



11주차: 분류를 위한 모델

ChulSoo Park

School of Computer Engineering & Information Technology
Korea National University of Transportation

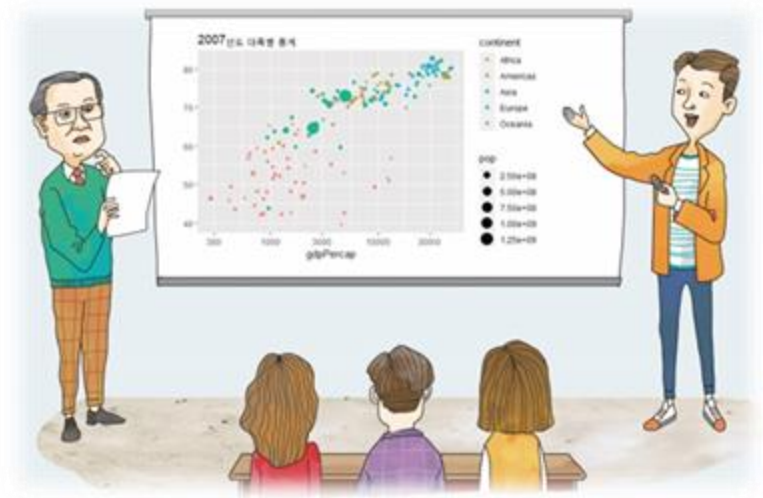
학습목표 (11주차)

- ❖ 회귀와 분류 문제의 이해
- ❖ 결정 트리의 이해
- ❖ 랜덤 포리스트 개념 이해
- ❖ SVM과 k-NN의 이해
- ❖ 분류 모델의 적용 학습

09

CHAPTER

분류를 위한 모델



CONTENTS

- 9.1 회귀와 분류
- 9.2 결정 트리의 원리
- 9.3 결정 트리 함수의 사용
- 9.4 결정 트리의 해석
- 9.5 랜덤 포리스트**
- 9.6 SVM과 k-NN**
- 9.7 분류 모델의 다양한 적용**
- 요약

9.5 랜덤 포리스트 (Random Forest)

랜덤 포레스트는 여러 개의 결정 트리를 결합하여 성능을 향상하는 앙상블 (ensemble) 기법, 각각의 분류기(결정 트리)를 요소 분류기(component classifier)라 부름(아래의 그림에서는 t 개의 요소 분류기로 구성)

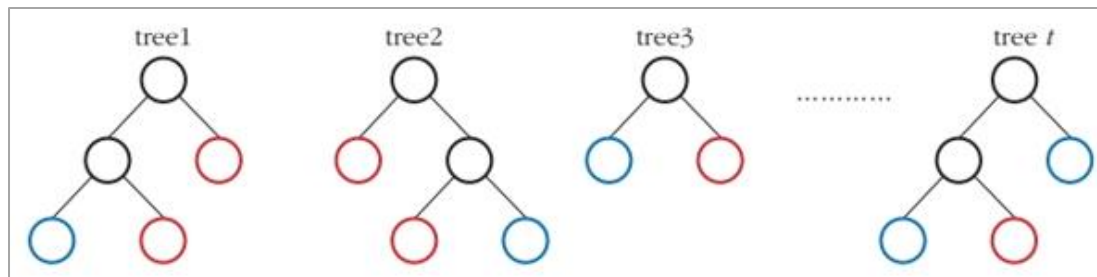
Decision Tree는 overfitting될 가능성이 높다는 약점을 가지고 있음.

가지치기를 통해 트리의 최대 높이를 설정해 줄 수 있음

이것만으로 overfitting을 충분히 해결할 수 없음

좀더 일반화된 트리를 만드는 방법을 생각해야함

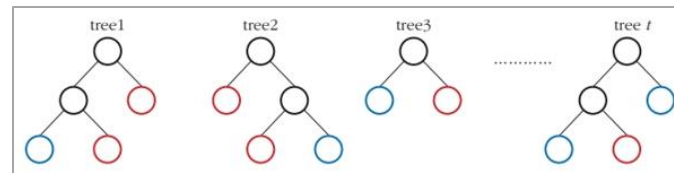
이런 아이디어가 랜덤 포리스트(Random Forest)의 기원임.



9.5 랜덤 포리스트 (Random Forest)

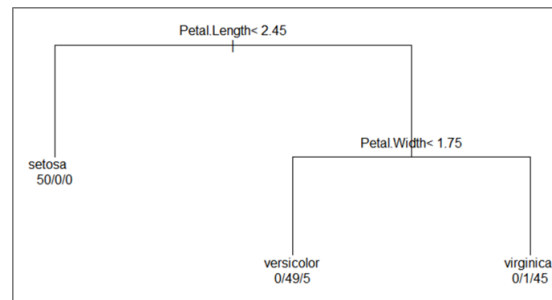
■ 랜덤 포리스트

- 나무를 여러 개 사용하므로 '포리스트'(숲)
- 결정 트리를 만들 때 난수를 사용하기 때문에 '랜덤'



■ 요소 분류기 만들기

- 주로 약한 분류기(weak classifier)를 사용(기본 성능보다 약간 높은 성능의 분류기)
- 서로 독립일수록 결합 성능이 좋으므로 난수를 사용하여 독립성 확보
 - ✓ 예) 결정 트리에서는 가장 좋은 질문을 노드에 부여하는 반면, 랜덤 포리스트에서는 가장 좋은 k 개의 후보 질문 중에서 랜덤하게 선택
- 따라서 랜덤 포리스트를 만들 때마다 다른 성능



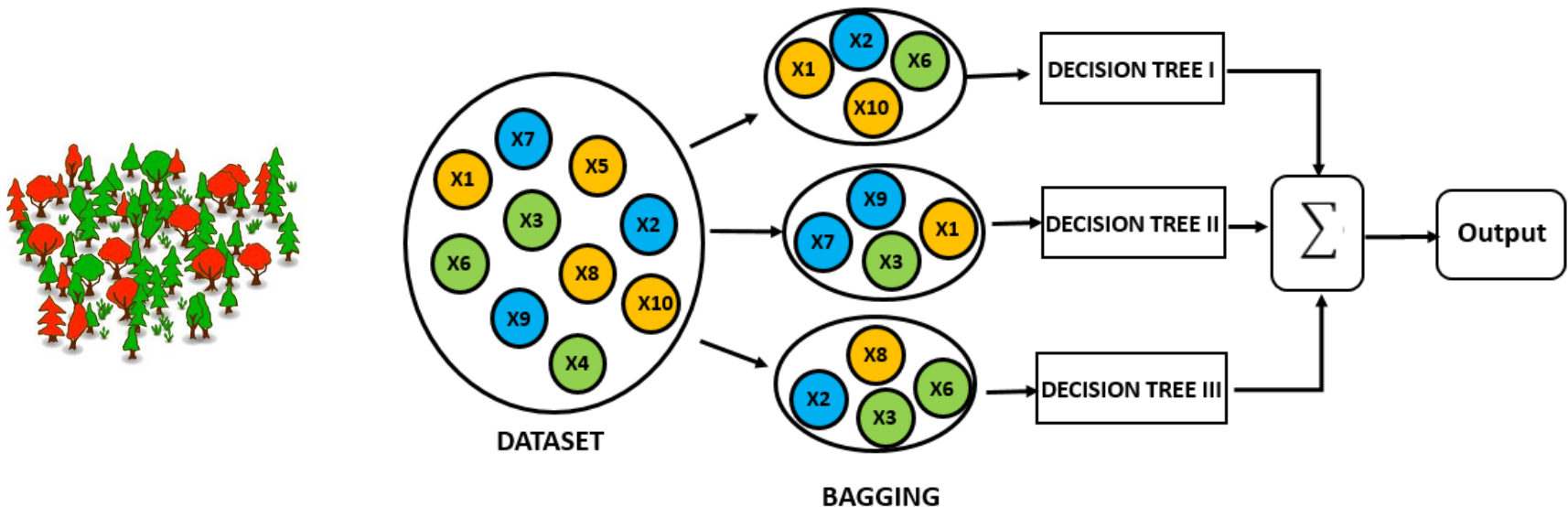
9.5 랜덤 포리스트 (Random Forest)

■ 예측 단계

- 새로운 샘플을 각각의 결정 트리에 입력하여 분류 결과를 얻음
- 투표(voting) 기법으로 결합
- 예) iris의 경우
 - 5개의 결정 트리가 setosa, versicolor, setosa, setosa, versicolor로 분류했다면 setosa가 3표를 얻어 최다 득표이므로 최종적으로 setosa로 분류함
- 결정 트리마다 성능을 평가한 다음 가중치를 부여하고 가중 투표(weighted voting) 기법을 쓰기도 함 (잘하는 요소 분류기에 표를 더 주는 방식)

9.5 랜덤 포리스트 (Random Forest)

As we see, there is multiple decision trees as base learners. Each decision tree is given a subset of random samples from the data set (Thus, the name 'Random'). The Random Forest algorithm uses Bagging (Bootstrap Aggregating) which we learned in ensemble methods. The general idea of the ensemble methods is that a combination of learning models increases the overall result.



