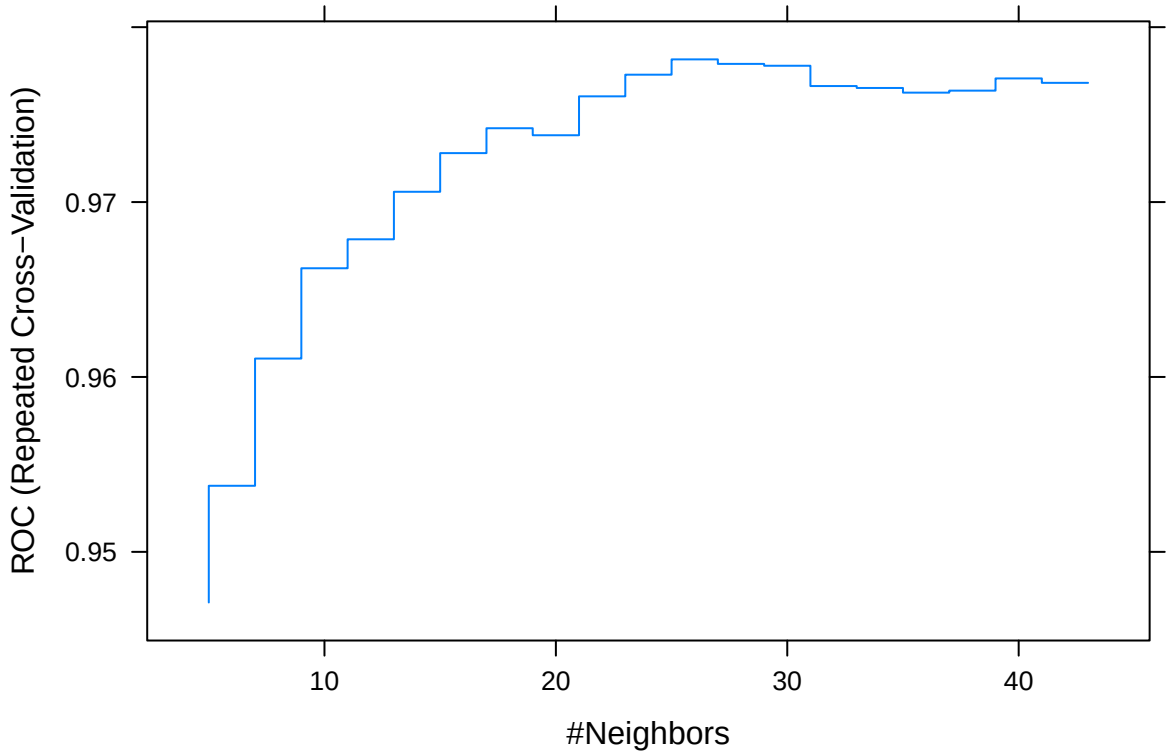
```
set.seed(200)
ctrl = trainControl(method = "repeatedcv",repeats = 3,classProbs = T,summaryFunction = twoClassSummary)
knnFit = train(Direction ~ ., data = training, method = "knn", trControl=ctrl, preProcess = c("center",
knnFit
```

```
## k-Nearest Neighbors
##
## 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, 843, 844, 845, 845, ...
## Resampling results across tuning parameters:
##
##   k   ROC        Sens       Spec
##    5  0.9471049  0.8584219  0.9073838
##    7  0.9537782  0.8635105  0.8998866
##    9  0.9610520  0.8540258  0.9040533
##   11  0.9662153  0.8672303  0.9171627
##   13  0.9678706  0.8613849  0.9212443
##   15  0.9705875  0.8599034  0.9342687
##   17  0.9727988  0.8569726  0.9418084
##   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
##   33  0.9766333  0.8569887  0.9512897
##   35  0.9765238  0.8621739  0.9540108
##   37  0.9762576  0.8599356  0.9519841
##   39  0.9763709  0.8584863  0.9513039
##   41  0.9770701  0.8570692  0.9554138
##   43  0.9768160  0.8606924  0.9540249
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was k = 27.
```

