그리디 알고리즘

분류 규칙 학습 알고리즘

분류 규칙의 이해

- 분류 규칙은 클래스를 레이블이 없는 예시에 할당하는 논리적인 if-else구문 형태로 지식을 표현
- 규칙이 명목형인 문제에 적용
- Classification rules
  - If **조건부**(특징들의 조합) then 결론부(배정할 클래스)
  - 예) 규칙 R1이 있다고 하면
    - If '하드 드라이브가 딸각하는 소리가 난다'

      then '이제 막 컴퓨터가 고장이 나려고 한다'
  - 예제
    - 기계 장비에 하드웨어 고장을 유발하는 조건의 식별
    - 고객 세분화를 위한 그룹의 주요 특징 기술
    - 주식 시장에서 큰 폭의 주가 하락이나 주가 상승에 선행하는 조건을 찾는 일

분리 정복

- Separate and conquer
  - 휴리스틱 알고리즘
    - Train dataset에서 예시의 부분집합을 커버하는 규칙을 식별하고 이 부분집합을 나머지 데이터와 분리하고 규칙이 추가되면서 데이터의 부분집합도 추가적으로 분리되고 전체 dataset이 모두 커버되고 더 이상의 예시가 남아있지 않을 때까지 진행
    - Data를 파고드는 것.

- Covering algorithms
  - 규칙이 큰 데이터 세그먼트를 점진적으로 소비해 결국 모든 인스턴스를 분류
  - 규칙이 데이터의 부분을 커버하는 것처럼 보임
  - 커버링 규칙

동물 중에서 포유류를 식별하는 규칙을 만들어 보자.

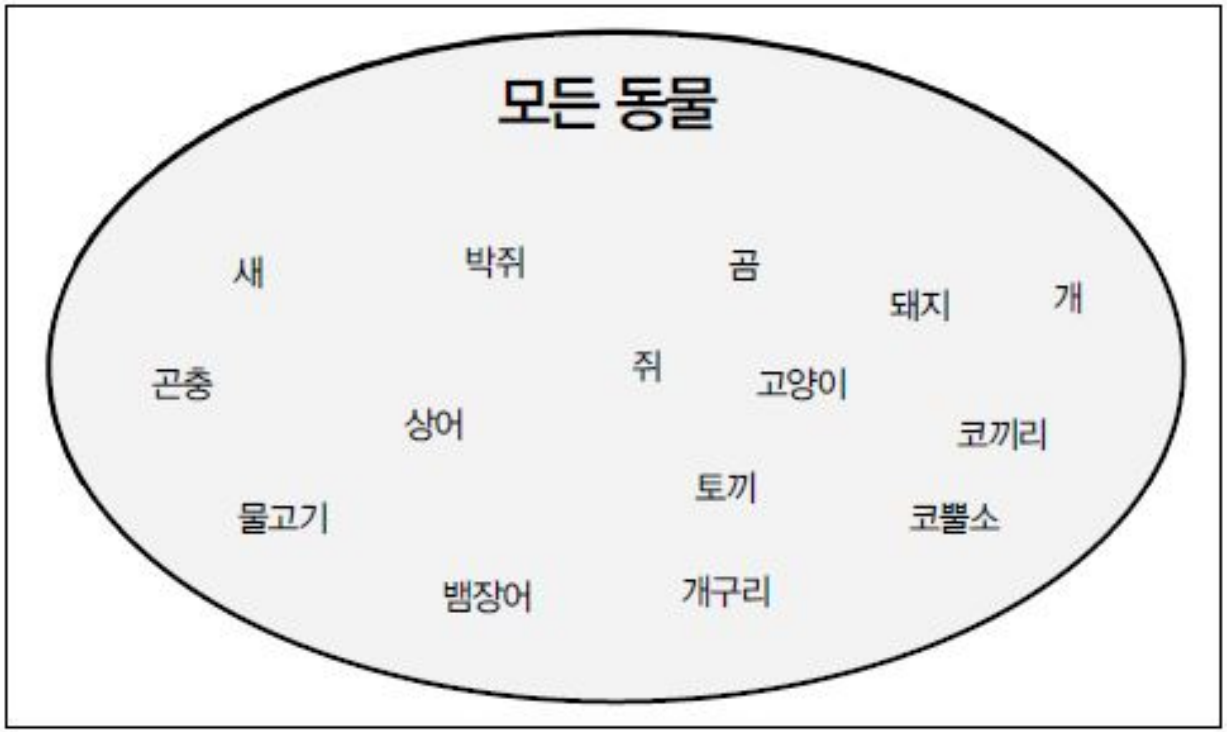


그림 5.7: 규칙 학습 알고리즘은 동물을 포유류와 비포유류로 나누는 데 도움을 줄 수 있다.

첫 번째 규칙 : 육지에 사는 동물은 포유류이다.