9.5 랜덤 포리스트 (Random Forest)

data 구조 및 data 확인

Console

Jobs x

C:/RSources/ ↗

```
> str(iris)
```

```
'data.frame': 150 obs. of 5 variables:
```

```
$ Sepal.Length: num 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
```

```
$ Sepal.Width : num 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
```

```
$ Petal.Length: num 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
```

```
$ Petal.Width : num 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
```

```
$ Species : Factor w/ 3 levels "setosa","versicolor",...: 1 1 1 1 1 1 1 1 1 1
```

```
> head(iris,10)
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa
7	4.6	3.4	1.4	0.3	setosa
8	5.0	3.4	1.5	0.2	setosa
9	4.4	2.9	1.4	0.2	setosa
10	4.9	3.1	1.5	0.1	setosa

9.5 랜덤 포리스트 (Random Forest)

■ iris 데이터에 random Forest 함수 적용하기

■ 랜덤 포리스트 해석

- Number of trees: 500
- error rate: 4%
- 훈련 집합에 대한 혼동 행렬을 보면 6개 샘플을 잘못 분류함
→ 정확률은 $144/150=96\%$

```
Console C:/RSources/
> dt_iris = randomForest(Species~., data=iris)
> # 학습된 결정트리 정보
> dt_iris

Call:
randomForest(formula = Species ~ ., data = iris)
Type of random forest: classification
Number of trees: 500
No. of variables tried at each split: 2

OOB estimate of error rate: 4%
Confusion matrix:
      setosa versicolor virginica class.error
setosa    50         0         0      0.00
versicolor  0        47         3      0.06
virginica   0         3        47      0.06
> # 학습된 결정트리 요약
> summary(dt_iris)

      Length Class Mode
call           3  -none- call
type            1  -none- character
predicted       150 factor numeric
err.rate        2000 -none- numeric
confusion        12  -none- numeric
votes           450 matrix numeric
oob.times        150 -none- numeric
classes           3  -none- character
importance         4  -none- numeric
importanceSD        0  -none- NULL
localImportance     0  -none- NULL
```

9.5 랜덤 포리스트 (Random Forest)

■ dt_iris data 구조

- 19개의 list로 구성

```
Console C:/RSources/
> str(dt_iris)
List of 19
 $ call           : language randomForest(formula = Species ~ ., data = iris)
 $ type           : chr "classification"
 $ predicted      : Factor w/ 3 levels "setosa","versicolor",...: 1 1 1 1 1 1 1 1 1 1 ...
 ..- attr(*, "names")= chr [1:150] "1" "2" "3" "4" ...
 $ err.rate       : num [1:500, 1:4] 0.0328 0.0435 0.0427 0.0677 0.0638 ...
 ..- attr(*, "dimnames")=List of 2
 .. ..$ : NULL
 .. ..$ : chr [1:4] "OOB" "setosa" "versicolor" "virginica"
 $ confusion      : num [1:3, 1:4] 50 0 0 0 47 3 0 3 47 0 ...
 ..- attr(*, "dimnames")=List of 2
 .. ..$ : chr [1:3] "setosa" "versicolor" "virginica"
 .. ..$ : chr [1:4] "setosa" "versicolor" "virginica" "class.error"
 $ votes          : 'matrix' num [1:150, 1:3] 1 1 1 1 1 1 1 1 1 1 ...
 ..- attr(*, "dimnames")=List of 2
 .. ..$ : chr [1:150] "1" "2" "3" "4" ...
 .. ..$ : chr [1:3] "setosa" "versicolor" "virginica"
 $ oob.times      : num [1:150] 188 187 172 183 188 185 183 178 171 197 ...
```

9.5 랜덤 포리스트 (Random Forest)

■ dt_iris data 살펴보기

```
> head(dt_iris,10)
```

```
$call
```

```
randomForest(formula = Species ~ ., data = iris)
```

```
$type
```

```
[1] "classification"
```

```
$predicted
```

1	2	3	4	5	6	7	8	9
setosa	setosa	setosa	setosa	setosa	setosa	setosa	setosa	setosa
10	11	12	13	14	15	16	17	18
setosa	setosa	setosa	setosa	setosa	setosa	setosa	setosa	setosa

```
$err.rate
```

	OOB	setosa	versicolor	virginica
[1,]	0.10204082	0	0.15789474	0.11111111
[2,]	0.09302326	0	0.10000000	0.16129032
[3,]	0.07894737	0	0.08108108	0.14634146

```
[250,] 0.04666667 0 0.06000000 0.08000000
```

```
[ getOption("max.print") 에 도달했습니다 -- 250 행들을 생략합니다 ]
```

```
$confusion
```

	setosa	versicolor	virginica	class.error
setosa	50	0	0	0.00
versicolor	0	47	3	0.06
virginica	0	4	46	0.08

9.5 랜덤 포리스트 (Random Forest)

■ dt_iris data 살펴보기

```
$votes
      setosa versicolor virginica
1  1.000000000 0.000000000 0.000000000
2  1.000000000 0.000000000 0.000000000
78 0.000000000 0.240641711 0.759358289
79 0.000000000 0.994219653 0.005780347
```

```
$classes
[1] "setosa"      "versicolor" "virginica"
```

```
$importance
      MeanDecreaseGini
Sepal.Length      8.739583
Sepal.Width       1.946064
Petal.Length     45.071353
Petal.Width      43.508172
```

```
$err.rate
      OOB setosa versicolor virginica
[1,] 0.10204082      0 0.15789474 0.11111111
[2,] 0.09302326      0 0.10000000 0.16129032
[3,] 0.07894737      0 0.08108108 0.14634146
```

OOB (Out-of-bag) 오류는 각 부트스트랩 샘플에 포함되지 않은 트리의 예측을 사용하여 계산된 각 평균 오류입니다. 이를 통해 RandomForestClassifier가 훈련되는 동안 적합하고 유효성을 검사할 수 있습니다.

9.5 랜덤 포리스트 (Random Forest)

■ 랜덤 포리스트의 내부를 살펴보는 함수

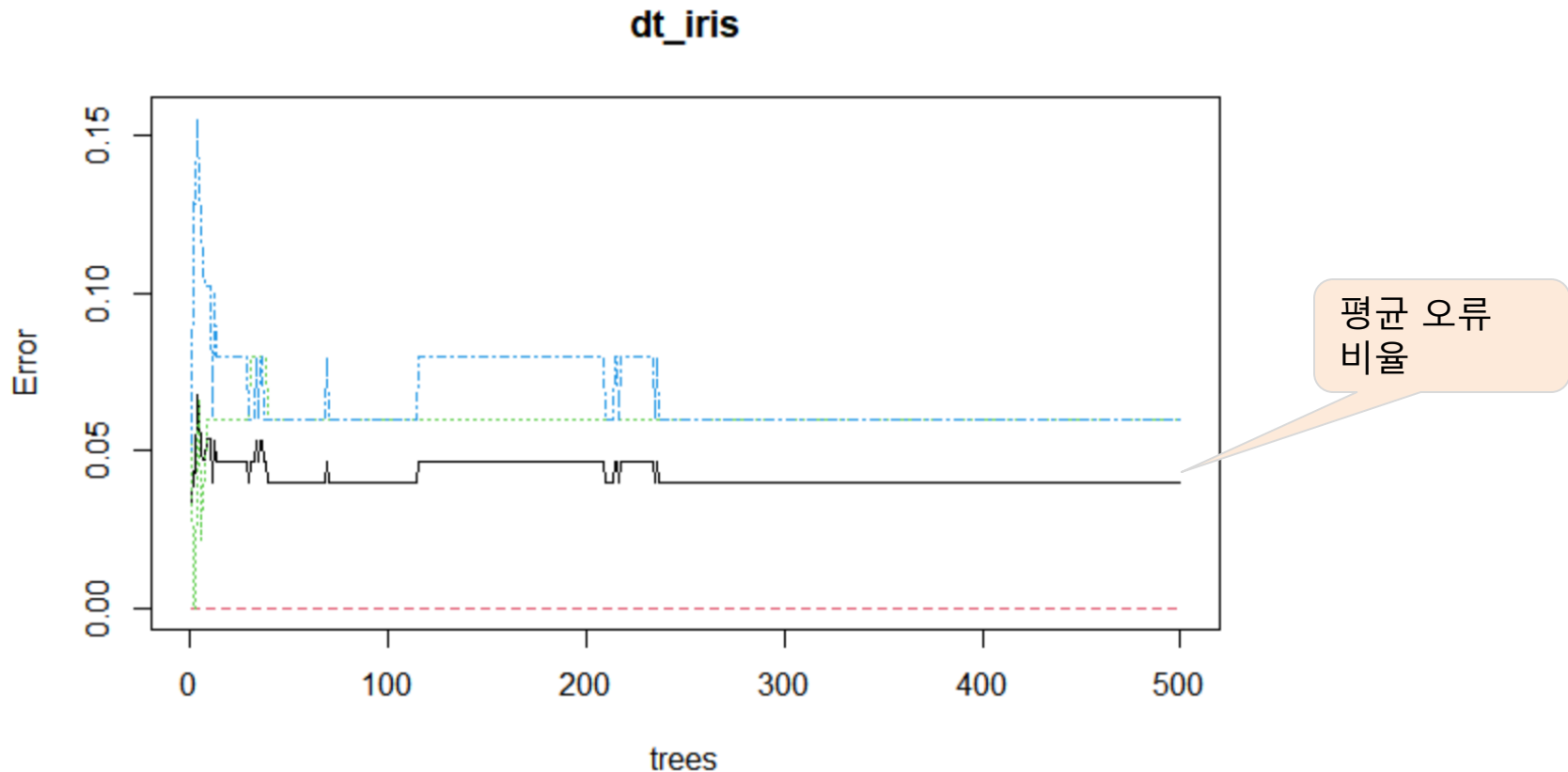
- `treecsize`는 결정 트리 각각의 리프 노드 개수를 알려줌
- 예) 1번, 2번, 499번, 500번 결정 트리는 10개, 9개, 7개, 9개의 리프 노드를 가짐
- 교재와 제가 한 것 학생 여러분의 것이 모두 다르게 나올 수 있음(이유 : **Random**)
- 매번 동일하게 하는 방법(유사 난수 사용 : `set.seed(1)`)

