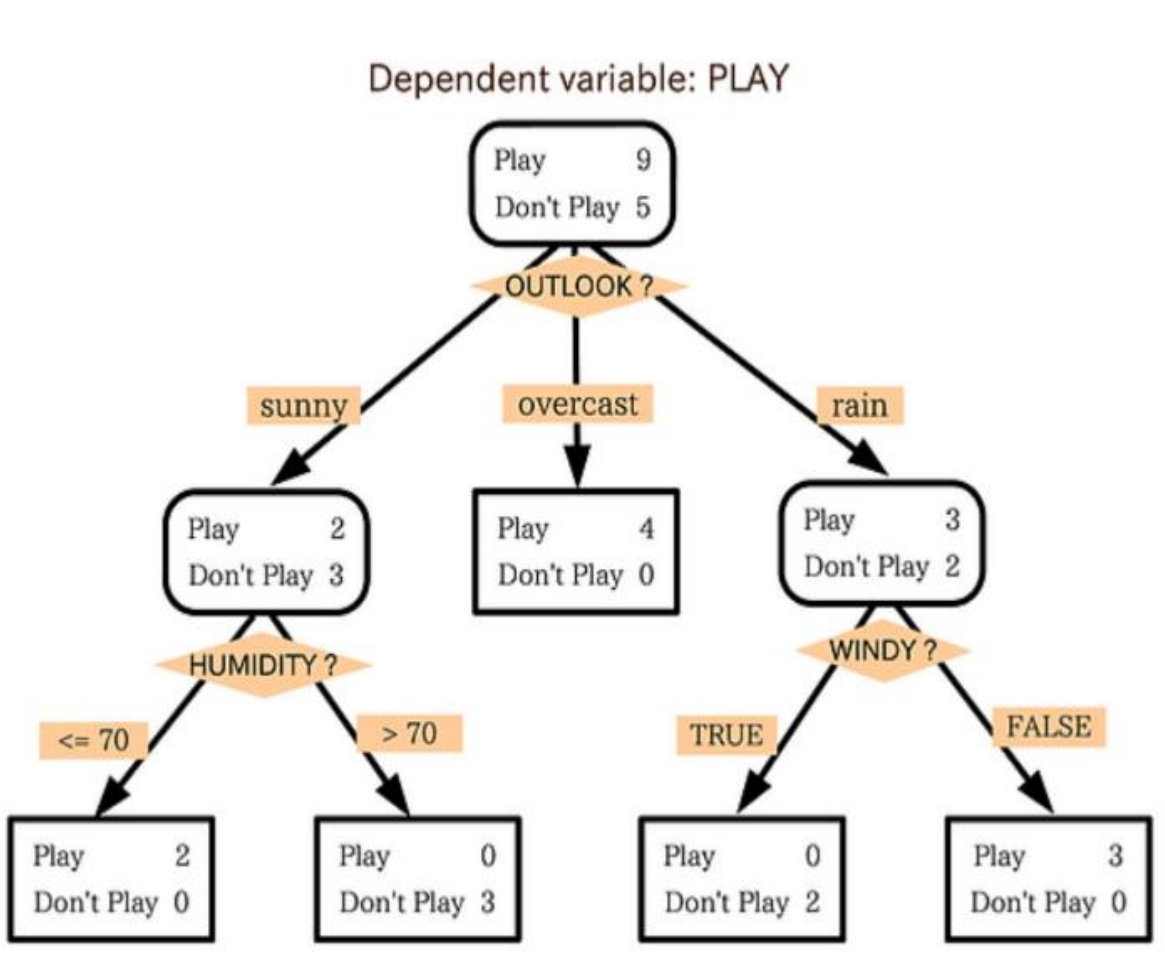
ML Session

03 decision tree + parameter tuning

INDEX

Decision tree의 개요



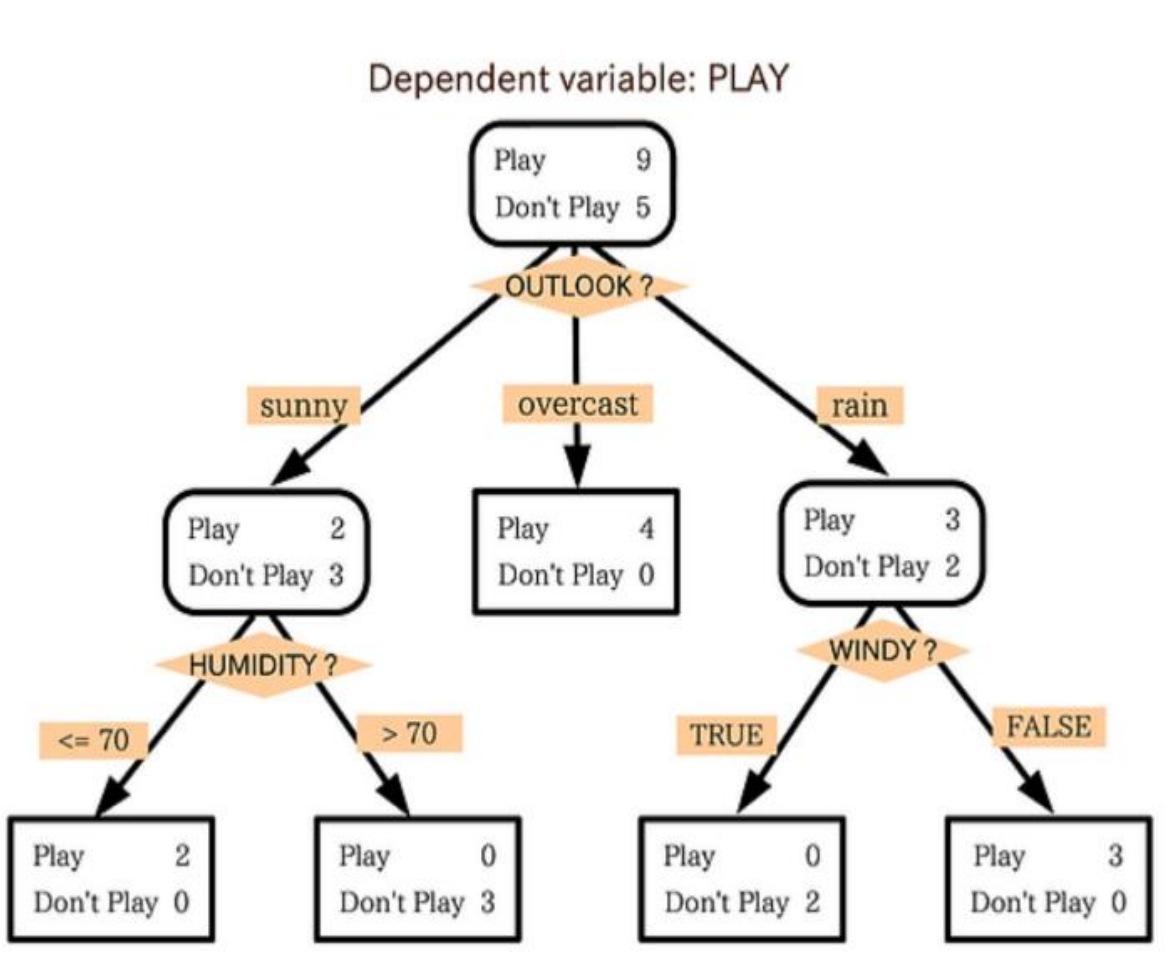
< 의사결정 나무 >

질문을 던져서 대상을 좁혀나가는 '스무고개' 놀이와 비슷한 개념

분류(classification)와 회귀(regression) 모두 가능.