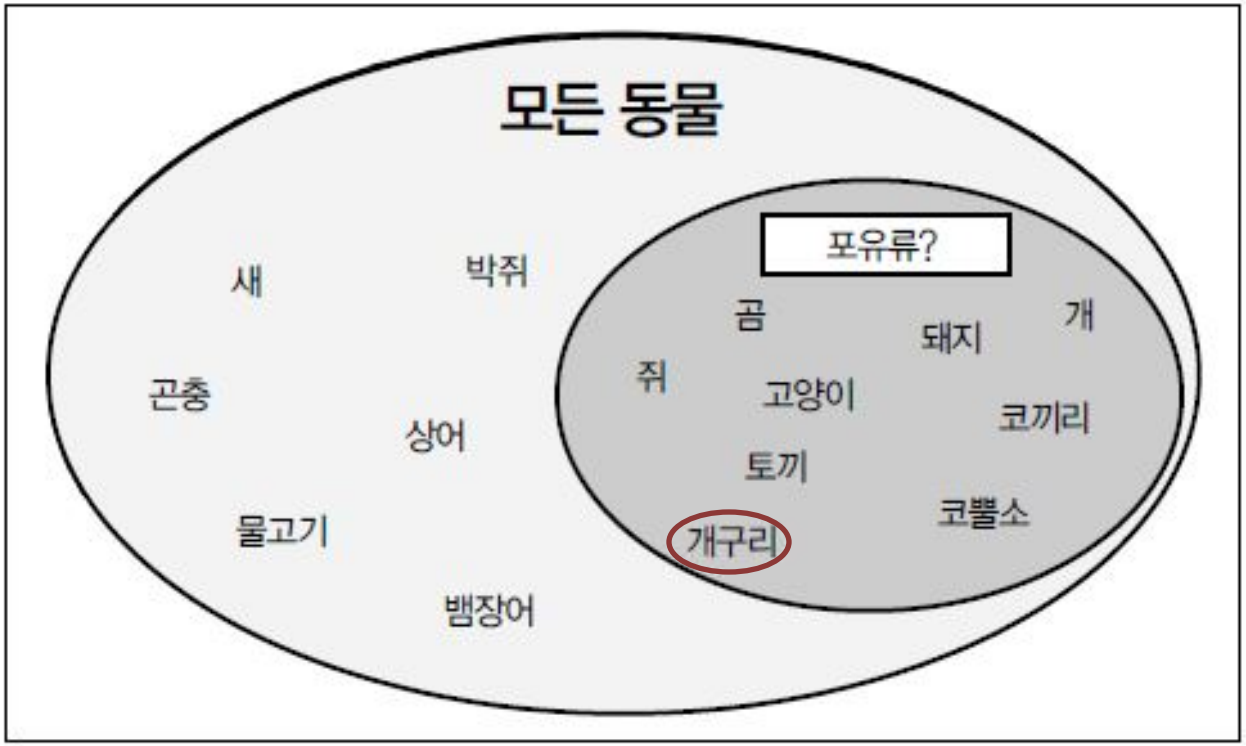


그림 5.8: 잠재 규칙은 육지에서 걷는 동물은 포유류로 간주한다.

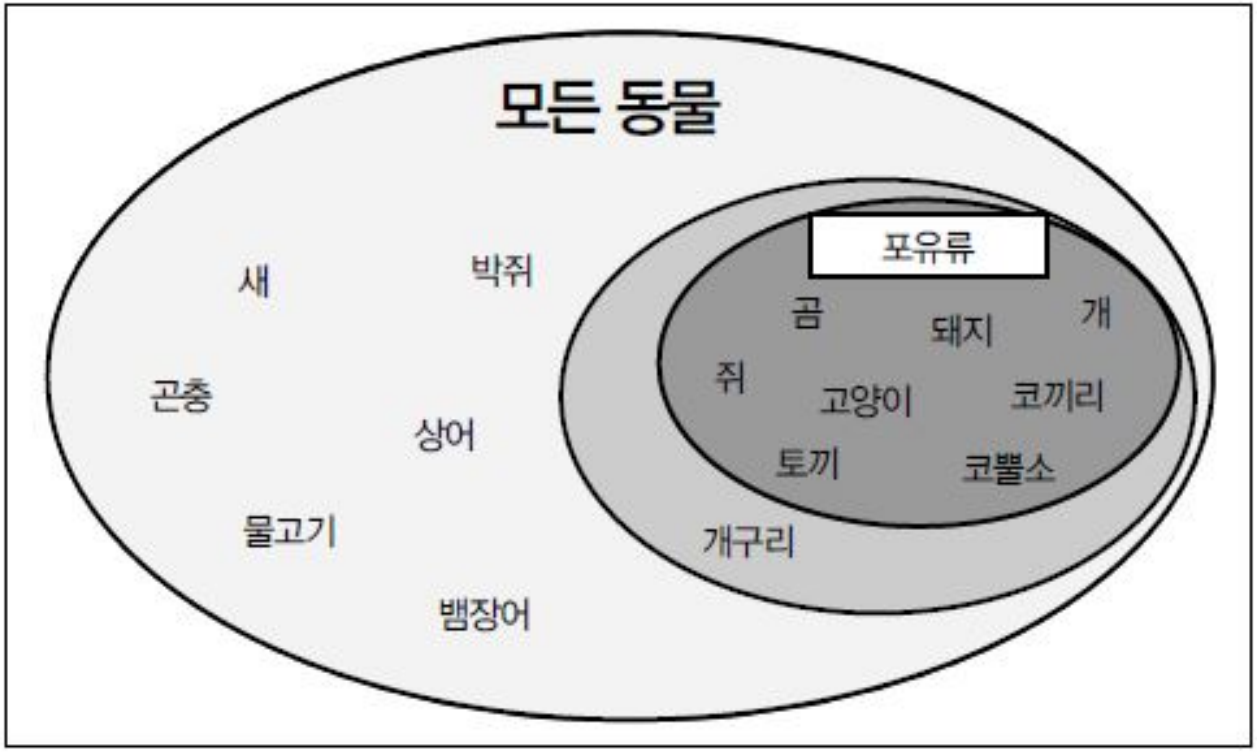


그림 5.9: 좀 더 구체적인 규칙은 육지에서 걷고 꼬리를 가진 동물을 포유류로 제시한다.

분리 정복 알고리즘 = 커버링 알고리즘(covering algorithms)

두 번째 규칙 : 털이 있는가?

