**Homework 4**

**21500131 김은총**

**Decision Tree**

**Question 1. decision tree를 사용해서 income 변수를 가장 잘 예측하는 예측 모델을 만드시오. 어떻게 그 모델을 찾게 되었는지 설명하시오. confusion matrix, accuracy, precision, recall을 사용해 만든 모델의 성능을 설명하시오.**

**답 :**

**모델 1과 모델2는 plotcp 함수를 사용해서 그래프를 그린 뒤 cp값을 조절한 모델이다.**

**모델 3 ~ 모델8은 minsplit을 조절한 모델이다.**

1. **예시\_모델 1번**

library(rpart)

library(rpart.plot)

new.train <- train

train\_model.train.full <- rpart(income~ . , data = new.train, method = "class", control = rpart.control(cp = 0))

plotcp(train\_model.train.full) #0.0014 = cp

##Prune the tree

train\_model\_pruned <- prune(train\_model.train.full, cp = 0.0014)

# train set

new.train$pred <- predict(train\_model.train.full, new.train, type = "class")

#table

table(new.train$income , new.train$pred)

mean(new.train$income == new.train$pred) # 0.888

get\_precision(new.train$pred, new.train$income) # precision : 0.814

get\_recall(new.train$pred, new.train$income) # recall : 0.713

1. **예시\_모델 9번**

new.train9 <- train

train\_model9 <- rpart(income~ . , data = new.train9, method = "class", control= rpart.control(cp = 0, minsplit = 10))

# train set

new.train9$pred <- predict(train\_model9, new.train9, type = "class")

#table

table(new.train9$income , new.train9$pred)

mean(new.train9$income == new.train9$pred) # accuracy : 0.911

get\_precision(new.train9$pred, new.train9$income) # precision : 0.856

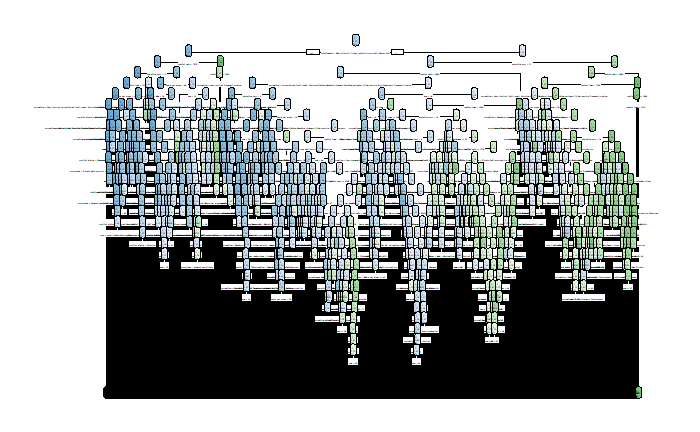
get\_recall(new.train9$pred, new.train9$income) # recall : 0.776

< 각 모델 별 성능 비교>

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **모델 명** | **모델 1** | **모델2** | **모델3** | **모델4** | **모델 5** | **모델6** | **모델 7** |
| **Accuracy** | **0.888** | **0.840** | **0.855** | **0.857** | **0.858** | **0.862** | **0.871** |
| **Precision** | **0.814** | **0.771** | **0.778** | **0.752** | **0.761** | **0.775** | **0.796** |
| **Recall** | **0.713** | **0.509** | **0.586** | **0.638** | **0.629** | **0.632** | **0.649** |
| **Note.** | **Cp=0.0014** | **Cp = 0.0091** | **minsplit = 300** | **minsplit= 250** | **minsplit = 200** | **minsplit = 100** | **minsplit = 50** |

|  |  |  |
| --- | --- | --- |
| **모델 명** | **모델 8** | **모델9** |
| **Acuuracy** | **0.889** | **0.911** |
| **Precision** | **0.825** | **0.856** |
| **Recall** | **0.706** | **0.776** |
| **Note.** | **minsplit = 30** | **minsplit = 10** |