```
> treecsize(dt_iris)
[1] 10 9 14 9 10 7 9 9 7 14 8 6 4 7 6 4 6 11 10 6 8 9 8 11 12 10 7 10 9 6 11
[32] 8 14 6 15 7 6 7 10 7 9 9 11 13 9 5 11 14 11 8 11 11 8 8 11 8 8 7 5 10 6 14
[63] 9 8 10 5 6 12 12 5 9 5 10 8 11 7 9 9 12 8 7 12 9 6 5 4 4 8 9 9 9 6 7
[94] 10 9 5 11 8 8 9 7 10 7 12 12 10 8 9 8 7 7 7 18 8 4 11 11 6 11 7 11 7 5 11
[125] 8 7 7 7 7 7 5 7 10 7 11 8 9 10 8 9 11 8 10 9 10 13 9 13 9 6 8 7 13 8 7
[156] 9 11 8 6 9 10 7 12 6 10 6 8 11 8 10 9 10 7 12 9 6 10 6 14 10 10 7 8 5 11 9
[187] 17 9 8 9 11 10 8 12 13 12 10 7 5 4 9 7 7 3 8 9 14 5 7 9 10 8 8 9 7 5 7
[218] 9 10 11 6 7 9 8 7 6 7 9 12 8 9 9 8 9 10 7 6 9 10 8 9 8 10 9 4 7 6 10
[249] 9 9 4 10 9 10 6 11 10 11 8 6 8 9 11 6 9 9 5 8 9 10 9 7 10 9 7 11 10 10 7
[280] 11 11 13 10 7 8 10 12 7 11 8 11 8 8 9 5 9 7 12 13 13 10 9 5 9 5 9 7 8 10 6
[311] 8 7 12 10 14 9 11 4 13 10 11 11 13 5 7 11 8 7 10 8 9 11 4 11 8 8 5 9 10 7 10
[342] 8 10 11 7 13 9 6 10 8 8 9 8 5 11 12 8 10 12 7 10 9 12 10 6 10 9 7 8 13 13 12
[373] 8 11 9 10 9 10 8 12 7 6 7 11 9 9 10 7 11 9 11 7 6 6 11 12 11 9 8 9 11 7 10
[404] 9 15 11 9 9 8 5 7 9 13 10 9 10 11 7 6 7 11 9 6 5 14 5 10 8 8 13 10 6 6 11
[435] 7 12 8 4 10 9 7 7 9 7 12 12 9 13 10 10 10 7 7 13 6 8 8 8 9 7 9 8 10 7 13
[466] 8 6 9 11 12 8 5 8 10 5 10 10 3 11 6 6 8 10 5 9 7 9 11 7 9 6 8 8 7 12 6
[497] 10 8 7 9
```

9.5 랜덤 포리스트 (Random Forest)

■ 랜덤 포리스트 시각화

- 가로축은 결정 트리의 개수
- 세로축은 랜덤 포리스트의 오류율 (빨강, 파랑, 녹색은 각각 setosa, versicolor, virginica의 오류율. 검정은 평균 오류율)



9.5 랜덤 포리스트 (Random Forest)

■ 랜덤 포리스트의 내부를 살펴보는 함수

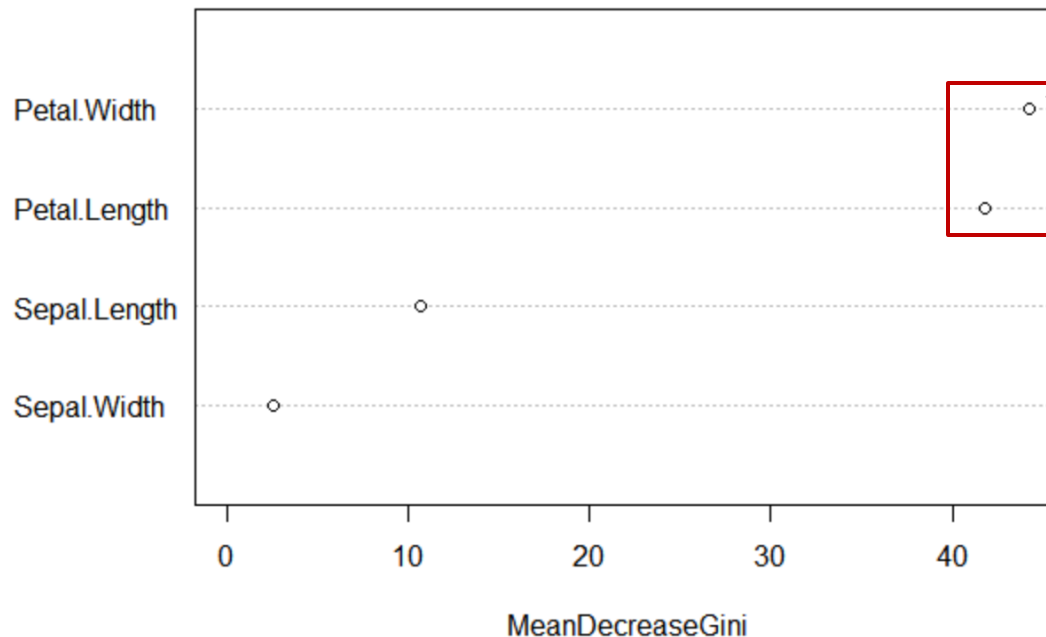
- varUsed는 설명 변수가 질문에 사용된 횟수를 알려줌
- varImpPlot은 설명 변수의 중요도를 그려줌

Console C:/RSources/ ↗

```
> varUsed(dt_iris)
[1] 798 571 1269 1244
> varImpPlot(dt_iris)
```

교재와 다름

dt_iris



40% 대 중요도

9.5 랜덤 포리스트 (Random Forest)

- 새로운 샘플에 대해 예측해 보기
 - 앞에서 (9장 3절) 사용한 가상의 3개 샘플을 재사용
 - 랜덤 포리스트는 결정 트리와 달리 부류 정보만 출력함

Console C:/RSources/ 

```
> newd = data.frame(Sepal.Length = c(5.11, 7.01, 6.32), Sepal.Width = c(3.51, 3.2, 3.31), Petal.Length = c(1.4, 4.71, 6.02), Petal.Width = c(0.19, 1.4, 2.49))
> predict(dt_iris, newdata = newd)
      1      2      3
setosa versicolor virginica
Levels: setosa versicolor virginica
```


9.5 랜덤 포리스트 (Random Forest)

- 새로운 샘플에 대해 예측해보기
 - type='prob' 옵션을 설정하면 확률 출력
 - (샘플은 setosa, versicolor, virginica에 속할 확률이
 - ✓ 첫번째 1.0, 0.000, 0.000
 - ✓ 두번째 0.0, 0.984, 0.016
 - ✓ 세번째 0.0, 0.000, 1.000임을 나타냄

```
> predict(dt_iris, newdata = newd, type = 'prob')
      setosa versicolor virginica
1         1      0.000      0.000
2         0      0.984      0.016
3         0      0.000      1.000
attr(,"class")
[1] "matrix" "array"  "votes"
```

9.5 랜덤 포리스트 (Random Forest)

■ 새로운 샘플에 대해 예측해보기

- 득표수 확인
- `type='vote', norm.votes=FALSE` 설정
- 두 번째 득표 현황 : setosa : 0표, versicolor 492표, virginica 8표를 받음

```
> predict(dt_iris, newd, type = 'vote', norm.votes = FALSE)
  setosa versicolor virginica
1    500          0         0
2      0        492         8
3      0          0        500
attr(,"class")
[1] "matrix" "array"  "votes"
```

9.5 랜덤 포리스트 (Random Forest)

■ 하이퍼 매개변수_{hyper parameter}

- 모델의 구조나 학습 방법을 제어하는 데 사용하는 변수
- 랜덤 포리스트의 하이퍼 매개변수
 - ✓ ntree: 결정 트리의 개수 (기본값은 500)
 - ✓ nodesize: 리프 노드에 도달한 샘플의 최소 개수 (크게 설정할수록 작은 결정 트리)
 - ✓ maxnodes: 리프 노드의 최대 개수

```
> small_forest=randomForest(Species~., data=iris, ntree=20, no
desize=6, maxnodes=12)
> treesize(small_forest)
[1] 9 6 5 6 6 7 6 8 6 7 7 5 5 6 8 7 8 8 6 9
> small_forest
```

Call:

```
randomForest(formula = Species ~ ., data = iris, ntree = 20,
  nodesize = 6, maxnodes = 12)
```

Type of random forest: classification

Number of trees: 20

No. of variables tried at each split: 2

OOB estimate of error rate: 5.33%

Confusion matrix:

	setosa	versicolor	virginica	class.error
setosa	50	0	0	0.00
versicolor	0	47	3	0.06
virginica	0	5	45	0.10

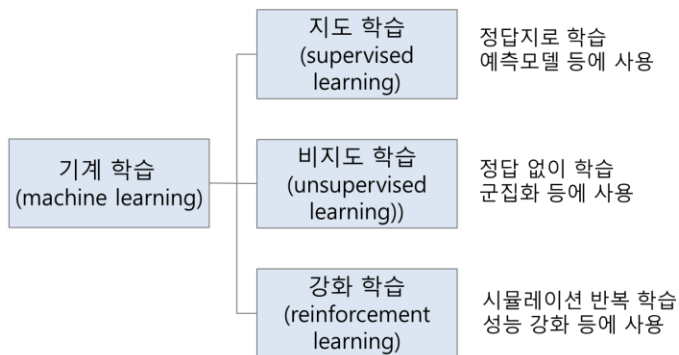
리프노드 수 : 5~9개

정확률 : 142/150=94.37%

9.6 SVM과 k-NN

■ 다양한 모델

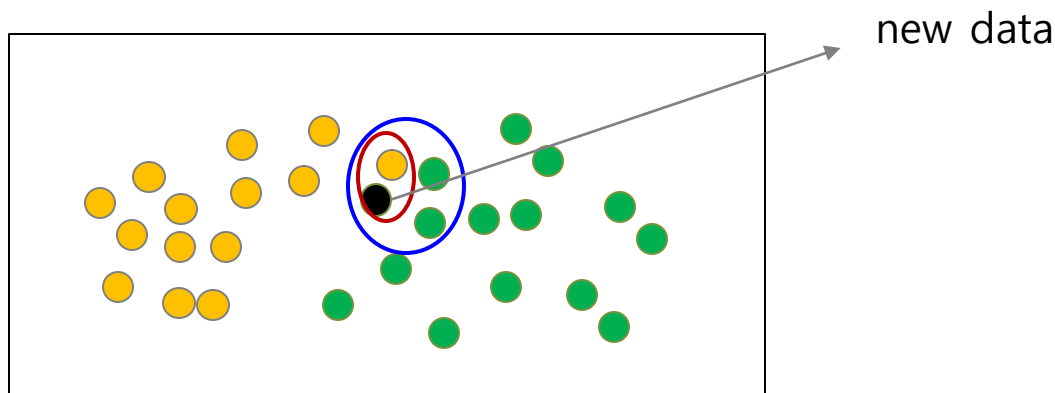
- 결정 트리, 랜덤 포리스트, SVM, k -NN, 신경망, 딥러닝 등
- 이 절에서는 SVM과 k -NN



구분	기계학습 유형		대표 알고리즘
비지도 학습 (unsupervised learning)	서술형 (descriptive)	클러스터링	k-means
		연관 분석	패턴 분석
지도 학습 (supervised learning)	예측형 (prediction)	분류 예측	k-NN, 베이어스, 의사결정 트리
		수치 예측	선형 회귀 분석, 회귀 트리, SVM

■ K-최근접이웃(k-NN, K-Nearest Neighbor)

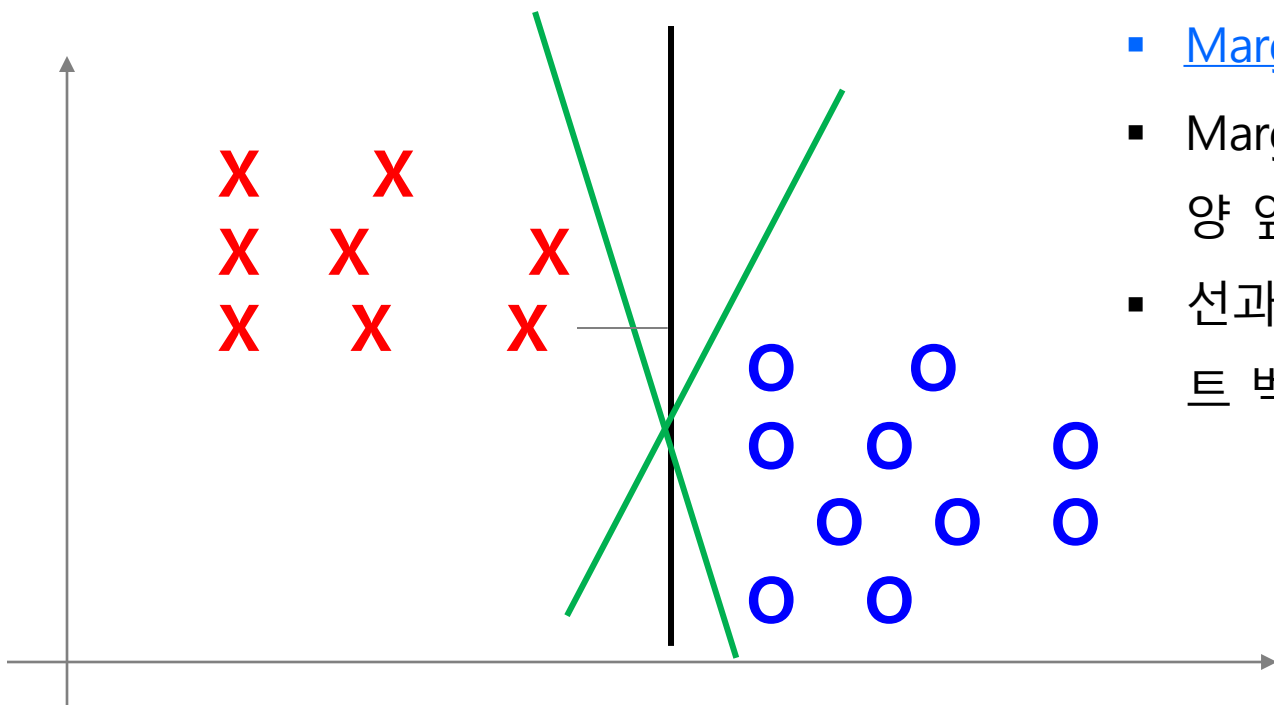
- k-NN은 새로운 데이터가 주어졌을 때 기존 데이터 가운데 가장 가까운 k 개 이웃의 정보로 새로운 데이터를 예측하는 방법론.
- 아래 그림처럼 검은색 점의 범주 정보는 주변 이웃들을 가지고 추론해낼 수 있음.
- 만약 k가 1이라면 오렌지색, k가 3이라면 녹색으로 분류(classification)하는 것.
- 만약 회귀(regression) 문제라면 이웃들 종속변수(y)의 평균이 예측값이 됨.



9.6 SVM과 k-NN

■ 서포트 벡터 머신(SVM, Support Vector Machine)

서포트 벡터 머신(SVM, Support Vector Machine)이란 주어진 데이터가 어느 카테고리에 속할지 판단하는 **이진 선형 분류 모델**입니다.

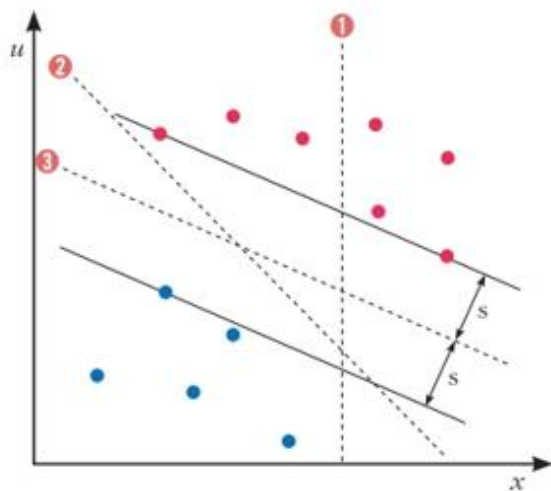


- Margin을 최대화
- Margin이란 선과 가장 가까운 양 옆 데이터와의 거리
- 선과 가장 가까운 포인트를 서포트 벡터(Support vector)

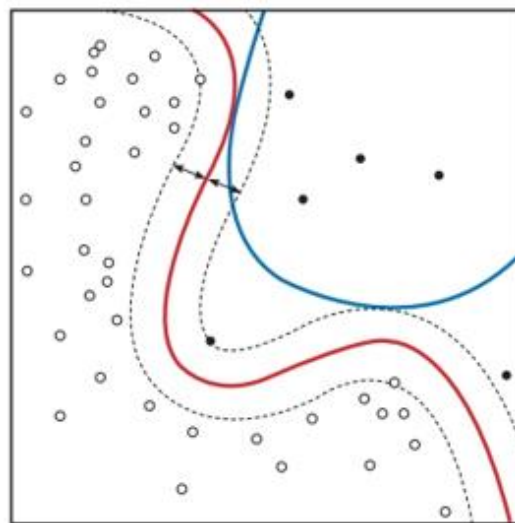
9.6 SVM과 k-NN

■ SVM의 원리

- 모델 ①은 빨간 샘플 3개를 잘못 분류 → 오류율 $3/12=25\%$
- ②와 ③은 오류율 0% ← 둘의 성능은 같은가? SVM의 원리는 이 질문에서 출발
- ②는 빨간 샘플에 조금만 변형이 발생해도 경계를 넘어 오류 발생할 가능성 높음
- ③은 두 부류 모두에 대해 멀리 떨어져 있어 변형이 발생해도 경계를 넘을 가능성 낮음
- 일반화 측면에서 모델 ③이 더 뛰어남. SVM 학습은 여백(margin) [아래 그림]에서 $2s$ 을 최대로 하는 최적 모델을 찾아 줌



(a) 선형 SVM



(b) 비선형 SVM

9.6 SVM과 k-NN

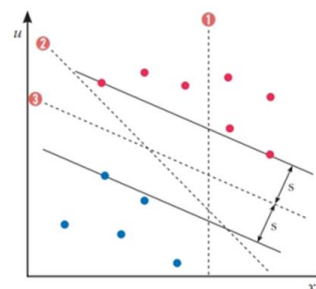
■ 비선형 SVM

- [그림 (a)]의 SVM은 선형 분류기(linear classifier)임
- [그림 (b)]처럼 커널 트릭(kernel trick)을 이용하여 비선형 분류기(nonlinear classifier)로 확장
- 주로 사용되는 커널 함수: polynomial, [radial basis function](#), sigmoid 세 종류

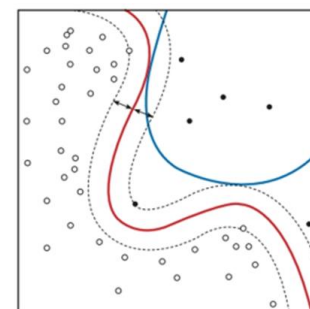
```
svm(formula, data = NULL, ..., subset, na.action =
na.omit, scale = TRUE)
## Default S3 method:
svm(x, y = NULL, scale = TRUE, type = NULL, kernel =
"radial", degree = 3, gamma = if (is.vector(x)) 1 else 1 / n
coef0 = 0, cost = 1, nu = 0.5,
class.weights = NULL, cachesize = 40, tolerance = 0.001, eps.
shrinking = TRUE, cross = 0, probability = FALSE, fitted = T
..., subset, na.action = na.omit)
```