분류 결과를 인간이 쉽게 이해할 수 있는 규칙으로 표현

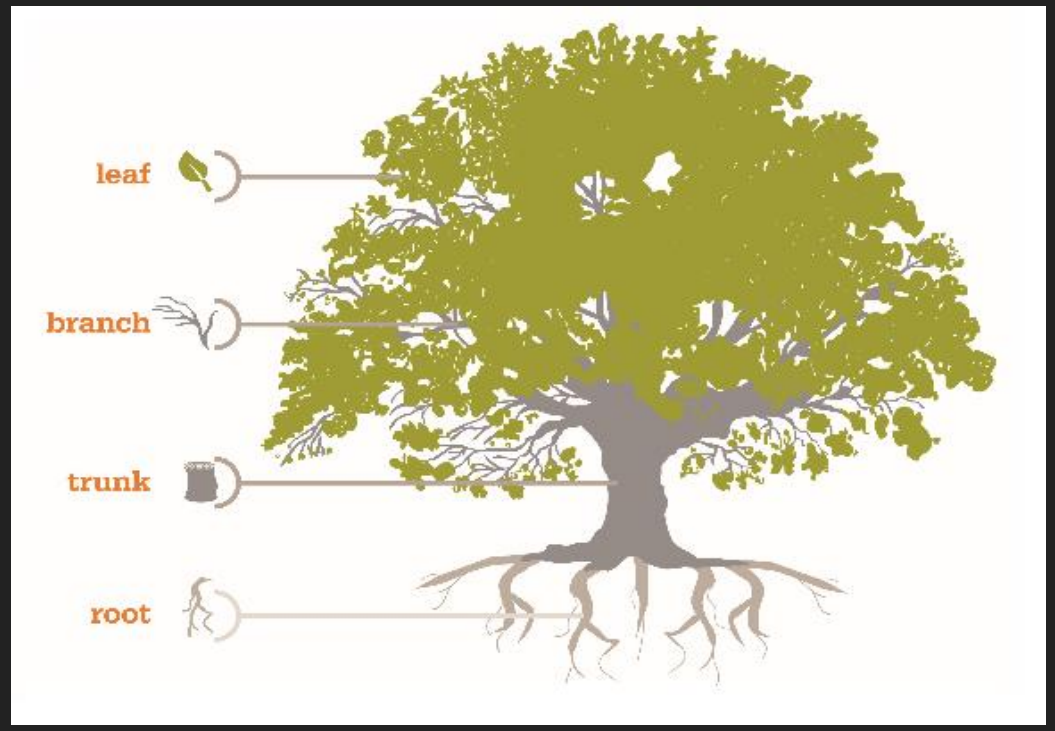
Decision tree의 개요



< 의사결정 나무 >

데이터를 분석하여 이들 사이에 존재하는 패턴을 예측 가능한 규칙들의 조합으로 나타내며,

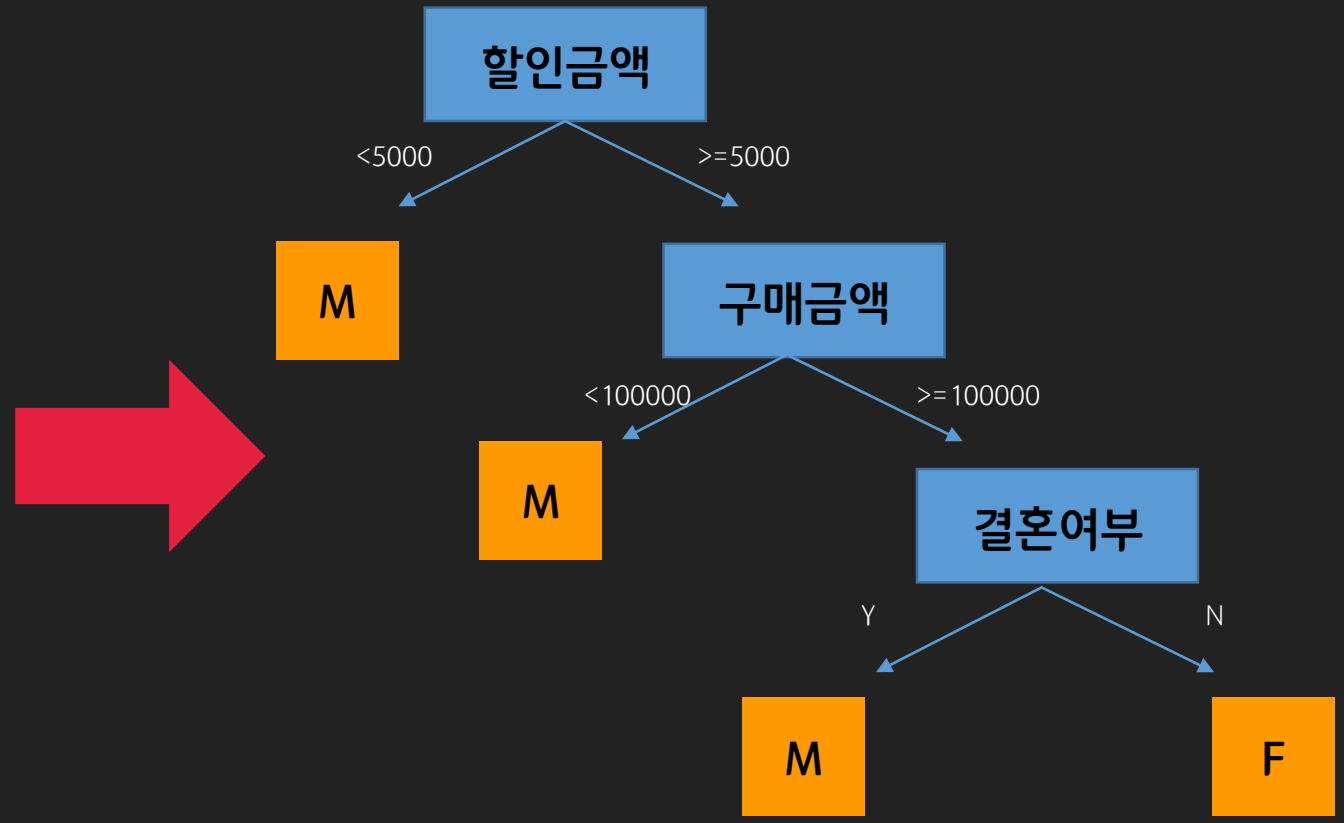
그 모양이 '나무'와 같다고 해서 의사결정나무라 불립니다.