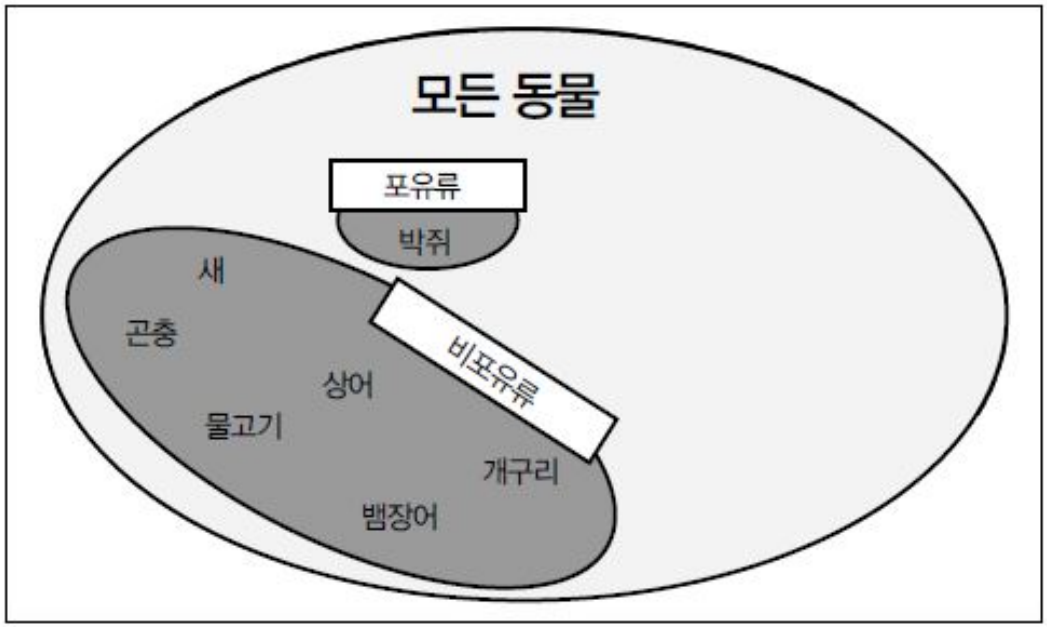


그림 5.10: 털을 가진 동물은 포유류라는 규칙은 나머지 동물을 완벽하게 분류한다.

커버링 규칙

1. 땅에서 걷고 꼬리가 있는 동물은 포유류이다.
2. 털이 없다면 포유류가 아니다
3. 그렇지 않은 동물은 모두 포유류이다.

계통 분류 [편집]

다음은 포유류 계통 분류이다.[3]

논란 [편집]

최근 발생되는 여러가지 신종 전염병의 자연숙주로 지목 받고 있으며 **메르스**, **사스**, **에볼라**, **니파바이러스**,**우한 폐렴**의 자연숙주로 알려 지고 있다. **에볼라**는 박쥐에서 바로 사람으로, **사스**는 박쥐에서 사향고양이로 **니파바이러스**는 박쥐에서 돼지를 통해, **메르스**는 박쥐에서 낙타를 통해 바이러스가 사람에게 옮겨 진다는 연구 결과가 보도 되고 있다.

1R 알고리즘

- 1R알고리즘
  - ZeroR 알고리즘을 개선
    - 어떤 특징도 고려하지 않고 아무 규칙도 학습하지 않는 규칙 학습자. 특징 값에 상관없이 레이블 되지 않은 모든 예시에 대해 가장 흔한 클래스를 예측한다.
  - One Rule(OneR)
    - 장점
      - 이해하기 쉽고 하나의 경험적 규칙을 생성
      - 예상보다 훨씬 잘 실행되는 경향
      - 좀 더 복잡한 알고리즘의 기준점 제공
    - 단점
      - 하나의 특징만을 사용
      - 너무 단순

1R 알고리즘

- 1R알고리즘
  1. 각 특징에 대해 유사한 값을 기준으로 데이터를 그룹으로 분리
  2. 각 세그먼트에 대해 대다수 클래스를 예측
  3. 각 특징에 기반을 두는 규칙의 오류율을 계산하고 최소 오류를 갖는 규칙을 단일 규칙으로 선정

동물	이동 경로	털이 있는	포유류
박쥐	하늘	예	예
곰	땅	예	예
새	하늘	아니오	
고양이	땅	예	예
개	땅	예	예
뱀장어	바다	아니오	아니오
코끼리	땅	아니오	예
물고기	바다	아니오	아니오
개구리	땅	아니오	아니오
곤충	하늘	아니오	아니오
돼지	땅	아니오	예
토끼	땅	예	예
쥐	땅	예	예
코뿔소	땅	아니오	예
상어	바다	아니오	아니오

전체 데이터셋

이동 경로	예측	포유류	
하늘	아니오	예	✖
하늘	아니오	아니오	
하늘	아니오	아니오	
땅	예	예	
땅	예	예	
땅	예	예	
땅	예	예	
땅	예	아니오	✖
땅	예	예	
땅	예	예	
땅	예	예	
땅	예	예	
바다	아니오	아니오	
바다	아니오	아니오	
바다	아니오	아니오	

'이동 경로' 규칙
오류율 = 2/15

털이 있는	예측	포유류	
아니오	아니오	아니오	
아니오	아니오	아니오	
아니오	아니오	예	✖
아니오	아니오	아니오	
아니오	아니오	아니오	
아니오	아니오	아니오	
아니오	아니오	예	✖
아니오	아니오	예	✖
아니오	아니오	아니오	
예	예	예	
예	예	예	
예	예	예	
예	예	예	
예	예	예	
예	예	예	

'털이 있는' 규칙
오류율 = 3/15

그림 5.11: 1R 알고리즘은 가장 작은 오분류율을 가진 단일 규칙을 선택한다.

1R 알고리즘

동물	이동 경로	털이 있는	포유류
박쥐	하늘	예	예
곰	땅	예	예
새	하늘	아니오	
고양이	땅	예	예
개	땅	예	예
뱀장어	바다	아니오	아니오
코끼리	땅	아니오	예
물고기	바다	아니오	아니오
개구리	땅	아니오	아니오
곤충	하늘	아니오	아니오
돼지	땅	아니오	예
토끼	땅	예	예
쥐	땅	예	예
코뿔소	땅	아니오	예
상어	바다	아니오	아니오

전체 데이터셋

이동 경로	예측	포유류	
하늘	아니오	예	✖
하늘	아니오	아니오	
하늘	아니오	아니오	
땅	예	예	
땅	예	예	
땅	예	예	
땅	예	예	
땅	예	아니오	✖
땅	예	예	
땅	예	예	
땅	예	예	
땅	예	예	
바다	아니오	아니오	
바다	아니오	아니오	
바다	아니오	아니오	

'이동 경로' 규칙
오류율 = 2/15

털이 있는	예측	포유류	
아니오	아니오	아니오	
아니오	아니오	아니오	
아니오	아니오	예	✖
아니오	아니오	아니오	
아니오	아니오	아니오	
아니오	아니오	아니오	
아니오	아니오	예	✖
아니오	아니오	예	✖
아니오	아니오	아니오	
예	예	예	
예	예	예	
예	예	예	
예	예	예	
예	예	예	
예	예	예	

'털이 있는' 규칙
오류율 = 3/15

그림 5.11: 1R 알고리즘은 가장 작은 오분류율을 가진 단일 규칙을 선택한다.