```
> head(iris,5)
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa



(a) 선형 SVM



(b) 비선형 SVM

9.6 SVM과 k-NN

■ SVM을 지원하는 e1071 라이브러리

Console C:/Rsources/ ↗

```
> library(e1071)
> svm_iris = svm(Species~., data = iris)    # 기본 커널은 radial basis 사용
> print(svm_iris)
```

Call:

```
svm(formula = species ~ ., data = iris)
```

Parameters:

SVM-Type: C-classification

SVM-Kernel: radial

cost: 1

Number of Support Vectors: 51

```
> table(predict(s, iris), iris$Species)
```

	setosa	versicolor	virginica
setosa	50	0	0
versicolor	0	48	2
virginica	0	2	48

9.6 SVM과 k-NN

■ polynomial 커널 사용하는 코드

- radial basis function의 오류율 : $4/150=2.667\%$
- polynomial 오류율 : $7/150=4.666\%$
- 기본값인 radial basis function 오류율(2.667%)보다 polynomial 오류율(4.666%)의 증가로 성능은 낮음

Console C:/RSources/ ↗

```
> svm_iris = svm(Species~., data = iris, kernel = 'polynomial') # polynomial 커널 사용  
> p = predict(svm_iris, iris)  
> table(p, iris$Species)
```

p	setosa	versicolor	virginica
setosa	50	0	0
versicolor	0	50	7
virginica	0	0	43

9.6 SVM과 k-NN

■ C라는 하이퍼 매개변수

- 실제 데이터에서는 오류를 허용하는 수밖에 없음
- 오류와 여백은 트레이드오프 관계: 오류를 줄이면 여백도 줄어 일반화 능력이 약해짐
- C를 크게 설정하면, 훈련 집합에 대해 오류율이 줄어듦
 - 예) 아래 코드는 $\text{cost}=100$ 으로 설정 \rightarrow 오류율 $= 2/150 = 1.333\%$ (R에서는 cost 매개변수가 C역할)
- C를 크게 설정하면, 훈련 집합에 대해 오류율이 줄지만 일반화 능력 떨어짐 \rightarrow 당장은 기분이 좋지만 새로운 데이터에 대한 낮은 성능. 적절한 값 설정이 중요

Console C:/RSources/ ↗

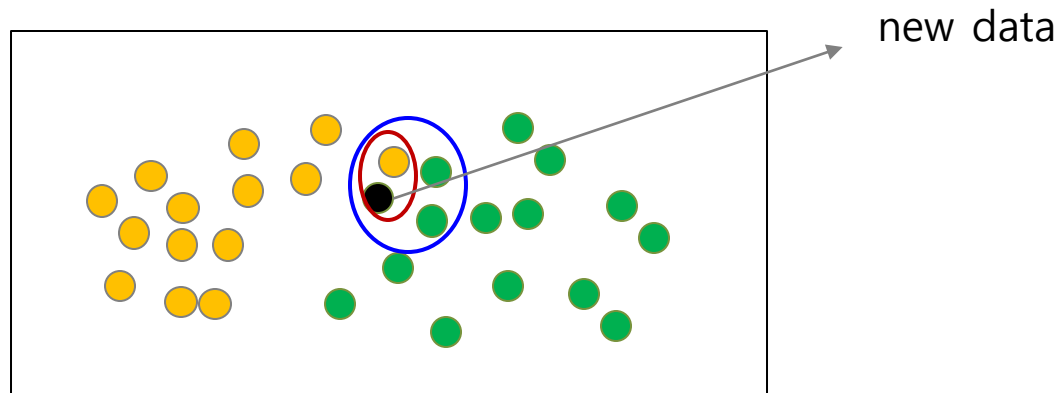
```
> svm_iris = svm(Species~., data = iris, cost = 100) # 기본으로 실행 radial basis
> p = predict(svm_iris, iris)
> table(p, iris$Species)
```

p	setosa	versicolor	virginica
setosa	50	0	0
versicolor	0	49	1
virginica	0	1	49

9.6 SVM과 k-NN

■ K-부류 분류기로 확장

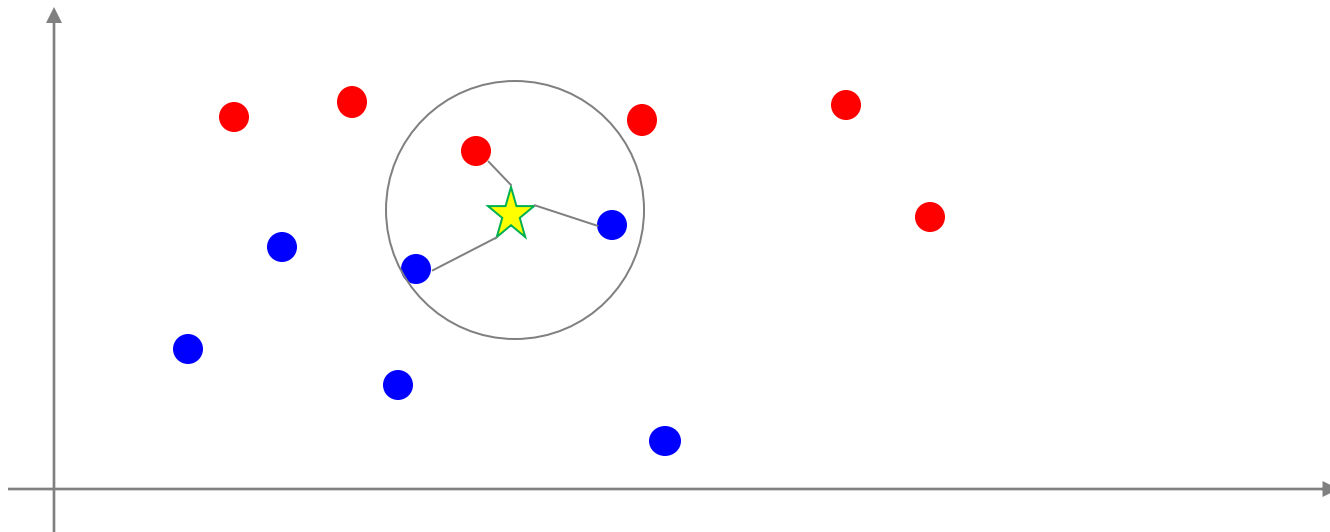
- SVM은 두 부류를 분류하는 이진 분류 모델
- SVM을 K 개 만들어 K -부류 분류기로 확장
- 예) 숫자 인식의 경우, 0과 1,2,...,9를 분류하는 SVM, 1과 0,2,...,9를 분류하는 SVM 등을 10개 만들어 10-부류 분류기로 확장함



9.6 SVM과 k-NN

■ k-nn의 원리

- 새로운 테스트 샘플([그림 9-10]에서 세모 모양 샘플)이 입력되면 훈련 집합에서 가장 가까운 k 개 샘플을 찾고 발생 빈도가 가장 높은 부류로 분류
- 훈련 집합을 메모리에 저장해 두어야 하므로, k -NN은 메모리 기반 모델
- 만약 k 가 1이라면 빨간색, k 가 3이라면 청색으로 분류(classification)하는 것.
- 만약 회귀(regression) 문제라면 이웃들 종속변수(y)의 평균이 예측값이 됨



9.6 SVM과 k-NN

■ knn 함수를 이용한 분류 예제

■ knn 함수 사용 방법

- 첫 번째 매개변수: (레이블 정보를 제외한) 훈련 집합
- 두 번째 매개변수: 테스트 샘플
- 세 번째 매개변수: 레이블 정보
- 네 번째 매개변수: k 값 (아래 코드는 5-NN)

Console C:/RSources/ ↗

```
> library(class)
> train = iris
> test = data.frame(Sepal.Length = c(5.11, 7.01, 6.32), Sepal.Width = c(3.51, 3.2, 3.3
1), Petal.Length = c(1.4, 4.71, 6.02), Petal.Width = c(0.19, 1.4, 2.49))
> k = knn(train[, 1:4], test, train$Species, k = 5)
> k
[1] setosa      versicolor virginica
Levels: setosa versicolor virginica
```

```
> head(train)
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa

9.6 SVM과 k-NN

- train 함수를 사용하면 여러 모델을 일관성있게 프로그래밍 가능
 - train 함수는 caret 라이브러리가 제공
 - `install.packages("caret")`
 - `names((getModelInfo()))`를 수행하면 train 함수가 제공하는 모든 모델을 알 수 있음

Console C:/RSources/ ↗

```
> library(caret)
> r = train(Species~., data = iris, method = 'rpart')
> f = train(Species~., data = iris, method = 'rf')
> s = train(Species~., data = iris, method = 'svmRadial')
> k = train(Species~., data = iris, method = 'knn')
> names(getModelInfo())
[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"
```

9.7 분류 모델의 다양한 적용

■ 여러 데이터에 분류 모델을 적용하는 연습

- UCLA admission 데이터
- colon 데이터
- voice 데이터

■ UCLA admission 데이터를 읽고 내용 확인