**Question 2. 앞서 만든 decision tree모델을 그려봅시다. 그린 모델(시각화)을 보고 무엇을 알 수 있는지 설명하시오**

**.**

**답 : 수치적으로 가장 성능이 좋은 9번 모델을 그렸다. 하지만 너무 많은 nodes가 있어 아무것도 설명할 수가 없으며 의사결정에 전혀 도움이 되지 않는다.**

**따라서 새로운 의사결정나무 모델을 만들 필요가 있다.**

만약 소득이 50만불 이상인 사람에게 ‘고소득자에게 적합한 신용카드 가입 권유 메일’을 발송하여 가입율과 회사의 이익을 높이는 데에 분석 목적이 갖는다고 가정해보자.

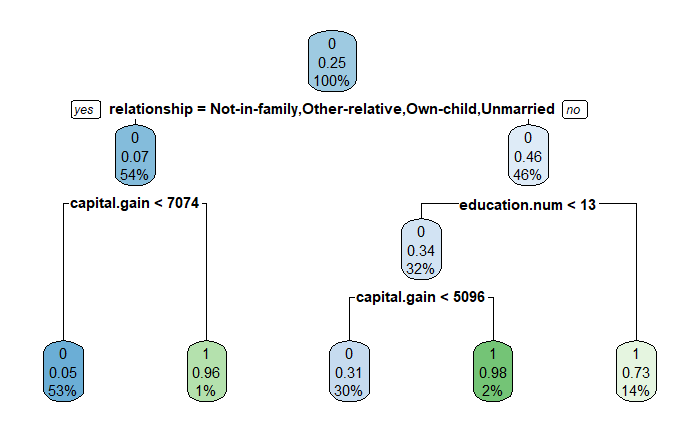
"실제로 고소득자인데 저소득자로 판단 한 사건" 과 "실제는 저소득인데 고소득으로 판단 한 사건" 의 기회비용을 따지면 "실제로 고소득자인데 저소득자로 판단하여 메일을 보내지 않는다" 의 결과가 더 손실이 클 것이다.

의사나무결정 모델의 장점은 직관적이고 한눈에 결과를 확인할 수 있다는 것이다. 만약 의사결정나무 모델이 이 장점을 갖지 못한다면 굳이 의사나무결정모델을 선택할 필요가 없다고 생각한다. 따라서 직관성과 편리함을 유지하면서 현재 가정한 상황을 잘 반영하는 모델을 찾기 위해 maxdepth를 조절했다.

Maxdepth가 6 이상부터는 한눈에 들어오지 않기 때문에 maxdepth를 3,4,5 로 조정했다.

|  |  |  |  |
| --- | --- | --- | --- |
| 모델 명 | 모델 10 | 모델 11 | 모델 12 |
| Acuuracy | 0.840 | 0.840 | 0.842 |
| Precision | 0.771 | 0.771 | 0.810 |
| Recall | 0.509 | 0.509 | 0.481 |
| Note. | maxdepth = 3 | maxdepth = 4 | maxdepth = 5 |

**<모델10>**



**가정한 상황에서는 recall값이 중요하므로 같은 accuracy에서 recall 값이 더 좋은 모델 10을 선택했다(모델 11도 성능과 plot 모양이 같으므로 모델 11을 선택해도 무관하다는 생각이 든다).**

**이 모델은 relationship, capital.gain, ecucaion.num 변수를 사용했으며 4번, 5번, 7번, 12번, 13번 노드가 leaf nodes이다.**

**Relationship이 Not-in-family, Othre-relative, Own-child, Unmarried 인 사람 중에 capital.gain이 7074보다 높은 1%의 사람과 Relationship이 위에 언급된 관계가 아닌 사람 들 중 education.num이 13보다 많은 14%의 사람, Relationship이 위에 언급된 관계가 아닌 사람 들 중 education.num이 13보다 적고 capital.gain이 5096 이상인 2% 사람이 소득이 50만불 이상이라고 분류하고 있다.**