이동 경로 규칙 : 박쥐와 개구리에서 두 개의 오류 발생
털이 있는 규칙 : 코끼리, 돼지, 코뿔소에서 세 개의 오류 발생
이동 경로 규칙의 오류가 적기 때문에 이동 경로 기반의 규칙 반환

1. 동물이 하늘로 이동을 하면 포유류가 아니다.
2. 동물이 땅으로 이동을 하면 포유류이다.
3. 동물이 바다로 이동을 하면 포유류가 아니다.

리퍼 알고리즘

- 초기 규칙 학습 알고리즘의 문제점
  - 속도가 느려서 빅데이터 처리에 효과가 떨어짐
  - 노이즈가 섞인 데이터의 경우 쉽게 부정확해짐
  - IREP(Incremental Reduced Error Pruning)
  - RIPPER(Repeated Incremental Pruning to Produce Error)
- RIPPER 알고리즘
  - 장점
    - 이해하기 쉽고 사람이 읽을 수 있는 규칙 생성
    - 크고 노이즈가 있는 데이터셋에 효율적
    - 비교 가능한 DT보다 좀 더 간단한 모델을 생성
  - 단점
    - 상식이나 전문가 지식에 위배되는 것처럼 보이는 규칙이 생길 가능성
    - 수치 데이터 작업에 부적합
    - 좀 더 복잡한 모델만큼 잘 수행되지 않을 수 있다.

규칙 알고리즘 과정

- 리퍼 알고리즘
  - 순차포괄 알고리즘
  - 1R보다 복수의 조건부를 갖는 규칙을 생성
  - 복잡한 데이터를 모델링하는 알고리즘 능력은 향상되지만 금방 규칙을 파악하기는 쉽지 않다.

- 알고리즘 3단계
  1. 기르기
     - 분할 정복 기법(재귀적분할 기법)
     - 데이터의 부분집합을 완벽하게 분류하거나 분할을 위한 속성이 없어질 때까지 진행
     - 정보 획득량을 기준
  2. 가지치기
     - 규칙의 구체성이 증가되어도 더 이상 엔트로피가 줄지 않을 때 해당 규칙의 사용을 멈춤
  3. 최적화하기
     - 전체 규칙이 최적화되는 종료 조건에 도달할 때까지 성장과 가지치기를 반복

의사 결정 트리에서 규칙 구성

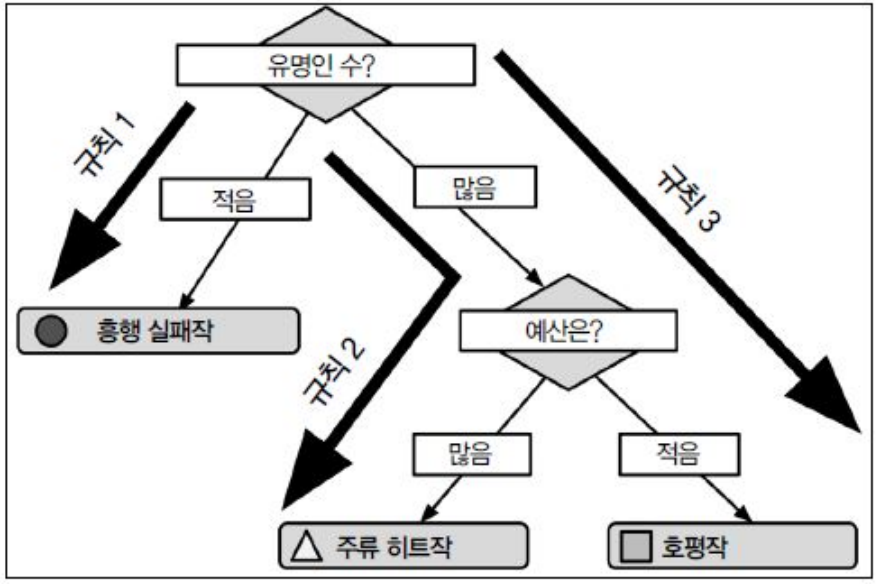


그림 5.12: 규칙은 결정 트리의 루트 노드에서 각 잎 노드를 따라가면 생성할 수 있다.

루트 노드에서 잎 노드까지 경로를 따라가는 규칙

1. 유명인 수가 적다면 영화는 흥행 실패작이 될 것이다.
2. 유명인 수가 많고 예산이 많으면 주류 히트작이 될 것이다.
3. 유명인 수가 많고 예산이 적다면 호평작이 될 것이다.

탐욕 알고리즘(Greedy Algorithm, 욕심쟁이 방법)

선입 선처리(First-Come, First-Served : FCFS)
: 한 번에 하나의 분할을 하고자 하며, 먼저 가장 동질적인 파티션을 찾고 난 후에 차선을 찾고, 모든 예시가 다 분류될 때까지 계속 찾는다. (최적화 문제 해결)

단점 : 가장 쉽게 달성할 수 있는 목표를 빨리 이룸으로써 데이터의 한 부분집합에 대해 정확한 한 개의 규칙을 빠르게 발견할 수 있지만 데이터 전체에 대한 더 나은 규칙을 개발하지 못한다. (근시안적인 방법)

특징 :
1. 입력 데이터 간의 관계를 고려하지 않고 수행 과정에서 '욕심스럽게' 최소값 또는 최대값을 가진 데이터를 선택.

2. 한 번 선택한 데이터를 번복하지 않고 그대로 진행하므로 단순하고 제한적이다.

그림 5.13: 결정 트리와 분류 규칙은 모두 탐욕 알고리즘이다.