Decision tree 의 구성

X Y

할인금액	구매금액	구매 시간대	결혼여부	성별
0	100000	19	Y	M
50000	280000	14	Y	F
13000	56000	20	Y	M
100000	780000	11	N	F
5000	35000	17	N	F



다수의 feature(X)와 class 간(Y)의 관계에 대한 통찰을 얻을 수 있기 때 문에 데이터 탐색 과정에서도 널리 활용

-> class를 잘 나누는 feature : Y를 잘 설명하는 feature

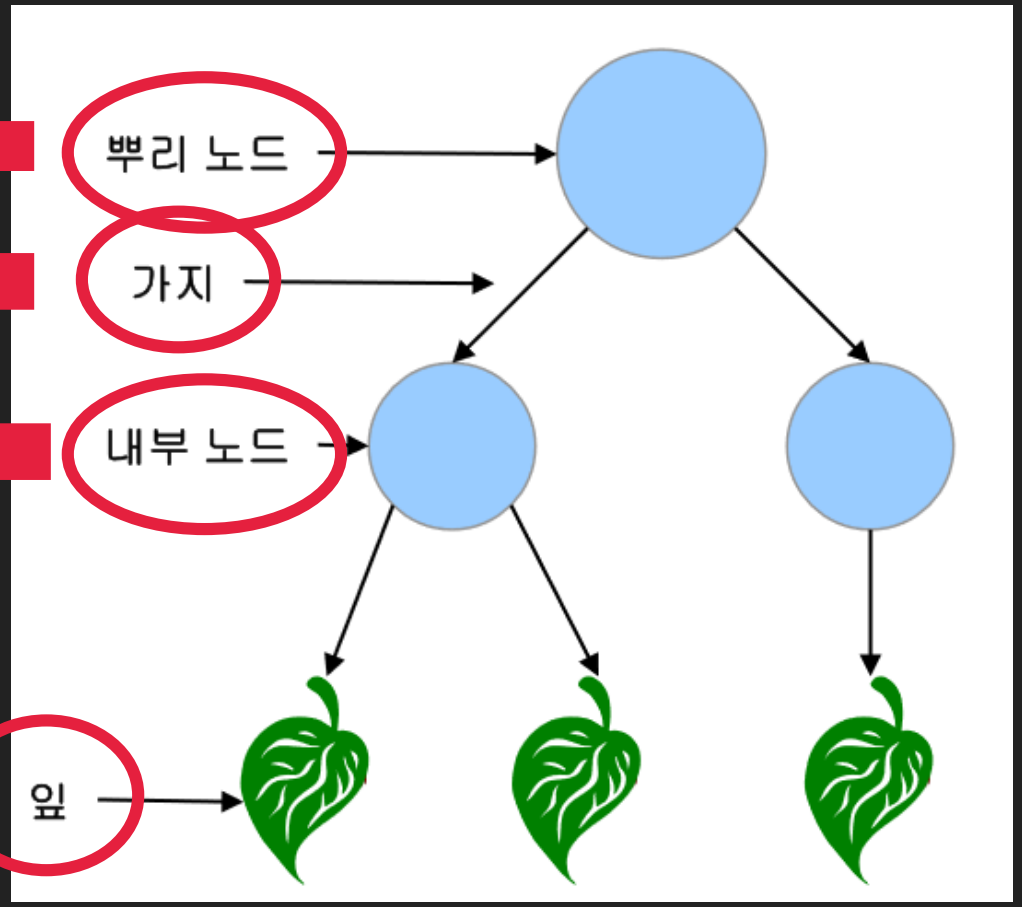
Decision tree 의 구성

Root node - 최상단에 위치 (스무고개의 첫 번째 질문)

Link - 마디와 마디를 이어줌 (질문에 대한 답)

Internal node - 속성의 분리 기준을 포함 (중간 질문)

Leaf - 최종 분류 값을 가짐 (결정)

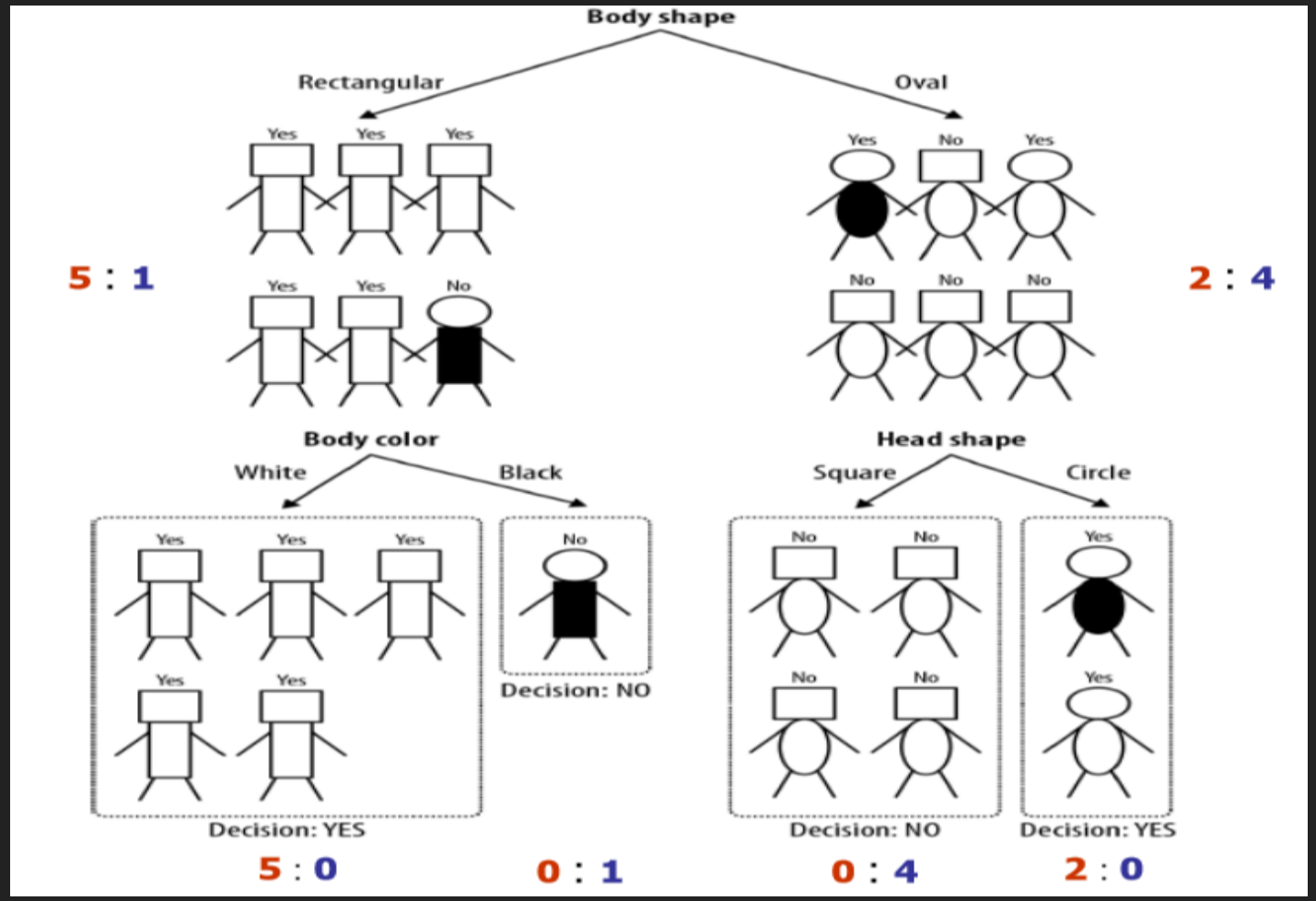


Decision tree의 형성

- 순수도 : class의 특정 범주에 개체들이 포함되는 정도

목표변수 측면에서 부모노드보다 더 순수도(purity)가 높은 자식노드들이 되도록,

데이터를 반복적으로 더 작은 집단으로 나눈다(repeatedly split)



순수도가 높아진다.

즉, 확실히 나뉘어 진다.