```
> # 07 UCLA admission data #
> ucla = read.csv('https://stats.idre.ucla.edu/stat/data/binary.csv')
> str(ucla)
'data.frame': 400 obs. of 4 variables:
 $ admit: int 0 1 1 1 0 1 1 0 1 0 ...
 $ gre : int 380 660 800 640 520 760 560 400 540 700 ...
 $ gpa : num 3.61 3.67 4 3.19 2.93 3 2.98 3.08 3.39 3.92 ...
 $ rank : int 3 3 1 4 4 2 1 2 3 2 ...
> ucla$admit = factor(ucla$admit)
> head(ucla, 5)
  admit gre  gpa rank
1     0 380 3.61   3
2     1 660 3.67   3
3     1 800 4.00   1
4     1 640 3.19   4
5     0 520 2.93   4
> str(ucla)
'data.frame': 400 obs. of 4 variables:
 $ admit: Factor w/ 2 levels "0"."1": 1 2 2 2 1 2 2 1 2 1 ...
 $ gre : int 380 660 800 640 520 760 560 400 540 700 ...
 $ gpa : num 3.61 3.67 4 3.19 2.93 3 2.98 3.08 3.39 3.92 ...
 $ rank : int 3 3 1 4 4 2 1 2 3 2 ...
```

반응 변수 admit를 factor 형으로 변환
(factor가 아니면 rpart와 randomForest가
분류 대신에 회귀로 작동함)

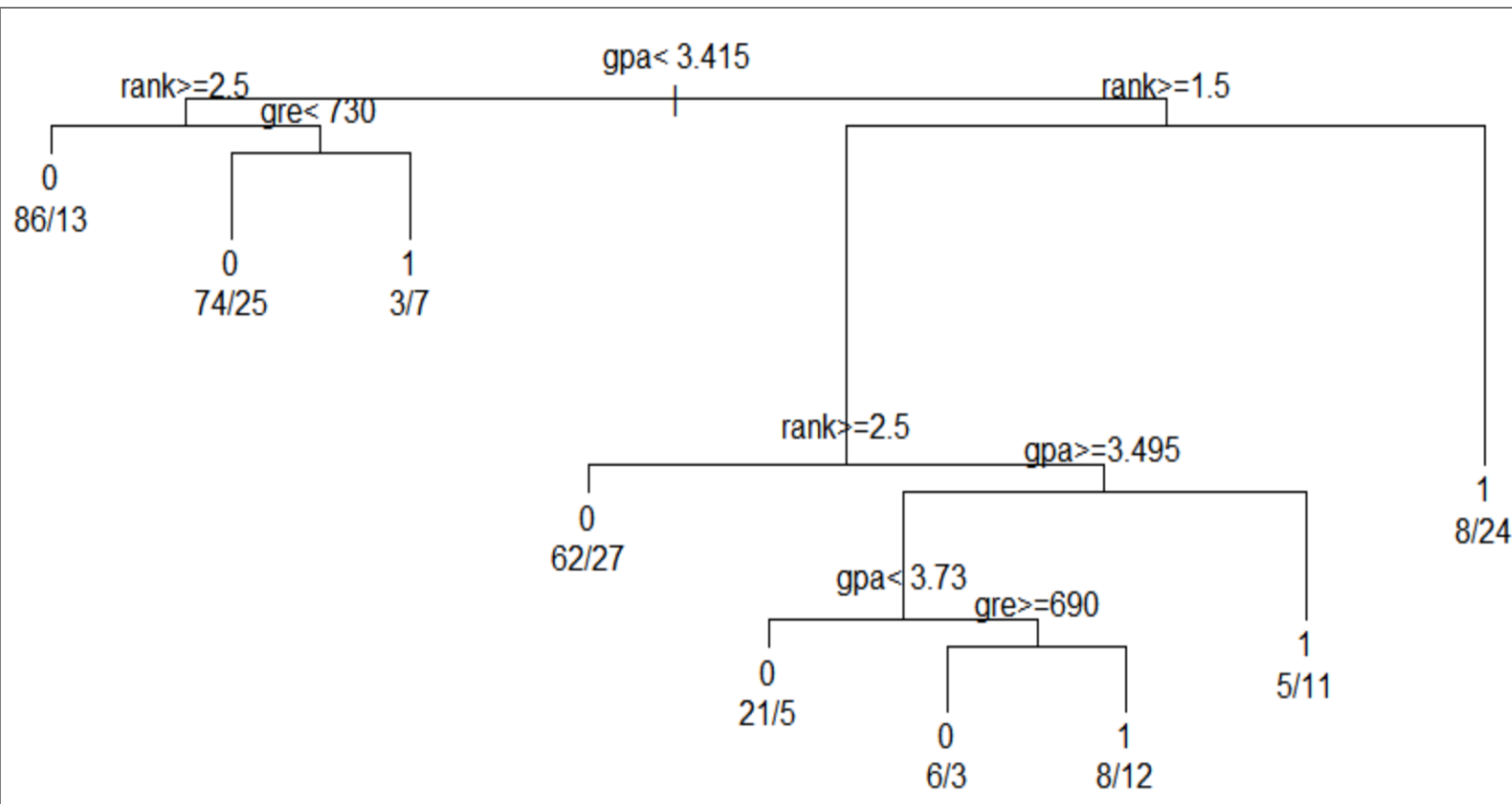
합격, 불합격

9.7 분류 모델의 다양한 적용

■ rpart 적용

Console C:/RSources/ ➔

```
> library(rpart)
> rpart_ucla = rpart(admit~., data = ucla)
> par(mfrow = c(1, 1), xpd = NA)
> plot(rpart_ucla)
> text(rpart_ucla, use.n = TRUE)
```



9.7 분류 모델의 다양한 적용

■ 훈련 집합에 대한 예측

- 정확률 :
- $(249+54)/400=80.5\%$

■ randomForest 적용

$$(255+31)/400=71.5\%$$

Console C:/RSources/ ↗

```
> p_ucla = predict(rpart_ucla, ucla, type = 'class')  
> table(p_ucla, ucla$admit)
```

```
p_ucla    0    1  
      0 249   73  
      1  24   54
```

```
> random_ucla = randomForest(admit~., data = ucla)  
> print(random_ucla)
```

call:

```
randomForest(formula = admit ~ ., data = ucla)  
      Type of random forest: classification  
      Number of trees: 500
```

No. of variables tried at each split: 1

OOB estimate of error rate: 28.5%

Confusion matrix:

```
      0  1 class.error  
0 255 18  0.06593407  
1  96 31  0.75590551
```

9.7 분류 모델의 다양한 적용

- colon 데이터에 전처리 수행 (8장 5절 참조)
 - 결측값 제거, 반응 변수 status를 factor로 변환, 홀수 번째 샘플만 취함

```

Console C:/RSources/
> library(survival)
> clean_colon = na.omit(colon) # 전처리 : 결측치 제거
> clean_colon = clean_colon[c(TRUE, FALSE), ] # 전처리 : 홀수 번째 자료만 취함
> clean_colon$status = factor(clean_colon$status)
> str(clean_colon)
'data.frame': 888 obs. of 16 variables:
 $ id      : num  1 2 3 4 5 6 7 8 9 10 ...
 $ study   : num  1 1 1 1 1 1 1 1 1 1 ...
 $ rx      : Factor w/ 3 levels "Obs","Lev","Lev+5FU": 3 3 1 3 1 3 2 1 2 3 ...
 $ sex     : num  1 1 0 0 1 0 1 1 1 0 ...
 $ age     : num  43 63 71 66 69 57 77 54 46 68 ...
 $ obstruct: num  0 0 0 1 0 0 0 0 0 0 ...
 $ perfor  : num  0 0 0 0 0 0 0 0 0 0 ...
 $ adhere  : num  0 0 1 0 0 0 0 0 1 0 ...
 $ nodes   : num  5 1 7 6 22 9 5 1 2 1 ...
 $ status  : Factor w/ 2 levels "0","1": 2 1 2 2 2 2 2 1 1 1 ...
 $ differ  : num  2 2 2 2 2 2 2 2 2 2 ...
 $ extent  : num  3 3 2 3 3 3 3 3 3 3 ...
 $ surg    : num  0 0 0 1 1 0 1 0 0 1 ...
 $ node4   : num  1 0 1 1 1 1 1 0 0 0 ...
 $ time    : num  1521 3087 963 293 659 ...
 $ etype   : num  2 2 2 2 2 2 2 2 2 2 ...
 - attr(*, "na.action")= 'omit' Named int [1:82] 127 128 165 166 179 180 187 188 197 198 ...
 .- attr(*, "names")= chr [1:82] "127" "128" "165" "166" ...

```

9.7 분류 모델의 다양한 적용

■ rpart 적용

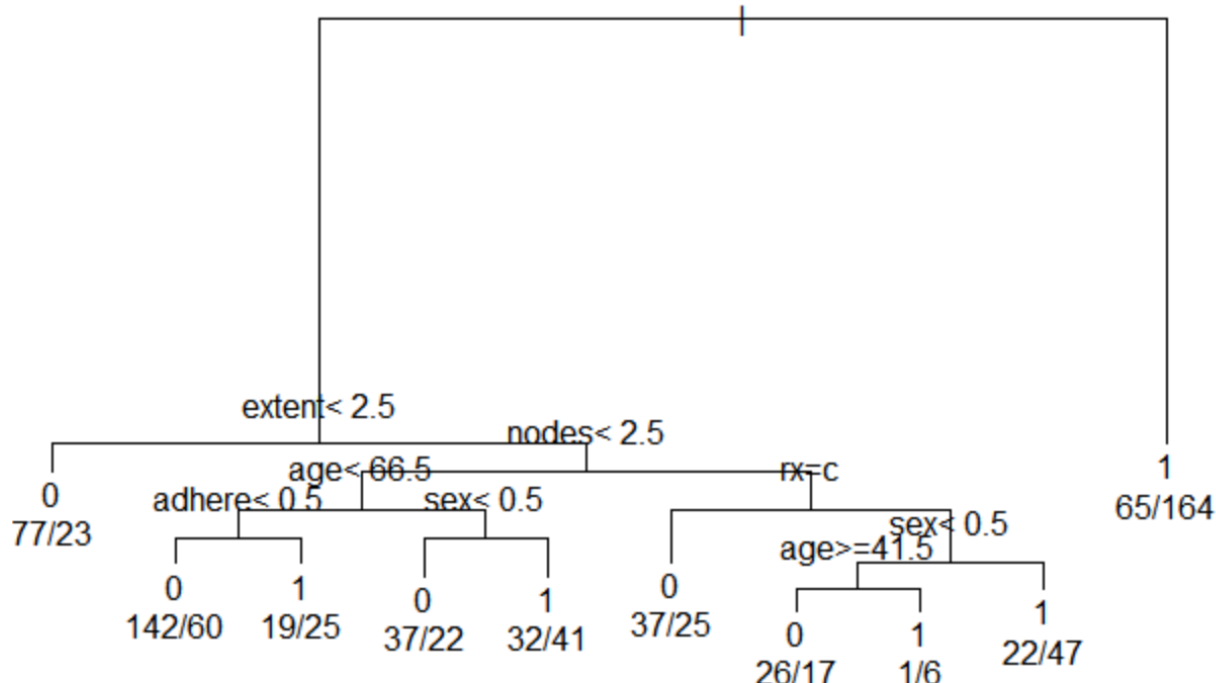
- 11개 변수만 설명 변수로 사용
- 정확률은 $(319+283)/888=67.793\%$

Console C:/RSources/