의사 결정 트리와 규칙 학습 알고리즘의 규칙 생성 차이점

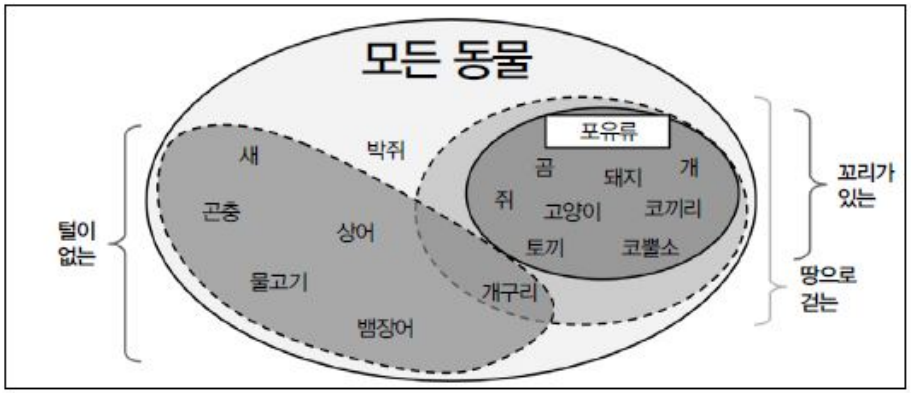


그림 5.14: 개구리의 처리법이 분할 정복과 분리 정복 휴리스틱의 차이를 구분해준다. 분리 정복은 개구리가 그 이후의 규칙에 의해 재정복되게 한다.

규칙 학습 알고리즘

•땅에서 걷고 꼬리가 있는 동물은 포유류다.

(곰, 고양이, 개, 코끼리, 돼지, 토끼, 쥐, 코뿔소)

•동물이 털이 없다면 포유류가 아니다.

(새, 독수리, 물고기, 개구리, 곤충, 상어)

•그렇지 않으면 동물은 포유류다(박쥐)

의사 결정 트리

•동물이 땅에서 걷고 털이 있다면 포유류다.

(곰, 고양이, 개, 코끼리, 돼지, 토끼, 쥐, 코뿔소)

•동물이 땅에서 걷고 털이 없으면 포유류가 아니다.(개구리)

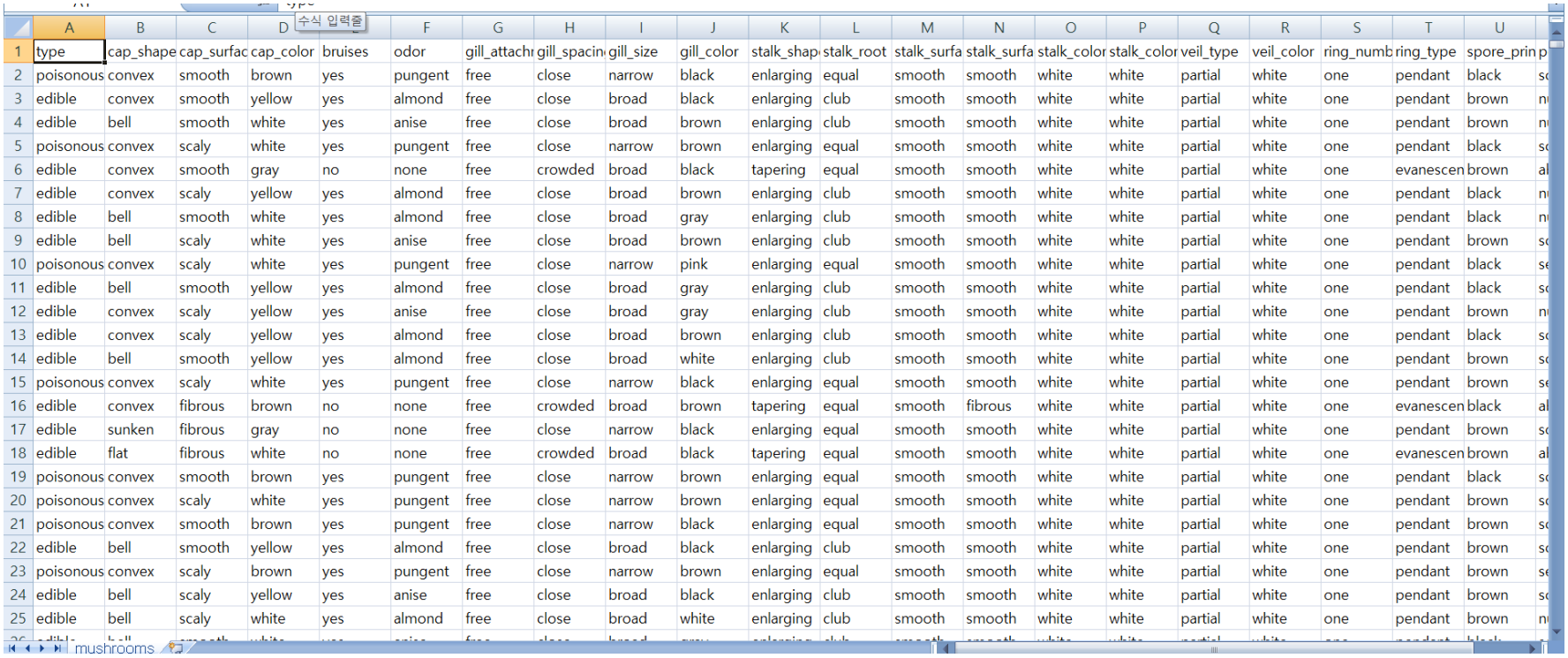
•동물이 땅에서 걷지 않고 털이 있으면 포유류다.(박쥐)

•동물이 땅에서 걷지 않고 털이 없으면 포유류가 아니다.