Decision tree의 형성

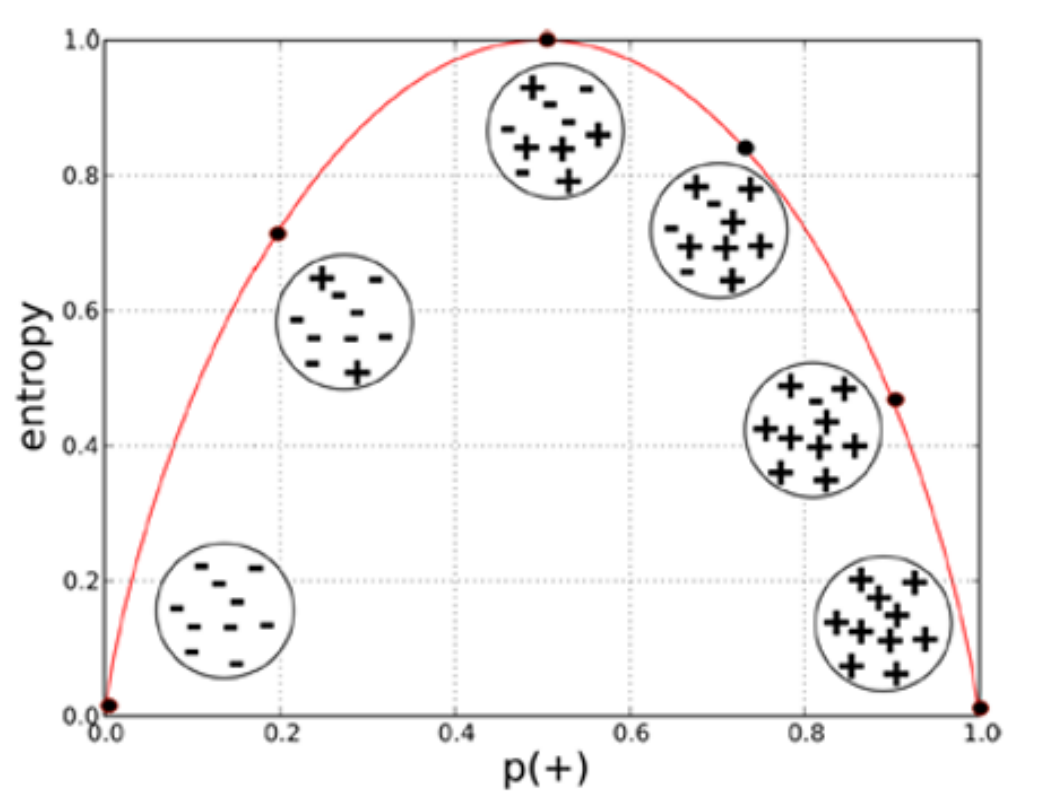
순수도 척도

- 범주형 클래스 의 순수도 척도 : 지니(Gini)
  - 엔트로피(Entropy)
  - 정보 이익 비율(Information gain ratio)
  - 카이제곱 검정(Chi-square test)
- 수치형 클래스 의 순수도 척도 : 분산의 감소(reduction in variance)
  - F 검정

Decision tree의 형성

엔트로피(Entropy)

데이터의 무질서 정도를 측정 할 수 있는 방법

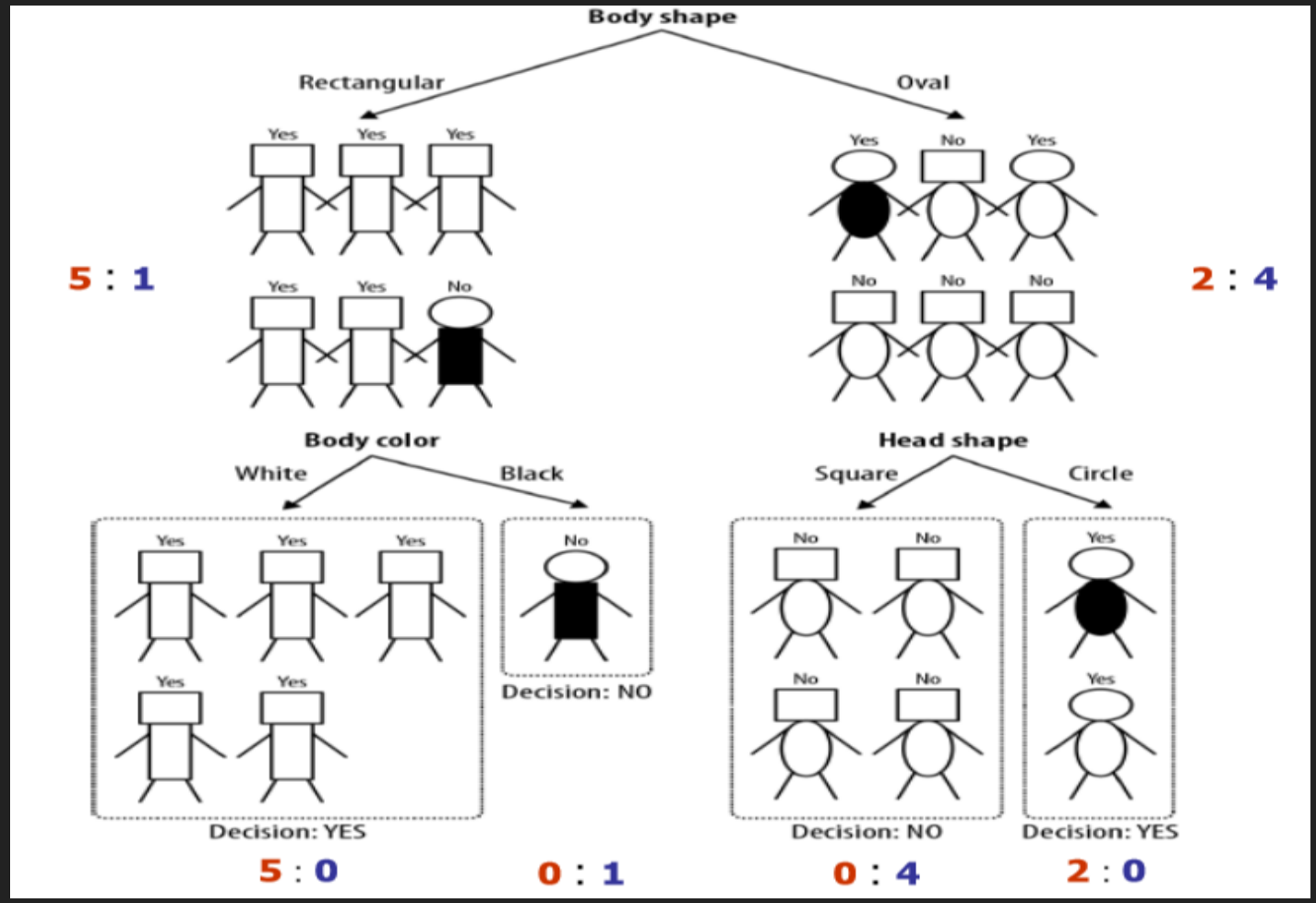


Decision tree의 형성

- 순수도 : class의 특정 범주에 개체들이 포함되는 정도

목표변수 측면에서 부모노드보다 더 순수도(purity)가 높은 자식노드들이 되도록,

데이터를 반복적으로 더 작은 집단으로 나눈다(repeatedly split)