```
> r = rpart(status~rx + sex + age + obstruct + p
erfor + adhere + nodes + differ + extent + surg
+ node4, data = clean_colon)
> p = predict(r, clean_colon, type = 'class')
> table(p, clean_colon$status)
```

```
p      0      1
0 319 147
1 139 283
```




```
>
> plot(r)
> text(r, use.n = TRUE)
nodes< 4.5
```



9.7 분류 모델의 다양한 적용

■ randomForest 적용

- 정확률은 $(295+266)/888=63.18\%$

```
Console C:/RSources/     
> f = randomForest(status~rx + sex + age + obstruct + perfor + adhere +  
  nodes + differ + extent + surg + node4, data = clean_colon)  
> print(f)  
  
call:  
  randomForest(formula = status ~ rx + sex + age + obstruct + perfor +  
    adhere + nodes + differ + extent + surg + node4, data = clean_colon)  
    Type of random forest: classification  
    Number of trees: 500  
No. of variables tried at each split: 3  
  
    OOB estimate of  error rate: 37.39%  
Confusion matrix:  
      0    1 class.error  
0 299 159    0.3471616  
1 173 257    0.4023256
```

9.7 분류 모델의 다양한 적용

■ voice 데이터

- 음성 신호를 듣고 여성인지 남성인지 알아내는데 사용하는 데이터
- 음성 신호에서 추출한 20개의 특징이 설명 변수. label은 성별을 나타내는 반응 변수
- is.na로 확인해 보면 결측치 없음

```
Console C:/RSources/
> voice = read.csv('C:/rdata/voice.csv')
> str(voice)
'data.frame': 3168 obs. of 21 variables:
 $ meanfreq: num 0.0598 0.066 0.0773 0.1512 0.1351 ...
 $ sd : num 0.0642 0.0673 0.0838 0.0721 0.0791 ...
 $ median : num 0.032 0.0402 0.0367 0.158 0.1247 ...
 $ Q25 : num 0.0151 0.0194 0.0087 0.0966 0.0787 ...
 $ Q75 : num 0.0902 0.0927 0.1319 0.208 0.206 ...
 $ IQR : num 0.0751 0.0733 0.1232 0.1114 0.1273 ...
 $ skew : num 12.86 22.42 30.76 1.23 1.1 ...
 $ kurt : num 274.4 634.61 1024.93 4.18 4.33 ...
 $ sp.ent : num 0.893 0.892 0.846 0.963 0.972 ...
 $ sfm : num 0.492 0.514 0.479 0.727 0.784 ...
 $ mode : num 0 0 0 0.0839 0.1043 ...
 $ centroid: num 0.0598 0.066 0.0773 0.1512 0.1351 ...
 $ meanfun : num 0.0843 0.1079 0.0987 0.089 0.1064 ...
 $ minfun : num 0.0157 0.0158 0.0157 0.0178 0.0169 ...
 $ maxfun : num 0.276 0.25 0.271 0.25 0.267 ...
 $ meandom : num 0.00781 0.00901 0.00799 0.2015 0.71281 ...
 $ mindom : num 0.00781 0.00781 0.00781 0.00781 0.00781 ...
 $ maxdom : num 0.00781 0.05469 0.01562 0.5625 5.48438 ...
 $ dfrange : num 0 0.04688 0.00781 0.55469 5.47656 ...
 $ modindx : num 0 0.0526 0.0465 0.2471 0.2083 ...
 $ label : chr "male" "male" "male" "male" ...
>
> table(is.na(voice))

FALSE
66528
```

9.7 분류 모델의 다양한 적용

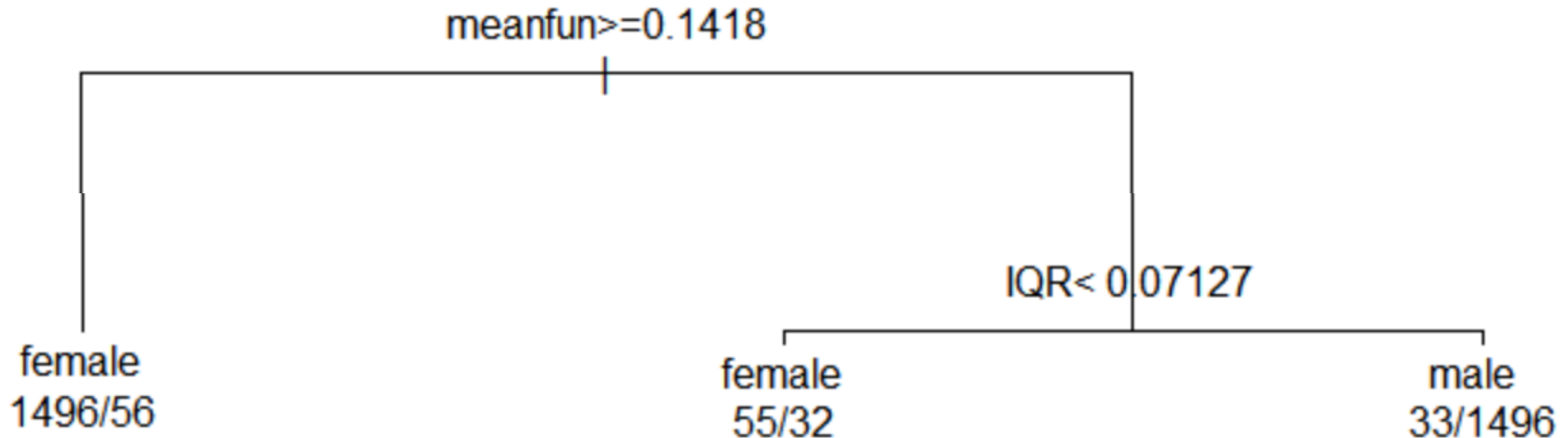
■ rpart 적용

- 정확률은 $(1551+1496)/3168=96.18\%$

Console C:/RSources/ ↗

```
> r = rpart(label~., data = voice)
> par(mfrow = c(1, 1), xpd = NA)
> plot(r)
> text(r, use.n = TRUE)
>
> p = predict(r, voice, type = 'class')
> table(p, voice$label)
```

p	female	male
female	1551	88
male	33	1496



9.7 분류 모델의 다양한 적용

■ randomForest 적용

```
> f = randomForest(label~., data = voice)
```

Error in y - ymean : 이항연산자에 수치가 아닌 인수입니다

추가정보: 경고메시지(들):

1: In randomForest.default(m, y, ...) :

The response has five or fewer unique values. Are you sure you want to do regression?

2: In mean.default(y) : argument is not numeric or logical: returning NA

```
> str(voice)
```

```
'data.frame': 3168 obs. of 21 variables:
 $ meanfreq: num 0.0598 0.066 0.0773 0.1512 0.1351 ...
 $ sd : num 0.0642 0.0673 0.0838 0.0721 0.0791 ...
 $ median : num 0.032 0.0402 0.0367 0.158 0.1247 ...
 $ Q25 : num 0.0151 0.0194 0.0087 0.0966 0.0787 ...
 $ Q75 : num 0.0902 0.0927 0.1319 0.208 0.206 ...
 $ IQR : num 0.0751 0.0733 0.1232 0.1114 0.1273 ...
 $ skew : num 12.86 22.42 30.76 1.23 1.1 ...
 $ kurt : num 274.4 634.61 1024.93 4.18 4.33 ...
 $ sp.ent : num 0.893 0.892 0.846 0.963 0.972 ...
 $ sfm : num 0.492 0.514 0.479 0.727 0.784 ...
 $ mode : num 0 0 0 0.0839 0.1043 ...
 $ centroid: num 0.0598 0.066 0.0773 0.1512 0.1351 ...
 $ meanfun : num 0.0843 0.1079 0.0987 0.089 0.1064 ...
 $ minfun : num 0.0157 0.0158 0.0157 0.0178 0.0169 ...
 $ maxfun : num 0.276 0.25 0.271 0.25 0.267 ...
 $ meandom : num 0.00781 0.00901 0.00799 0.2015 0.71281 ...
 $ mindom : num 0.00781 0.00781 0.00781 0.00781 0.00781 ...
 $ maxdom : num 0.00781 0.05469 0.01562 0.5625 5.48438 ...
 $ dfrange : num 0 0.04688 0.00781 0.55469 5.47656 ...
 $ modindx : num 0 0.0526 0.0465 0.2471 0.2083 ...
 $ label : chr "male" "male" "male" "male" ...
```