**Question 3. 당신의 decision tree 모델이 overfitting 되지 않았다는 것을 어떻게 알 수 있나요?**

1. test set에서 확인하기

new.test10 <- test

new.test10$pred <- predict(train\_model10 , new.test10, type = "class")

mean(new.test10$income == new.test10$pred) **# accuracy : 0.842**

get\_precision(new.test10$pred, new.test10$income) **# precision : 0.768**

get\_recall(new.test10$pred, new.test10$income) **# recall : 0.522**

1. 교차

set.seed(2019)

n.income <- nrow(test)

rgroup <- runif(n.income)

# test1, test2

test.df1 <- subset(test, rgroup <= 0.6)

test.df2 <- subset(test, rgroup > 0.6)

# test1

new.test.df1 <- test.df1

new.test.df1$pred <- predict(train\_model10 , new.test.df1, type = "class")

mean(new.test.df1$income == new.test.df1$pred) **# accuracy : 0.838**

get\_precision(new.test.df1$pred, new.test.df1$income) **# precision : 0.756**

get\_recall(new.test.df1$pred, new.test.df1$income) **# recall : 0.513**

|  |  |  |  |
| --- | --- | --- | --- |
| 방법 | Train data | Test data | Test data 분리 후 testing |
| Acuuracy | 0.840 | 0.842 | 0.838 |
| Precision | 0.771 | 0.768 | 0.756 |
| Recall | 0.509 | 0.522 | 0.513 |

Testing 한 결과 train data와 두번의 testing 결과가 성능이 비슷하게 나오기 때문에 과적합이 아니라고 생각한다.

KNN

**Question4. kNN 방법으로 test data에 있는 income 변수를 예측하시오. confusion matrix,accuracy, precision, recall를 설명하시오.**

train.knn <- train

test.knn <- test

# one hot encoding

library(dummies)

train.knn <- dummy.data.frame(train.knn)

test.knn <-dummy.data.frame(test.knn)

str(train.knn)

# min-max normalization

minmax\_norm <- function(x) {

(x-min(x))/(max(x)-min(x))

}

train.knn$age <- minmax\_norm(train.knn$age)

train.knn$capital.gain <- minmax\_norm(train.knn$capital.gain)

train.knn$capital.loss <- minmax\_norm(train.knn$capital.loss)

train.knn$hours.per.week <- minmax\_norm(train.knn$hours.per.week)

train.knn$education.num <- minmax\_norm(train.knn$education.num)

train.knn$income <- as.factor(train.knn$income)

train.knn$income <- ifelse(train.knn$income =="1", "rich", "poor")

test.knn$age <- minmax\_norm(test.knn$age)

test.knn$capital.gain <- minmax\_norm(test.knn$capital.gain)

test.knn$capital.loss <- minmax\_norm(test.knn$capital.loss)

test.knn$hours.per.week <- minmax\_norm(test.knn$hours.per.week)

test.knn$education.num <- minmax\_norm(test.knn$education.num)

test.knn$income <- as.factor(test.knn$income)

test.knn$income <- ifelse(test.knn$income =="1", "rich", "poor")

# split data into train and test set

dim(train.knn)

train.knn\_new <- train.knn[, -47]

test.knn\_new <- test.knn[, -47]

train.knn\_label <- train.knn[, 47]

test.knn\_label <- test.knn[, 47]

# choosing proper k

sqrt(nrow(train.knn)) # 146.639

library(class)

train.knn\_pred <- knn(train = train.knn\_new , test = test.knn\_new , cl = train.knn\_label, k = 147)

#accuracy

mean(test.knn\_label == train.knn\_pred) # 0.827

#confusion matrix

cmat <- table(test.knn\_label, train.knn\_pred) # 실제, 예측

cmat

#precision

cmat[2,2] / sum(cmat[,2]) # 0.687

#recall

cmat[2,2] / sum(cmat[2,]) # 0.56

############################################

test.knn\_pred\_P <- knn(train = train.knn\_new , test = test.knn\_new , cl = train.knn\_label, k = 147, prob = TRUE)

head(attributes(test.knn\_pred\_P)$prob,10)