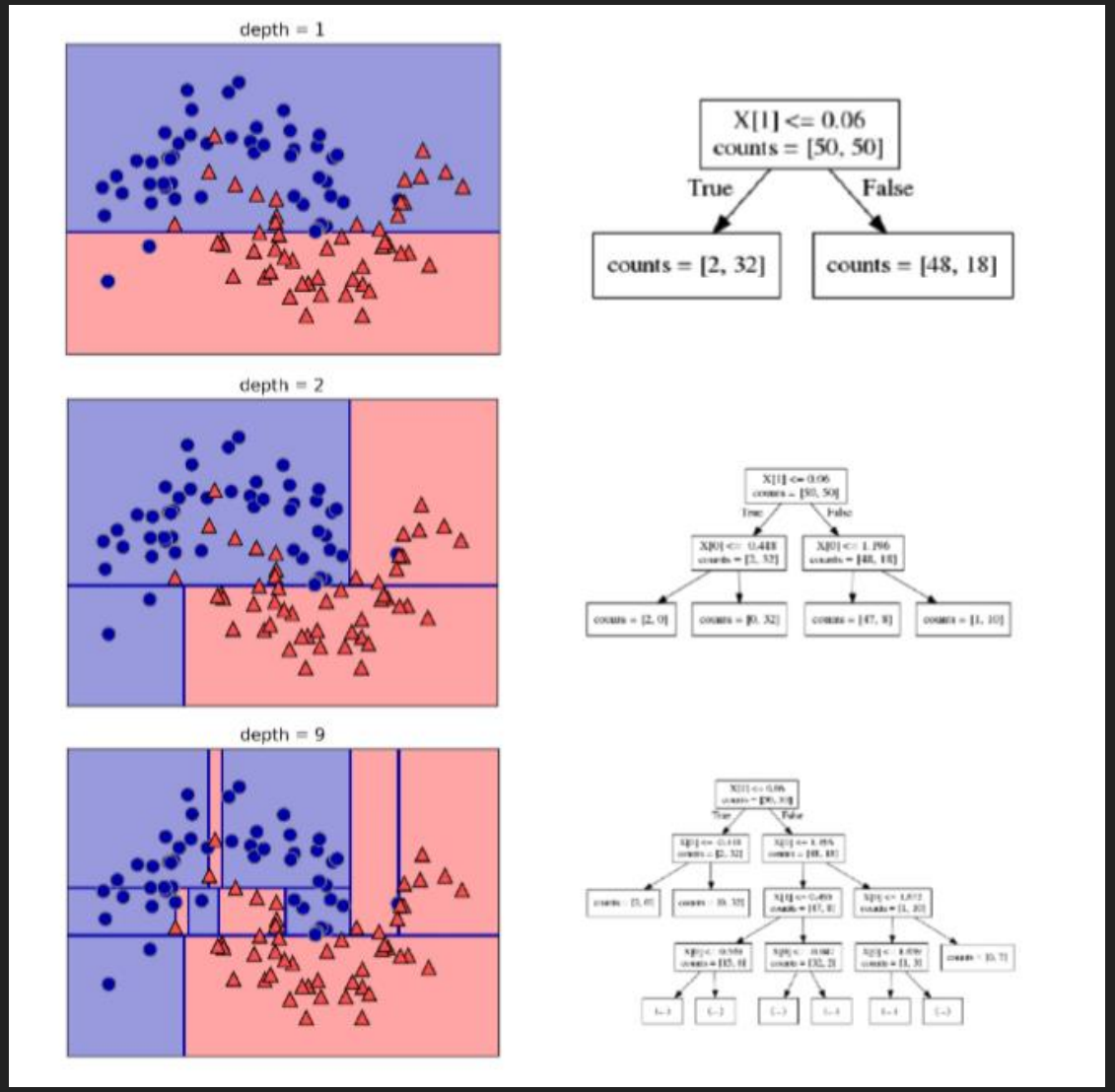


순수도가 높아진다.

엔트로피 감소

불확실성 감소

Decision tree의 형성



가지가 많아지면 즉, 나무의 깊이가 깊을수록

즉, 최대나무(full tree)는 학습 데이터에 최적화된다

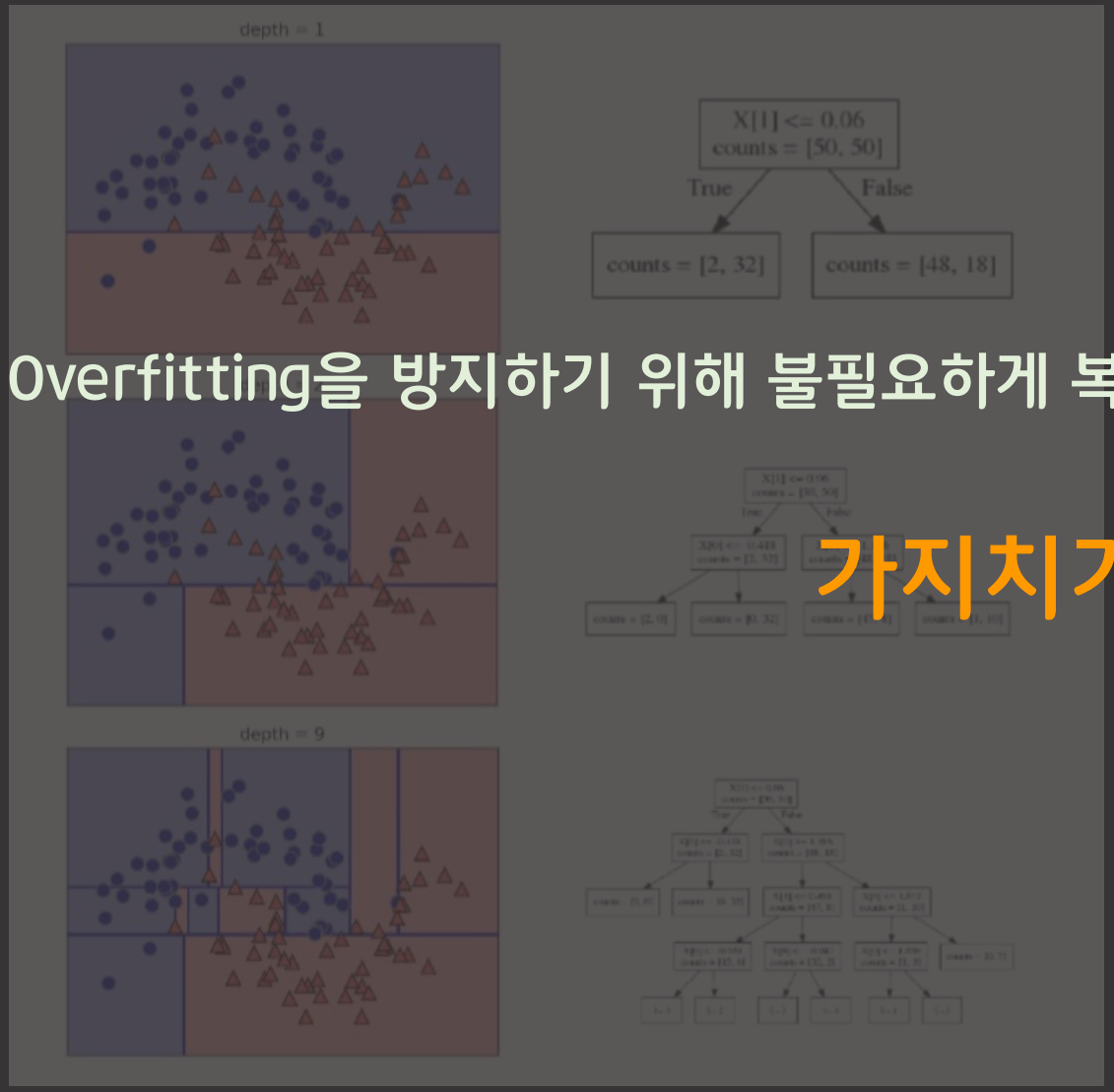
Decision tree 형성

최대나무가 새로운 데이터에 대해서도 가장 잘 분류하는 나무일까?