(새, 독수리, 물고기, 곤충, 상어)

예제 : 규칙 학습자를 이용한 독버섯 식별

- 독버섯을 구분하는 규칙 식별
  - 데이터 : 8124개의 버섯 샘플, 22개의 특징
  - Type : 식용(edible), 독이 있는(poisonous)



	A	B	C	D	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
1	type	cap_shape	cap_surfac	cap_color	bruises	odor	gill_attachr	gill_spacin	gill_size	gill_color	stalk_shap	stalk_root	stalk_surfa	stalk_surfa	stalk_color	stalk_color	veil_type	veil_color	ring_numb	ring_type	spore_prin
2	poisonous	convex	smooth	brown	yes	pungent	free	close	narrow	black	enlarging	equal	smooth	smooth	white	white	partial	white	one	pendant	black
3	edible	convex	smooth	yellow	yes	almond	free	close	broad	black	enlarging	club	smooth	smooth	white	white	partial	white	one	pendant	brown
4	edible	bell	smooth	white	yes	anise	free	close	broad	brown	enlarging	club	smooth	smooth	white	white	partial	white	one	pendant	brown
5	poisonous	convex	scaly	white	yes	pungent	free	close	narrow	brown	enlarging	equal	smooth	smooth	white	white	partial	white	one	pendant	black
6	edible	convex	smooth	gray	no	none	free	crowded	broad	black	tapering	equal	smooth	smooth	white	white	partial	white	one	evanescen	brown
7	edible	convex	scaly	yellow	yes	almond	free	close	broad	brown	enlarging	club	smooth	smooth	white	white	partial	white	one	pendant	black
8	edible	bell	smooth	white	yes	almond	free	close	broad	gray	enlarging	club	smooth	smooth	white	white	partial	white	one	pendant	black
9	edible	bell	scaly	white	yes	anise	free	close	broad	brown	enlarging	club	smooth	smooth	white	white	partial	white	one	pendant	brown
10	poisonous	convex	scaly	white	yes	pungent	free	close	narrow	pink	enlarging	equal	smooth	smooth	white	white	partial	white	one	pendant	black
11	edible	bell	smooth	yellow	yes	almond	free	close	broad	gray	enlarging	club	smooth	smooth	white	white	partial	white	one	pendant	black
12	edible	convex	scaly	yellow	yes	anise	free	close	broad	gray	enlarging	club	smooth	smooth	white	white	partial	white	one	pendant	brown
13	edible	convex	scaly	yellow	yes	almond	free	close	broad	brown	enlarging	club	smooth	smooth	white	white	partial	white	one	pendant	black
14	edible	bell	smooth	yellow	yes	almond	free	close	broad	white	enlarging	club	smooth	smooth	white	white	partial	white	one	pendant	brown
15	poisonous	convex	scaly	white	yes	pungent	free	close	narrow	black	enlarging	equal	smooth	smooth	white	white	partial	white	one	pendant	brown
16	edible	convex	fibrous	brown	no	none	free	crowded	broad	brown	tapering	equal	smooth	fibrous	white	white	partial	white	one	evanescen	black
17	edible	sunken	fibrous	gray	no	none	free	close	narrow	black	enlarging	equal	smooth	smooth	white	white	partial	white	one	pendant	brown
18	edible	flat	fibrous	white	no	none	free	crowded	broad	black	tapering	equal	smooth	smooth	white	white	partial	white	one	evanescen	brown
19	poisonous	convex	smooth	brown	yes	pungent	free	close	narrow	brown	enlarging	equal	smooth	smooth	white	white	partial	white	one	pendant	black
20	poisonous	convex	scaly	white	yes	pungent	free	close	narrow	brown	enlarging	equal	smooth	smooth	white	white	partial	white	one	pendant	brown
21	poisonous	convex	smooth	brown	yes	pungent	free	close	narrow	black	enlarging	equal	smooth	smooth	white	white	partial	white	one	pendant	brown
22	edible	bell	smooth	yellow	yes	almond	free	close	broad	black	enlarging	club	smooth	smooth	white	white	partial	white	one	pendant	brown
23	poisonous	convex	scaly	brown	yes	pungent	free	close	narrow	brown	enlarging	equal	smooth	smooth	white	white	partial	white	one	pendant	brown
24	edible	bell	scaly	yellow	yes	anise	free	close	broad	black	enlarging	club	smooth	smooth	white	white	partial	white	one	pendant	brown
25	edible	bell	scaly	white	yes	almond	free	close	broad	white	enlarging	club	smooth	smooth	white	white	partial	white	one	pendant	brown

mushrooms

1R classification rule syntax

using the `OneR()` function in the `RWeka` package

Building the classifier:

```
m <- OneR(class ~ predictors, data = mydata)
```

- `class` is the column in the `mydata` data frame to be predicted
- `predictors` is an R formula specifying the features in the `mydata` data frame to use for prediction
- `data` is the data frame in which `class` and `predictors` can be found

The function will return a 1R model object that can be used to make predictions.

Making predictions:

```
p <- predict(m, test)
```

- `m` is a model trained by the `OneR()` function
- `test` is a data frame containing test data with the same features as the training data used to build the classifier.

The function will return a vector of predicted class values.

Example:

```
mushroom_classifier <- OneR(type ~ odor + cap_color,
                            data = mushroom_train)
mushroom_prediction <- predict(mushroom_classifier,
                               mushroom_test)
```

RIPPER classification rule syntax

using the `JRip()` function in the `RWeka` package

Building the classifier:

```
m <- JRip(class ~ predictors, data = mydata)
```

- `class` is the column in the `mydata` data frame to be predicted
- `predictors` is an R formula specifying the features in the `mydata` data frame to use for prediction
- `data` is the data frame in which `class` and `predictors` can be found

The function will return a RIPPER model object that can be used to make predictions.

Making predictions:

```
p <- predict(m, test)
```

- `m` is a model trained by the `JRip()` function
- `test` is a data frame containing test data with the same features as the training data used to build the classifier.

The function will return a vector of predicted class values.