# converting all Prob to P(rich)

test\_pred\_prob <- ifelse(test.knn\_pred\_P == 'rich', attributes(test.knn\_pred\_P)$prob, 1-attributes(test.knn\_pred\_P)$prob)

head(test\_pred\_prob)

##### set lower threshold #####

threshold <- 0.3

test\_pred\_new <- ifelse(test\_pred\_prob > threshold, 'rich', 'poor')

# matrix

cmat <- table(test.knn\_label, test\_pred\_new)

> cmat

test\_pred\_new

test.knn\_label poor rich

poor 5549 1375

rich 458 1833

#accuracy

**mean(test.knn\_label == test\_pred\_new) # 0.801**

#precision

**cmat[2,2] / sum(cmat[,2]) # 0.571**

#recall

**cmat[2,2] / sum(cmat[2,]) # 0.8**

factor 형식의 변수는 더미변수로 바꾸고 거리 계산에서 스케일을 맞추기 위해 정규화를 시행했다

목적변수로 사용할 income 변수를 제거한 dataset과 labelset을 train과 data set 각각 만들었다.

K는 trainset의 observation 제곱근 값인 147을 선택했다.

앞서 세운 가정에 의해 더 높은 recall값을 얻기 위하여 threshold를 0.3으로 내렸고 그 결과 recall값을 0.8까지 올렸다.

**Question 5.** **다양한 k를 시도해보고 가장 성능이 좋은 k를 찾아보시오. k를 변화하면서**

**accuracy, precision, recall, AUC가 변화하는 것을 보기 위해 그래프를 그리시오.**

**가장 성능이 좋은 k는 무엇입니까(AUC 기준으로)?**

**k\_value <- c(10, 30, 50, 70, 90, 110, 130, 147, 170, 200)**

**knn\_accuracy** <- function(k\_value){

test.knn\_pred\_P <- knn(train = train.knn\_new , test = test.knn\_new , cl = train.knn\_label, k = k\_value, prob = TRUE)

test\_pred\_prob <- ifelse(test.knn\_pred\_P == 'rich', attributes(test.knn\_pred\_P)$prob, 1-attributes(test.knn\_pred\_P)$prob)

test\_pred\_new <- ifelse(test\_pred\_prob > 0.3 , 'rich', 'poor')

a <- mean(test.knn\_label == test\_pred\_new)

return(a)

}

**knn\_precision** <- function(k\_value){

test.knn\_pred\_P <- knn(train = train.knn\_new , test = test.knn\_new , cl = train.knn\_label, k = k\_value, prob = TRUE)

test\_pred\_prob <- ifelse(test.knn\_pred\_P == 'rich', attributes(test.knn\_pred\_P)$prob, 1-attributes(test.knn\_pred\_P)$prob)

test\_pred\_new <- ifelse(test\_pred\_prob > 0.3 , 'rich', 'poor')

cmat <- table(test.knn\_label, test\_pred\_new)

a <- cmat[2,2] / sum(cmat[,2])

return(a)

}

**knn\_recall** <- function(k\_value){

test.knn\_pred\_P <- knn(train = train.knn\_new , test = test.knn\_new , cl = train.knn\_label, k = k\_value, prob = TRUE)

test\_pred\_prob <- ifelse(test.knn\_pred\_P == 'rich', attributes(test.knn\_pred\_P)$prob, 1-attributes(test.knn\_pred\_P)$prob)

test\_pred\_new <- ifelse(test\_pred\_prob > 0.3 , 'rich', 'poor')

cmat <- table(test.knn\_label, test\_pred\_new)

a <- cmat[2,2] / sum(cmat[2,])

return(a)

}

**calAUC** <- function(predCol, targetCol){

perf <- performance(prediction(predCol, targetCol), 'auc')

as.numeric(perf@y.values)