```
> voice$label=factor(voice$label)
```

```
> str(voice)
```

```
'data.frame': 3168 obs. of 21 variables:
 $ meanfreq: num 0.0598 0.066 0.0773 0.1512 0.1351 ...
 $ sd : num 0.0642 0.0673 0.0838 0.0721 0.0791 ...
 $ median : num 0.032 0.0402 0.0367 0.158 0.1247 ...
 $ Q25 : num 0.0151 0.0194 0.0087 0.0966 0.0787 ...
 $ Q75 : num 0.0902 0.0927 0.1319 0.208 0.206 ...
 $ IQR : num 0.0751 0.0733 0.1232 0.1114 0.1273 ...
 $ skew : num 12.86 22.42 30.76 1.23 1.1 ...
 $ kurt : num 274.4 634.61 1024.93 4.18 4.33 ...
 $ sp.ent : num 0.893 0.892 0.846 0.963 0.972 ...
 $ sfm : num 0.492 0.514 0.479 0.727 0.784 ...
 $ mode : num 0 0 0 0.0839 0.1043 ...
 $ centroid: num 0.0598 0.066 0.0773 0.1512 0.1351 ...
 $ meanfun : num 0.0843 0.1079 0.0987 0.089 0.1064 ...
 $ minfun : num 0.0157 0.0158 0.0157 0.0178 0.0169 ...
 $ maxfun : num 0.276 0.25 0.271 0.25 0.267 ...
 $ meandom : num 0.00781 0.00901 0.00799 0.2015 0.71281 ...
 $ mindom : num 0.00781 0.00781 0.00781 0.00781 0.00781 ...
 $ maxdom : num 0.00781 0.05469 0.01562 0.5625 5.48438 ...
 $ dfrange : num 0 0.04688 0.00781 0.55469 5.47656 ...
 $ modindx : num 0 0.0526 0.0465 0.2471 0.2083 ...
 $ label : Factor w/ 2 levels "female","male": 2 2 2 2 2 2 2 2 2 2 .
```


9.7 분류 모델의 다양한 적용

■ randomForest 적용

- 정확률은 $(1554+1548)/3168=97.92\%$

```
> f = randomForest(label~., data = voice)
> print(f)
```

Call:

```
randomForest(formula = label ~ ., data = voice)
  Type of random forest: classification
    Number of trees: 500
No. of variables tried at each split: 4
```

OOB estimate of error rate: 2.08%

Confusion matrix:

```
      female male class.error
female  1554   30  0.01893939
male     36 1548  0.02272727
```

```
> treesize(f)
```

```
[1] 69 68 80 98 84 56 68 85 78 118 102 81 74 75 96 72 85 72 101 88 82 98 109 111
[25] 74 89 68 82 78 73 92 67 78 71 85 90 69 76 79 82 96 67 66 96 85 117 76 69
[49] 68 76 92 72 110 76 67 90 83 69 92 96 93 96 92 96 78 91 48 65 80 68 98 72
[73] 98 83 72 70 63 85 91 75 105 74 60 82 88 65 103 95 86 96 74 85 62 76 68 88
[97] 72 89 60 89 77 83 70 65 64 68 75 66 76 77 76 75 96 97 87 92 71 73 63 97
[121] 86 73 108 96 76 82 111 88 62 71 90 85 83 99 109 89 64 97 89 97 91 88 88 72
[145] 104 97 96 81 70 98 71 88 68 83 115 65 61 77 74 83 87 64 95 81 88 92 79 76
[169] 92 76 86 63 76 69 76 64 76 80 96 92 79 93 61 80 92 83 86 70 78 72 80 84
[193] 72 63 72 99 86 94 71 66 89 76 66 71 85 68 64 69 80 86 96 88 81 83 98 81
[217] 93 79 80 81 64 84 74 134 75 81 100 89 75 106 81 80 62 62 66 89 82 79 65 72
[241] 76 95 78 84 67 93 80 61 83 71 70 67 96 101 77 93 85 82 64 70 75 69 88 69
[265] 65 97 81 78 113 79 114 100 51 56 72 78 73 101 94 71 84 116 102 78 108 95 107 83
[289] 91 98 77 83 67 67 86 98 71 64 64 65 73 91 74 66 73 111 91 95 111 69 77 101
[313] 66 107 65 69 106 86 97 83 76 77 70 91 71 60 70 89 102 93 83 101 60 81 74 69
[337] 70 76 80 108 70 63 88 96 79 75 87 75 91 102 69 91 70 78 76 88 64 67 74 68
[361] 62 71 75 79 91 86 74 101 100 82 82 92 92 84 76 82 66 60 97 68 89 69 75 83
[385] 70 86 75 88 83 79 108 81 72 100 87 76 88 69 59 65 72 87 87 116 66 78 68 73
[409] 62 80 101 70 59 62 73 66 101 97 75 66 92 103 117 81 67 75 110 72 95 71 79 78
[433] 88 76 107 85 64 101 99 84 98 94 83 83 113 78 72 76 106 82 63 77 87 79 92 88
[457] 99 78 73 63 78 87 92 78 76 84 76 77 93 63 80 82 72 81 66 62 62 73 105 91
[481] 82 74 86 94 78 96 91 58 85 69 100 74 53 60 89 63 71 101 103 95
```

책의 1555+1549와 차이나는 이유?

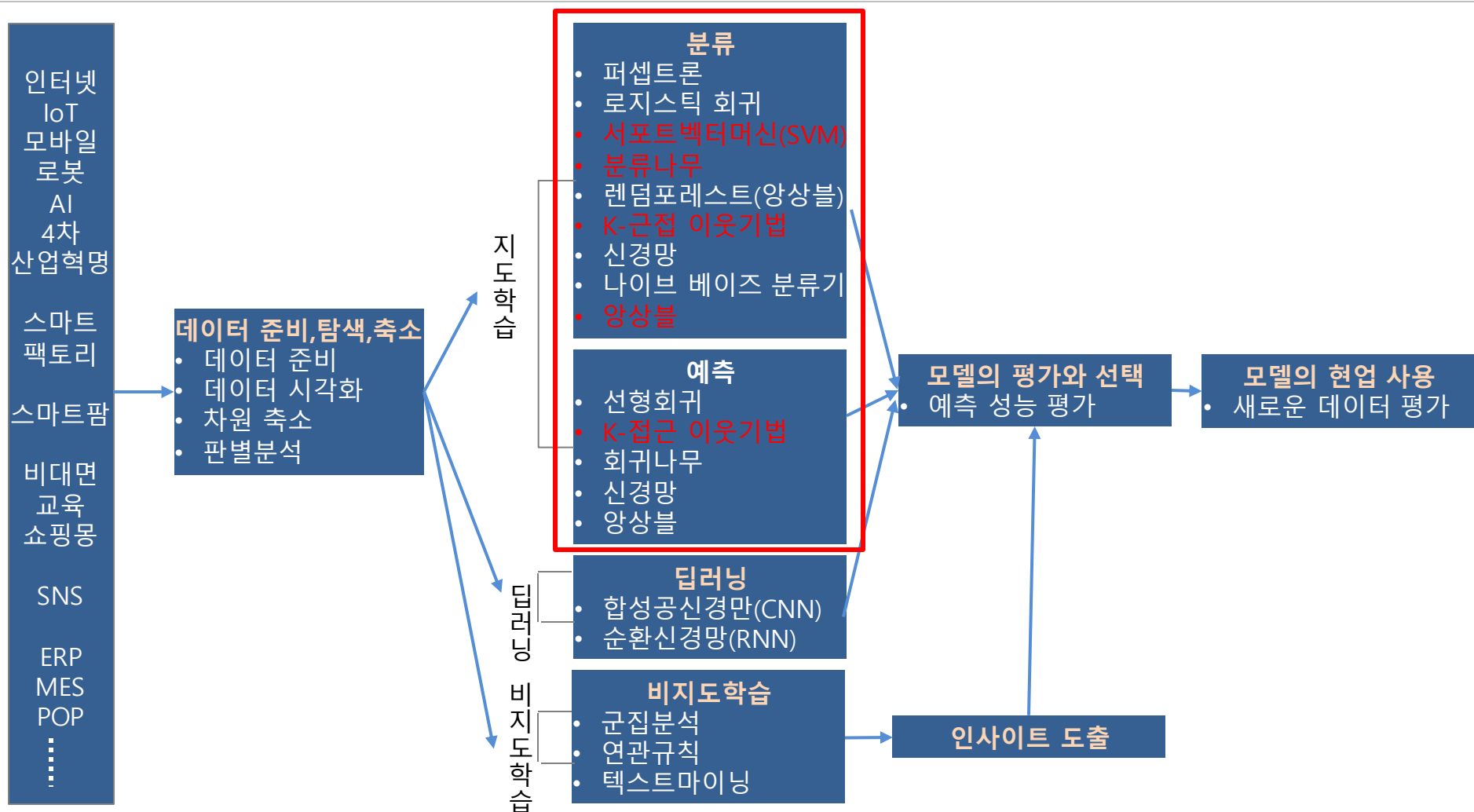
9.7 분류 모델의 다양한 적용

■ 지금까지 성능 측정의 문제점

- 지금까지의 성능 측정 기준 : 정확률 ($146/150 = 97.3\%$)
- 지금까지의 문제점
 - ✓ 학습할 때 사용한 훈련 집합을 성능 측정에 다시 사용함
→ 교과서에서 풀어본 문제가 그대로 출제된 셈. 일반화 능력을 제대로 측정하지 못함

■ 10장의 주제는 성능 측정

요약



Thank you