Accuracy 대신 ROC로 모델을 평가하는 것을 확인 할 수 있다

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



ROC그래프를 확인 할 수 있다

```
knnPredict = predict(knnFit, newdata = testing)
confusionMatrix(knnPredict,testing$Direction)
```

```
## 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
##
##        'Positive' Class : Down
##
```

```
mean(knnPredict == testing$Direction)
```

```
## [1] 0.8878205
```

정확도를 구하는 방법 중 하나

ROC Curve 그리기

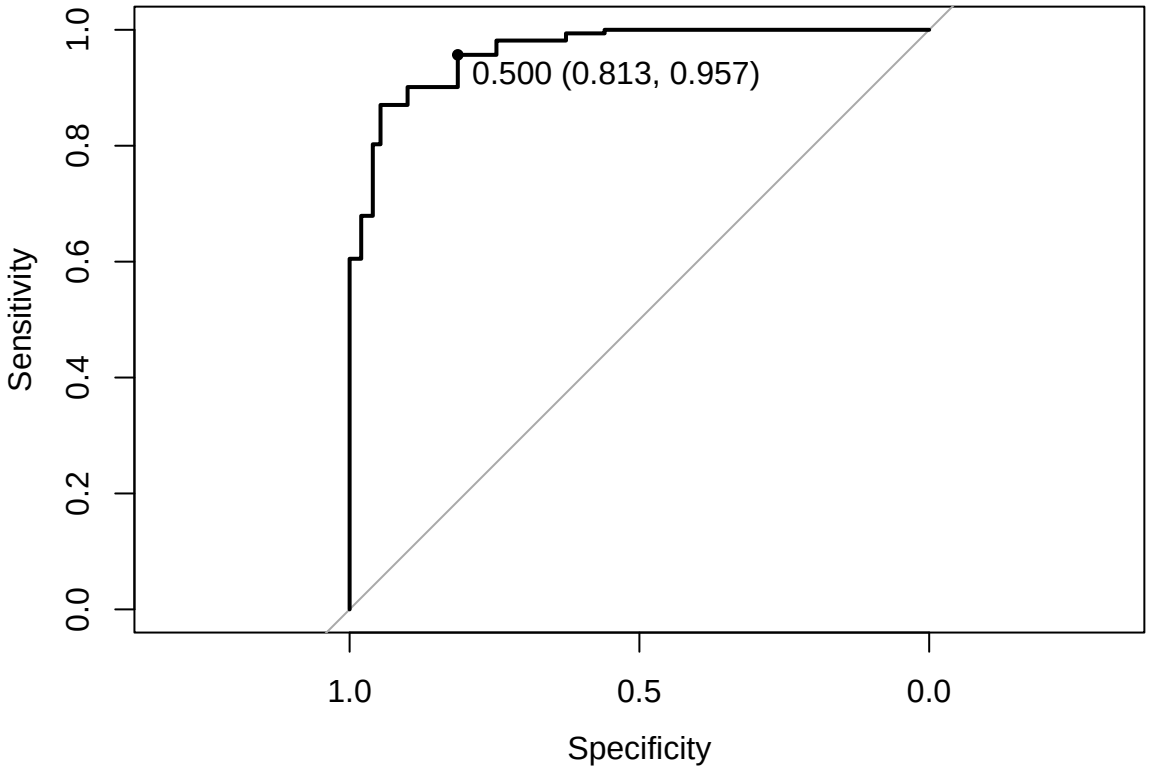
```
library(pROC)

knnPredict = predict(knnFit, newdata = testing, type = "prob")
head(knnPredict)
```

```
##         Down        Up
## 1 0.37037037 0.6296296
## 2 0.03703704 0.9629630
## 3 0.14814815 0.8518519
## 4 0.37037037 0.6296296
## 5 0.40740741 0.5925926
## 6 0.74074074 0.2592593
```

```
knnROC = roc(testing$Direction, knnPredict[,"Down"],levels = levels(testing$Direction))
knnROC
```

```
##
## Call:
## roc.default(response = testing$Direction, predictor = knnPredict[,     "Down"], levels = levels(testi
##
## Data: knnPredict[, "Down"] in 150 controls (testing$Direction Down) > 162 cases (testing$Direction Up
## Area under the curve: 0.9698
```

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



거의 활처럼 휘어있는 모습을 볼 수 있다
=> 매우 좋은 모델이라는 것을 확인 할 수 있음