}

**knn\_AUC** <- function(k\_value){

test.knn\_pred\_P <- knn(train = train.knn\_new , test = test.knn\_new , cl = train.knn\_label, k = k\_value, prob = TRUE)

test\_pred\_prob <- ifelse(test.knn\_pred\_P == 'rich',

attributes(test.knn\_pred\_P)$prob,

1-attributes(test.knn\_pred\_P)$prob)

a <- calAUC(test\_pred\_prob,test.knn\_label== 'rich')

}

accuracy <- sapply(k\_value, knn\_accuracy)

precision <- sapply(k\_value, knn\_precision)

recall <- sapply(k\_value, knn\_recall)

AUC <- sapply(k\_value, knn\_AUC)

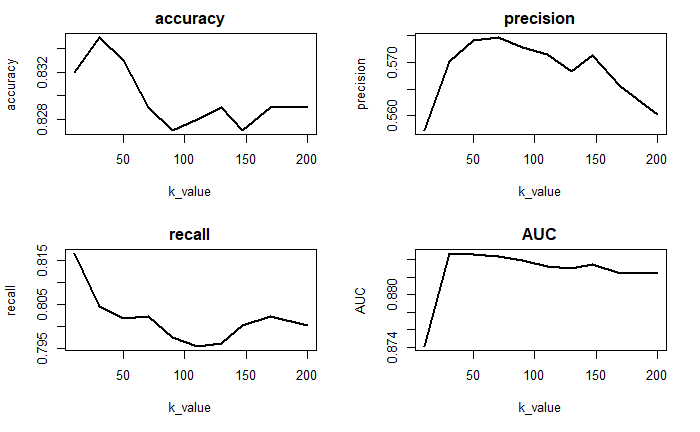
par(mfrow=c(2,2))

plot(k\_value, accuracy, xlim=c(10, 200),type="l", main = "accuracy",lwd=2)

plot(k\_value, precision, xlim=c(10, 200), type="l",main = "precision",lwd=2)

plot(k\_value, recall , xlim=c(10, 200),type="l", main = "recall",lwd=2)

plot(k\_value, AUC, xlim=c(10, 200), type="l",main = "AUC",lwd=2)



답 : 10, 30, 50, 70, 90, 110, 130, 147, 170, 200 의 k값 중에서 가장 좋은 것은 30이다.

**Question 6. best K를 사용한 kNN에 대해서 ROC 커브를 그리고 AUC를 계산해보시오.**

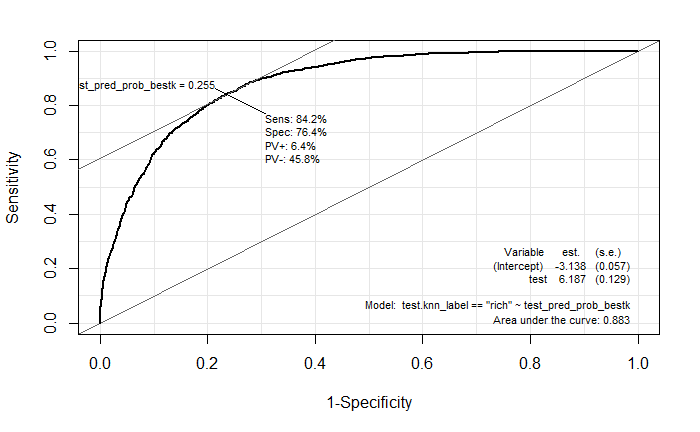
library(ROCR)

library(Epi)

train.knn\_pred\_bestk <- knn(train = train.knn\_new , test = test.knn\_new , cl = train.knn\_label, k = 30)

test\_pred\_prob\_bestk <- ifelse(test.knn\_pred\_P == 'rich', attributes(test.knn\_pred\_P)$prob, 1-attributes(test.knn\_pred\_P)$prob)

ROC(test\_pred\_prob\_bestk, test.knn\_label== 'rich') **# 0.883**



답 : AUC는 0.883이다.