즉, 최대나무(full tree)는 학습 데이터에 최적화된다

No! Overfitting이 발생한다!

Decision tree 형성



Overfitting을 방지하기 위해 불필요하게 복잡해진 나무의 의미없는 가지를 제거하는 작업 :

즉, 최대나무(full tree)는 학습 데이터에 최적화된다

가지치기(Pruning)

가지치기

가지치기

- 사전 가지치기 : 트리 생성을 일찍 중단하는 방법
- 사후 가지치기 : 트리를 만든 후에 데이터 포인트가 적은 노드를 삭제하거나 병합하는 방법

가지치기

사전 가지치기 하는 방법 : 하이퍼파라미터 (=규제)를 추가한다.

->현재의 노드에서 더 이상 분리가 일어나지 못하게 하는 규칙

- Minsplit : 노드에 속하는 자료가 일정한 수 이하일 때

Ex) minsplit = 10 : 노드에 속하는 자료가 10개 이하일때 더 이상 나무를 크게 만들지 말아라(분리X).

- Maxdepth : 뿌리노드로부터 깊이가 일정 수 이상일 때

Ex) maxdepth = 5: 깊이가 5이상이면 더 이상 나무를 크게 만들지 말아라(분리X).

->즉 , 깊이가 5인 나무를 만들어라.

가지치기

사전 가지치기 하는 방법 : 파라미터 (=규제)를 추가한다. ->현재의 노드에서 더 이상 분리가 일어나지 못하게 하는 규칙

- Minsplit : 노드에 속하는 자료가 일정한 수 이하일 때

Ex) Minsplit = 10: 노드에 속하는 자료가 10개 이하일 때 더 이상 나무를 크게 만들지 말아라(분리X).

- Maxdepth : 루트노드로부터 깊이가 일정 수 이상일 때

Ex) maxdepth = 5: 깊이가 5이상이면 더 이상 나무를 크게 만들지 말아라(분리X).

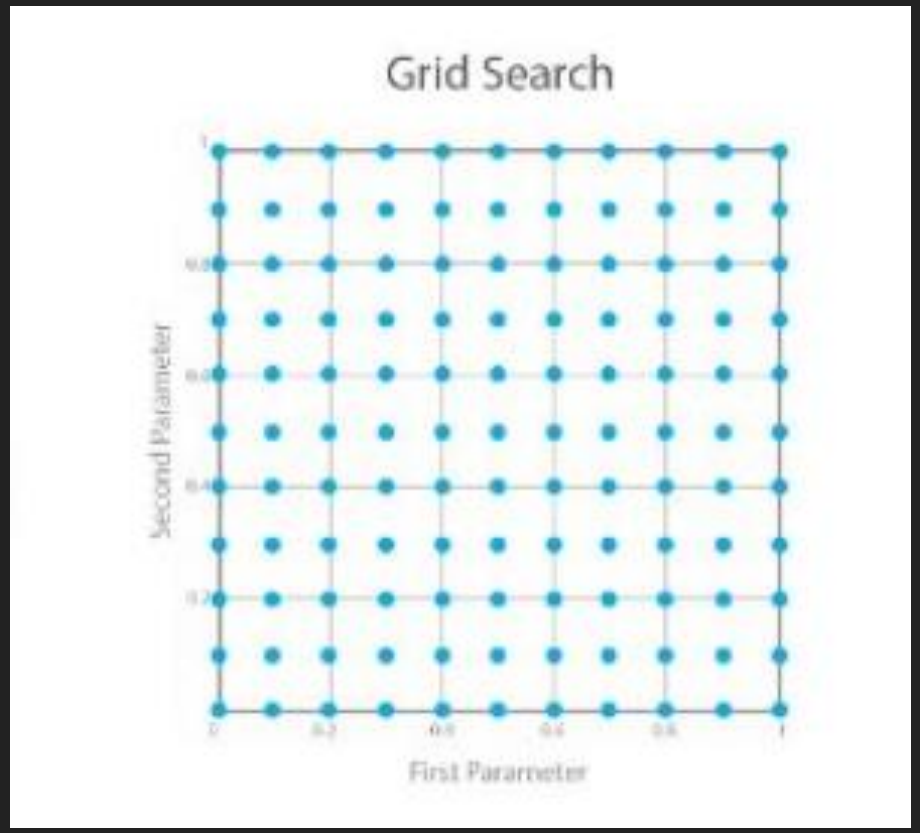
->즉 , 깊이가 5인 나무를 만들어라.

그럼 일반화 할 수 있고 데이터에 알맞은 모델의 파라미터와 그 조합을 어떻게 알지?

파라미터 튜닝(Parameter tuning)

Parameter tuning

- Grid search : 각 하이퍼파라미터에 적용해볼 값들을 미리 정해두고,
  모든 조합(combination)을 시행하여 최적의 조합을 찾는 방법을 말한다



- 장점은 간단하다
- 단점은 튜닝 대상인 하이퍼파라미터의 개수가 많아지면 필요
  한 시행횟수가 기하급수적으로 늘어난다는 것이다.

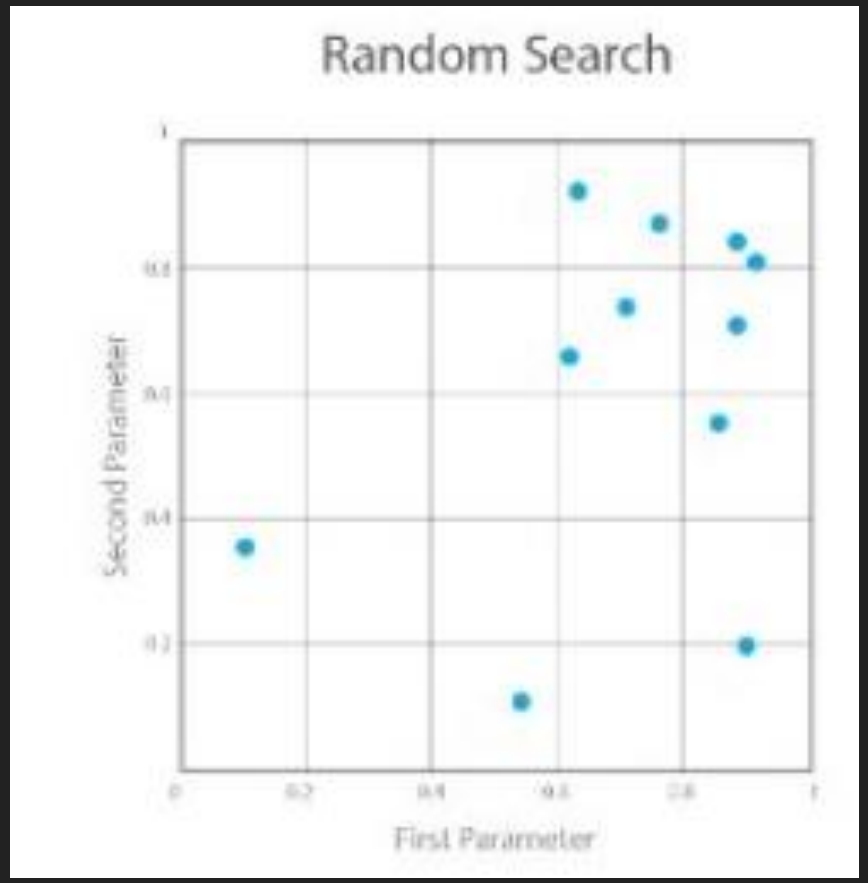
Ex) minsplit = c(1,2,3,4,5,6,7,8,9,10)

maxdepth = c(6,7,8,9,10,11,12)

-> 총 최적의 조합을 찾기위해 총 10X7=70번 시행.