Example:

```
mushroom_classifier <- JRip(type ~ odor + cap_color,
                            data = mushroom_train)
mushroom_prediction <- predict(mushroom_classifier,
                               mushroom_test)
```

데이터 전처리

```
#install.packages("RWeka")

#install.packages("OneR")

library(RWeka)

library(OneR)

mushrooms <- read.csv("mushrooms.csv", stringsAsFactors = TRUE)

str(mushrooms) # oneR은 모두 명목화


#변수제거

mushrooms$veil_type <- NULL

str(mushrooms)

table(mushrooms$type)
```

데이터 모델 훈련

```
#oneR modeling

mushroom_1R <- OneR(type ~ ., data=mushrooms)
```

결과

```
Rules:
If odor = almond   then type = edible
If odor = anise    then type = edible
If odor = creosote then type = poisonous
If odor = fishy    then type = poisonous
If odor = foul     then type = poisonous
If odor = musty    then type = poisonous
If odor = none     then type = edible
If odor = pungent  then type = poisonous
If odor = spicy    then type = poisonous

Accuracy:
8004 of 8124 instances classified correctly (98.52%)
```

```
#model 평가

mushroom_1R_pred <- predict(mushroom_1R,mushrooms)

table(actual=mushrooms$type, predicted=mushroom_1R_pred)
```

```
                predicted
actual        edible poisonous
  edible        4208         0
  poisonous      120      3796
```

결과 : 실제 독버섯 120개를 식용 버섯으로 잘못 분류.

```
#model 성능개선

mushroom_JRip <- JRip(type ~ ., data=mushrooms)

mushroom_JRip_pred <- predict(mushroom_JRip, mushrooms)

mushroom_Jrip

table(actual=mushrooms$type, predicted=mushroom_JRip_pred)
```

```
JRIP rules:
===========

(odor = foul) => type=poisonous (2160.0/0.0)
(gill_size = narrow) and (gill_color = buff) => type=poisonous (1152.0/0.0)
(gill_size = narrow) and (odor = pungent) => type=poisonous (256.0/0.0)
(odor = creosote) => type=poisonous (192.0/0.0)
(spore_print_color = green) => type=poisonous (72.0/0.0)
(stalk_surface_below_ring = scaly) and (stalk_surface_above_ring = silky) => type=poisonous (68.0/0.0)
(habitat = leaves) and (cap_color = white) => type=poisonous (8.0/0.0)
(stalk_color_above_ring = yellow) => type=poisonous (8.0/0.0)
 => type=edible (4208.0/0.0)
Number of Rules : 9
```

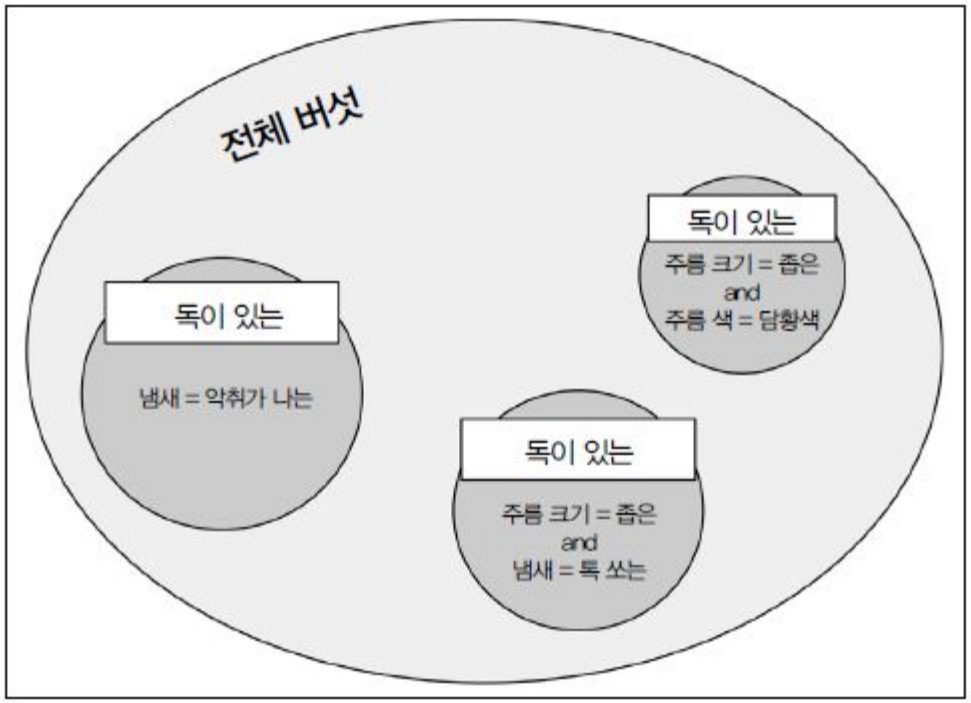


그림 5.15: 정교한 규칙 학습 알고리즘으로 모든 종류의 독버섯을 완벽히 인식하는 규칙을 알아냈다.

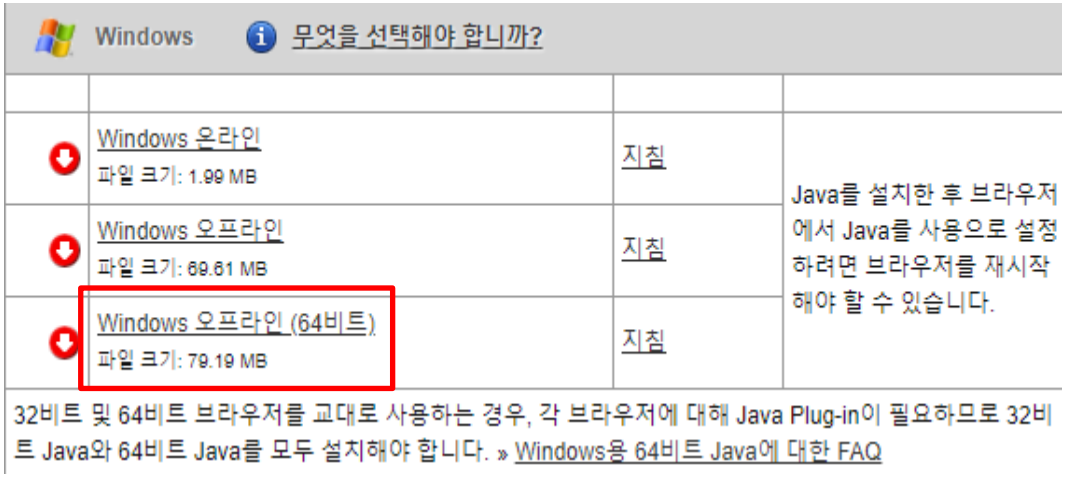
에러: package or namespace load failed for 'RWeka':
.onLoad가 loadNamespace()에서 'rJava'때문에 실패했습니다:
호출: fun(libname, pkgname)
에러: JAVA_HOME cannot be determined from the Registry