**Question 7.** **데이터에 kNN을 적용하기 위해서 데이터를 어떻게 변형하였나요? 이유와 방**

**법을 설명하시오.**

1) knn은 입력변수가 numeric 변수어야 하므로 dummies library를 사용하여 faoctor 변수를 one-hot encodig을 했다.

train.knn <- dummy.data.frame(train.knn)

test.knn <-dummy.data.frame(test.knn)

2) 거리계산에서 거리의 차이에 대한 스케일을 맞추기 위해 정규화를 진행했다.

# min-max normalization

minmax\_norm <- function(x) {

(x-min(x))/(max(x)-min(x))

}

train.knn$age <- minmax\_norm(train.knn$age)

train.knn$capital.gain <- minmax\_norm(train.knn$capital.gain)

train.knn$capital.loss <- minmax\_norm(train.knn$capital.loss)

train.knn$hours.per.week <- minmax\_norm(train.knn$hours.per.week)

train.knn$education.num <- minmax\_norm(train.knn$education.num)

test.knn$age <- minmax\_norm(test.knn$age)

test.knn$capital.gain <- minmax\_norm(test.knn$capital.gain)

test.knn$capital.loss <- minmax\_norm(test.knn$capital.loss)

test.knn$hours.per.week <- minmax\_norm(test.knn$hours.per.week)

test.knn$education.num <- minmax\_norm(test.knn$education.num)

3) 오류를 방지하기 위해 income 변수를 numeric에서 factor로 변환하고 0이면 poor, 1이면 rich로 바꾸었다.

train.knn$income <- as.factor(train.knn$income)

test.knn$income <- as.factor(test.knn$income)

train.knn$income <- ifelse(train.knn$income =="1", "rich", "poor")

test.knn$income <- ifelse(test.knn$income =="1", "rich", "poor")

**Question 8.** **해당 데이터 셋에서 decision tree, kNN 두가지 모델을 비교했을 때 어떤 것이**

**성능이 더 좋은가요? 그 이유는 무엇이라고 생각하나요?**

|  |  |  |
| --- | --- | --- |
| 평가지표 | Decision tree | KNN |
| Accuracy (mean) | 0.841 | 0.832 |
| Precision (mean) | 0.771 | 0.557 |
| Recall (mean) | 0.509 | 0.816 |

답 : Decision tree의 각 평가 지표 값은 depth가 4,5,6 인 모델의 평균이고 KNN의 각 평가지표 값은 5번에서 계산한 값들의 평균이다.

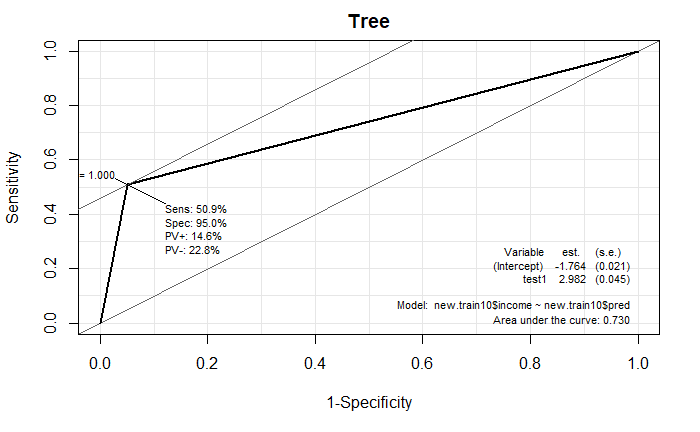
Decision tree는 위에서 확인했듯 Accuracy, Precision, Recall 값을 높이려면 node의 수를 증가시키면 된다. 하지만 증가시킬수록 모델은 복잡해지고, 의사결정에 도움이 되지 않는다 그래서 적은 깊이를 갖는 모델을 선택했는데 Precision에 비해 recall이 낮다,

그러나 KNN의 경우 적절히 threshold 값을 조정하여 모델을 만든 결과 0.8 이상의 recall값을 얻었다. 또한 AUC 값의 경우 0.882 이며 이는 꽤 괜찮은 값을 갖는다.