Parameter tuning

- Random search :각 하이퍼파라미터의 최소값-최대값을 정해두고,
  범위 내에서 무작위 값을반복적으로 추출하여 최적의 조합을 찾는 방법을 말한다



- 장점은 시간대비 성능은 Random search가 훨씬 좋다고 평가
- 단점은 완전 최적의 파라미터의 조합을 찾을 수 없다.

의사결정 나무 실습

실습하기전 알아야 할 점!

R에서는 의사결정 나무 패키지가 크게 총 3가지가 있다.

- RPART : 전체 데이터셋을 갖고 시작하여 반복해서 두 개의 자식 노드를 생성하기 위해 모든 예측 변수를 사용하여 데이터 셋의 부분집합을 쪼갬으로써 의사결정 트리를 생성함 (즉, 이진분류 밖에 안됨.) / 지니계수 이용 / 가지치기 필요.

- TREE : binary recursive partitioning방법 사용./ 엔트로피 이용/ 가지치기 필요

- PARTY :p-test 를 거친 중요도 기준으로 가지치기 할 변수를 결정해서 가지치기 할 필요 없음

의사결정 나무 실습

Spam 데이터 살펴보기

X

Y

charSemicolon	charRoundbracket	charSquarebracket	charExclamation	charDollar	charHash	capitalAve	capitalLong	capitalTotal	type
0.000	0.000	0.000	0.778	0.000	0.000	3.756	61	278	spam
0.000	0.132	0.000	0.372	0.180	0.048	5.114	101	1028	spam
0.010	0.143	0.000	0.276	0.184	0.010	9.821	485	2259	spam
0.000	0.137	0.000	0.137	0.000	0.000	3.537	40	191	spam
0.000	0.135	0.000	0.135	0.000	0.000	3.537	40	191	spam
0.000	0.223	0.000	0.000	0.000	0.000	3.000	15	54	spam
0.000	0.054	0.000	0.164	0.054	0.000	1.671	4	112	spam
0.000	0.206	0.000	0.000	0.000	0.000	2.450	11	49	spam
0.000	0.271	0.000	0.181	0.203	0.022	9.744	445	1257	spam
0.040	0.030	0.000	0.244	0.081	0.000	1.729	43	749	spam
0.000	0.000	0.000	0.462	0.000	0.000	1.312	6	21	spam
0.022	0.044	0.000	0.663	0.000	0.000	1.243	11	184	spam
0.000	0.056	0.000	0.786	0.000	0.000	3.728	61	261	spam

의사결정 나무 실습

1. 데이터 분할

```
inTrain <- createDataPartition(y = spam$type, p = 0.75, list = FALSE)
```

CreateDataPartition은 데이터를 분할해 주는 함수!

CreateDataPartition(y=y값 ,즉 분할 기준, p= train data set를 만들기 위한 비율 , list= list로 반환할것인가)

위에 코드를 해석 하면 y값은 spam 데이터의 type이라는 열이고, 전체데이터를 0.75:0.25=train 데이터:test 데이터로 만들것.

InTrain을 살펴보면 전체 데이터의 75%의 인덱스 값이 random으로 뽑혀져 있다.

```
training <- spam[inTrain, ]
testing <- spam[-inTrain, ]
```

inTrain에는 75%의 인덱스 값이 들어가져 있기 때문에 inTrain에 해당하는 행을 추출하면 전체 데이터의 75%의 데이터만 뽑을 수 있고 inTrain에 해당하지 않은 행을 추출하면 전체 데이터의 25%의 데이터만 뽑을수 있다.

즉, 전체 train 데이터와 test 데이터를 나눌 수 있다.

의사결정 나무 실습

2. Train 데이터 모델링

```
modelFit <- train(type ~ ., data = training, method = "rpart")
```

Train 함수를 사용하면 training 데이터를 훈련 시킬수 있다.

Train(y에 대한 열이름 ~ . , data = train 데이터 , method = '모델이름')

위에 모델 이름에 rpart 라고 쓴 이유는 caret 패키지에서는 의사결정나무를 그릴때 rpart라는 방법으로 그리는 방법 밖에 지원을 해주지 않기 때문이다.

75%의
Train data

25%의 Test data

Train data를 통해서 decision tree(rpart 방법)(모델)를 훈련시킨 것 이다.

How? Y값(스팸여부) 에 따른 x값들을 보고 x값들의 특징을 잡아내어 a특징일 때는 스팸일 것 이다. B 특징일때는 스팸이 아닐 것 이다.라고 훈련함.

의사결정 나무 실습

3. 예측하기

```
predictions <- predict(modelFit, newdata = testing)
```

75%의
Train data

modelFit 이라는 모델을 train data의 y값에 대한 x값을 보고 훈련시켰다.
얼마나 x값의 특징을 잘 잡아냈는지 살펴보자.

25%의 Test data

modelFit이라는 모델을 Test data에 적용 시킨다.
그럼, modelFit이라는 모델은 Test data의 x값들을 보고 y값 ,
즉 스팸여부를 예측할것이다.

Predictions 안에는 modelFit이 예측한 TEST data의 Y값이 담겨져 있다.

의사결정 나무 실습

4. 성능체크

예측값 실제값

```
confusionMatrix(predictions, testing$type)
```

예측값과 실제값을 비교 하여 이 모델이 얼마나 잘 맞추는지 알 수 있다.

```
Confusion Matrix and Statistics

          Reference
Prediction nonspam spam
   nonspam     646  121
   spam         51  332

               Accuracy : 0.8504
                 95% CI : (0.8285, 0.8706)
    No Information Rate : 0.6061
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.6781

 Mcnemar's Test P-Value : 1.431e-07

            Sensitivity : 0.9268
            Specificity : 0.7329
         Pos Pred Value : 0.8422
```

정확도 : 85.04%

의사결정 나무 실습

5. 파라미터 튜닝 & 모델 훈련

```
cv <- trainControl(method = "cv", number = 5, verbose = T)
```

Cross-validation 실행!
trainControl(method='cv',number=몇번 나누어서 할것인지, verbose=T : 실행과정이 직접보인다)

```
grid = expand.grid(maxdepth = c(3,5,7,9,10,11,12))
modelFit <- train(type ~ ., data = training, method = "rpart2",tuneGrid = grid,trControl=cv)
```

tuneGrid= 파라미터 튜닝의 범위, trControl=cross-validation코드

의사결정 나무 실습

6. 성능체크

```
predictions <- predict(modelFit, newdata = testing)
confusionMatrix(predictions, testing$type)
```

예측값과 실제값을 비교 하여 이 모델이 얼마나 잘 맞추는지 알 수 있다.

```
spam           44  378

               Accuracy : 0.8965
                 95% CI : (0.8775, 0.9135)
    No Information Rate : 0.6061
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.7807

 Mcnemar's Test P-Value : 0.005958

            Sensitivity : 0.9369
```