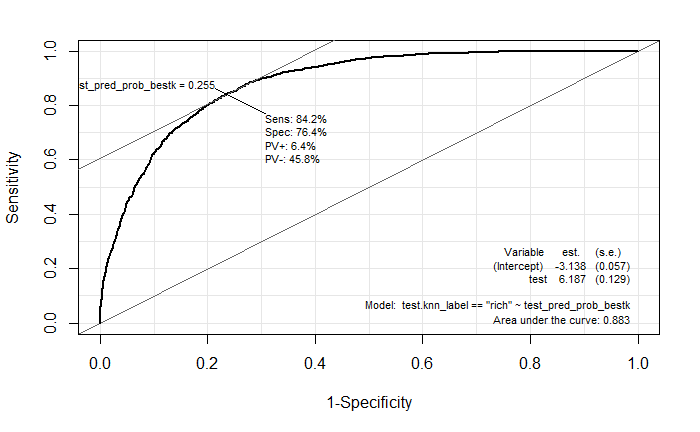
따라서 Decision Tree보다 KNN이 성능이 더 좋다는 판단이 든다.

또한 decision Tree에서 최적 모델이라고 생각한 것의 AUC를 계산해본 결과 0.730이 나왔다.

ROC(test=new.train10$pred, new.train10$income, plot="ROC", AUC=T,main="Tree")



이에 비해 KNN의 경우 최적 모델이라고 평가한 K=30일때의 ROC 그래프와 AUC 값을 보면



**AUC값이 0.883으로 KNN이 더 성능이 좋다고 할 수 있다.**

**또한 mean split이 10이었던 모델9의 train set에서의 성능은**

**accuracy : 0.911**

**precision : 0.856**

**recall : 0.776**

**AUC : 0.867**

**이었지만, test set에서는**

**accuracy : 0.842**

**precision : 0.768**

**recall : 0.522**

**AUC : 0.766**

**으로 전체적으로 성능이 떨어져 과적합이 우려된다.**

Decision tree의 성능이 더 낮은 이유는

연속형 변수를 비연속적인 값으로 취급하기 때문에 분리의 경계점 근방에서는 예측 오류가 클 가능성이 있기 때문이다.

변수 형태를 살펴보면 입력변수 11개 중 5개가 연속형이기 때문에 거의 절반 이상이 비연속적 값이므로 경계점 근방에서 오류가 날 확률이 비교적 높을 수 있다. 만약 전처리 과정에서 미리 구간을 정해서 factor로 바꿔줬다면 오류가 덜 나지 않았을까 라는 추측도 해본다.

또한 의사결정나무는 분류 시 수직, 수평적으로 데이터가 분류가 가능할 때 분류율이 높아지는데. 이번 분석에서는 데이터가 수직, 수평적으로 분류가 안됐기 때문에 분류율이 낮아져서 KNN보다 성능이 낮을 수도 있다는 생각이 든다.

**Question 9.** **여러분이 만든 모델의 성능을 여러분의 Group Member들과 비교해보세요. 차이점을 파악하고 여러분의 모델에서 개선해야할 점이 있다면 개선해봅시다.**

답 : 조원들은 k 값을 30, 24, 43, 25로 했는데 다들 AUC가 0.883이 나왔다.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| K값 | 24 | 25 | 30 | 43 | 147 |
| accuracy | 0.837 | 0.838 | 0.834 | 0.834 | 0.801 |
| Precision | 0.707 | 0.710 | 0.699 | 0.702 | 0.571 |
| Recall | 0.590 | 0.590 | 0.584 | 0.578 | 0.80 |

Accuracy 값은 k 값이 변해도 비슷한 반면 precision과 recall은 147일 때 precision은 낮고 recall은 높은 것을 확인 할 수 있다.