정확도 : 89.65%

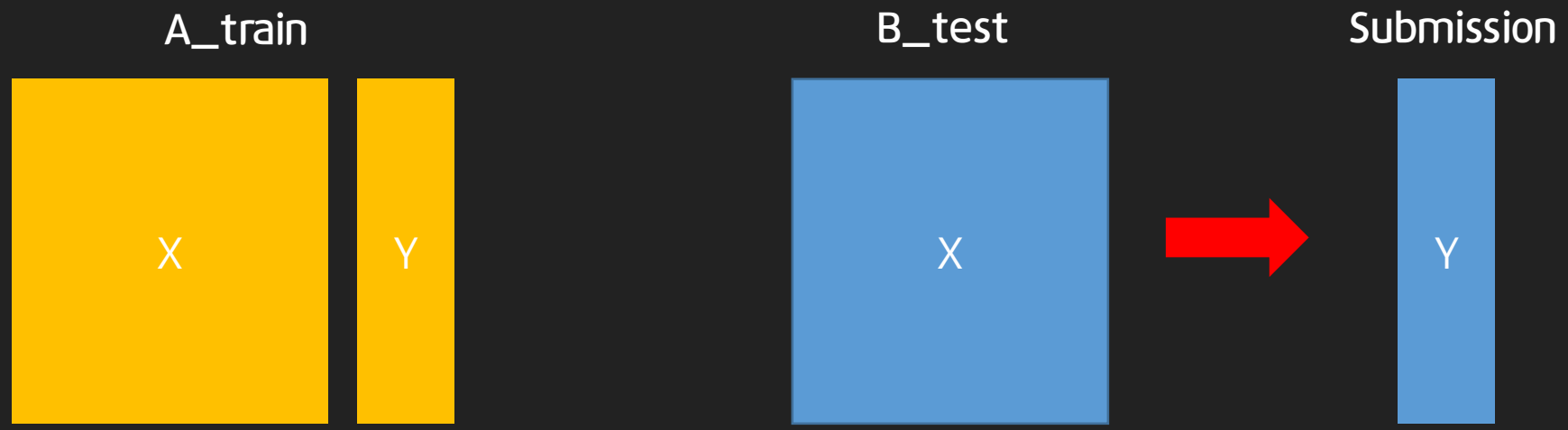
참고(Caret에서 지원해주는 모델)

```
  [1] "ada"           "AdaBag"        "AdaBoost.M1"
  [4] "adaboost"      "amdai"         "ANFIS"
  [7] "avNNet"        "awnb"          "awtan"
 [10] "bag"           "bagEarth"      "bagEarthGCV"
 [13] "bagFDA"        "bagFDAGCV"     "bam"
 [16] "bartMachine"   "bayesglm"      "binda"
 [19] "blackboost"    "blasso"        "blassoAveraged"
 [22] "bridge"        "brnn"          "BstLm"
 [25] "bstSm"         "bstTree"       "C5.0"
 [28] "C5.0Cost"      "C5.0Rules"     "C5.0Tree"
 [31] "cforest"       "chaid"         "CSimca"
 [34] "ctree"         "ctree2"        "cubist"
 [37] "dda"           "deepboost"     "DENFIS"
 [40] "dnn"           "dwdLinear"     "dwdPoly"
 [43] "dwdRadial"     "earth"         "elm"
 [46] "enet"          "evtree"        "extraTrees"
 [49] "fda"           "FH.GBML"       "FIR.DM"
 [52] "foba"          "FRBCS.CHI"     "FRBCS.W"
 [55] "FS.HGD"        "gam"           "gamboost"
 [58] "gamLoess"      "gamSpline"     "gaussprLinear"
 [61] "gaussprPoly"   "gaussprRadial" "gbm_h2o"
 [64] "gbm"           "gcvEarth"      "GFS.FR.MOGUL"
 [67] "GFS.GCCL"      "GFS.LT.RS"     "GFS.THRIFT"
 [70] "glm.nb"        "glm"           "glmboost"
 [73] "glmnet_h2o"    "glmnet"        "glmStepAIC"
 [76] "gpls"          "hda"           "hdda"
 [79] "hdrda"         "HYFIS"         "icr"
 [82] "J48"           "JRip"          "kernelpls"
 [85] "kknn"          "knn"           "krlsPoly"
 [88] "krlsRadial"    "lars"          "lars2"
 [91] "lasso"         "lda"           "lda2"
 [94] "leapBackward"  "leapForward"   "leapSeq"
 [97] "Linda"         "lm"            "lmStepAIC"
[100] "LMT"           "loclda"        "logicBag"
[103] "LogitBoost"    "logreg"        "lssvmLinear"
[106] "lssvmPoly"     "lssvmRadial"   "lvq"
[109] "M5"            "M5Rules"       "manb"
[112] "mda"           "Mlda"          "mlp"
[115] "mlpML"         "mlpSGD"        "mlpWeightDecay"
[118] "mlpWeightDecayML" "monmlp"     "msaenet"
[121] "multinom"      "naive_bayes"   "nb"
[124] "nbDiscrete"    "nbSearch"      "neuralnet"
[127] "nnet"          "nnls"          "nodeHarvest"
[130] "oblique.tree"  "OneR"         "ordinalNet"
[133] "ORFlog"        "ORFpls"        "ORFridge"
[136] "ORFsvm"        "ownn"          "pam"
[139] "parRF"         "PART"          "partDSA"
[142] "pcaNNet"       "pcr"           "pda"
[145] "pda2"          "penalized"     "PenalizedLDA"
[148] "plr"           "pls"           "plsRglm"
[151] "polr"          "ppr"           "PRIM"
[154] "protoclass"    "pythonKnnReg"  "qda"
[157] "QdaCov"        "qrf"           "qrnn"
[160] "randomGLM"     "ranger"        "rbf"
[163] "rbfDDA"        "Rborist"       "rda"
[166] "regLogistic"   "relaxo"        "rf"
[169] "rFerns"        "RFlda"         "rfRules"
[172] "ridge"         "rlda"          "rlm"
[175] "rmda"          "rocc"          "rotationForest"
[178] "rotationForestCp" "rpart"      "rpart1SE"
[181] "rpart2"        "rpartCost"     "rpartScore"
[184] "rqlasso"       "rqnc"          "RRF"
[187] "RRFglobal"     "rrlda"         "RSimca"
[190] "rvmLinear"     "rvmPoly"       "rvmRadial"
[193] "SBC"           "sda"           "sdwd"
[196] "simpls"        "SLAVE"         "slda"
[199] "smda"          "snn"           "sparseLDA"
[202] "spikeslab"     "spls"          "stepLDA"
[205] "stepQDA"       "superpc"       "svmBoundrangeString"
[208] "svmExpoString" "svmLinear"     "svmLinear2"
[211] "svmLinear3"    "svmLinearWeights" "svmLinearWeights2"
[214] "svmPoly"       "svmRadial"     "svmRadialCost"
[217] "svmRadialSigma" "svmRadialWeights" "svmSpectrumString"
[220] "tan"           "tanSearch"     "treebag"
[223] "vbmpRadial"    "vglmAdjCat"    "vglmContRatio"
[226] "vglmCumulative" "widekernelpls" "WM"
[229] "wsrf"          "xgbLinear"     "xgbTree"
[232] "xyf"
```

과제 설명

A_train 을 decision tree로 훈련 시켜 B_test의 값을 예측해주세요. (하이퍼 파라미터 튜닝 해야합니다)

(데이터 : 암인지 아닌지 구분하는 피쳐, y값(diagnosis)이 1이면 암,0이면 아니다.)



Submission 은 다음주 월요일 6시까지 제출 바람!

총 3번의 Submission 제출 가능(바로 정확도 계산하여 보내드릴게요!) -> submissio이름: 조이름_x번째.csv

Accuracy baseline : 0.935 (baseline 못넘을 경우 과제 추가)

QUESTION