오류 발생시 해결방법

1. JAVA_HOME을 설정하려면 일단 Java가 설치되어있는지 확인하고 설치가 되어있지 않다면 Java를 설치해야 한다.
먼저 C:₩Program Files 아래에 Java라는 폴더가 있는지 확인한다. (Program Files (x86) 폴더 아님)

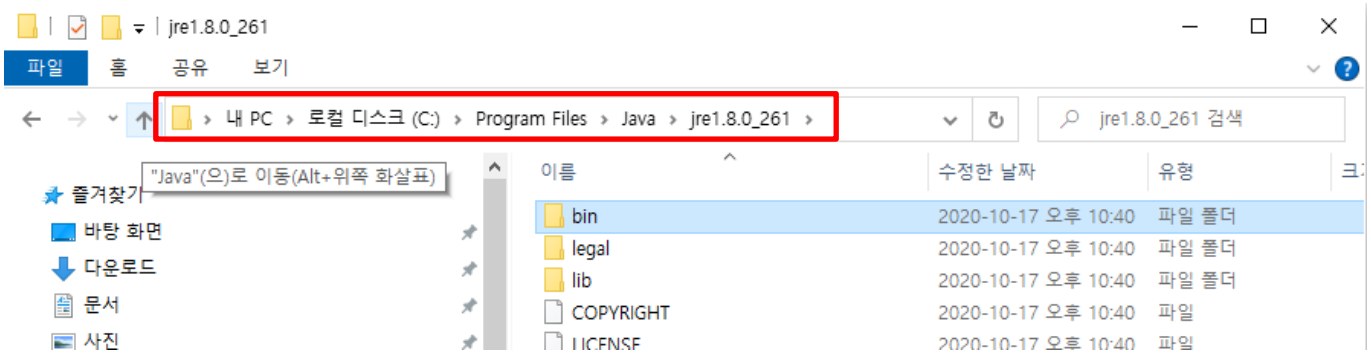
2. 내 컴퓨터 또는 내 PC에 마우스 오른쪽 클릭 메뉴에서 속성을 클릭한다.
시스템 종류가 64-bit 인지 32-bit 인지 확인한다.

3. Java 설치하기
만약 Java 폴더가 없다면 Java 다운로드(https://www.java.com/ko/download/manual.jsp)에서 Java를 다운로드 및 설치한다.
컴퓨터가 64-bit인 경우 반드시 Windows 오프라인 (64비트) 를 클릭하여 설치해야 한다.

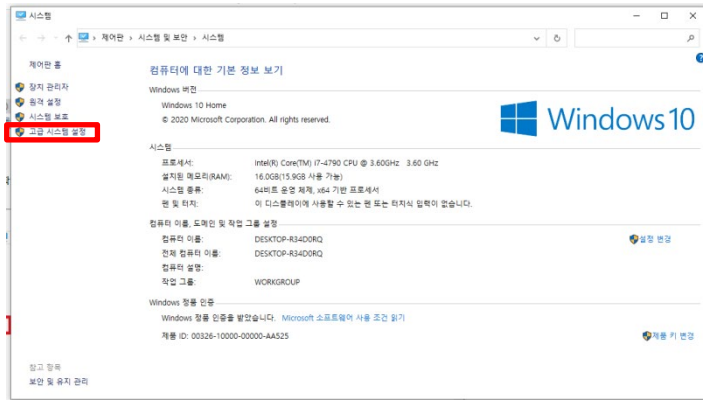


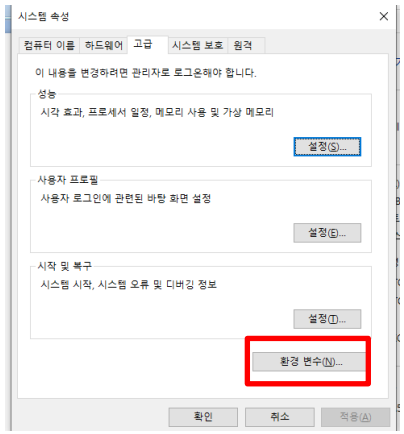
오류 발생시 해결방법

4. 환경변수 설정하기 - JAVA_HOME, PATH
설치가 끝났으면 (3)번 처럼 다시 C:₩Program Files 아래에 Java라는 폴더가 있는지 확인한다.
Java 폴더가 있다면 Java 폴더 아래에 있는 폴더의 경로를 복사해둔다.



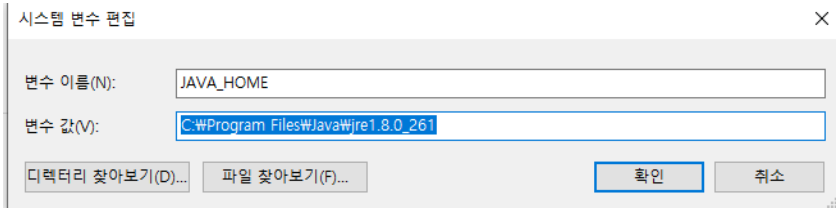
4. 환경변수 설정하기 - JAVA_HOME, PATH
설치가 끝났으면 (3)번 처럼 다시 C:₩Program Files 아래에 Java라는 폴더가 있는지 확인한다.
Java 폴더가 있다면 Java 폴더 아래에 있는 폴더의 경로를 복사해둔다.



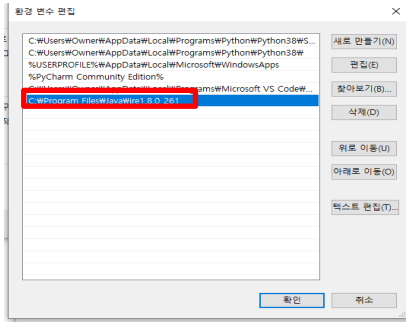


오류 발생시 해결방법

5. 시스템 변수에서 JAVA_HOME 설치
자바를 새로 설치한 경우, 새로 만들기를 클릭하여 변수 이름에는 JAVA_HOME을 입력하고
변수 값에는 복사해둔 경로 C:₩Program Files₩Java₩jre1.8.0_261 를 입력한 후 확인을 눌러준다.
만약 JAVA_HOME이 이미 있는 경우에는 편집을 클릭한다.
편집인 경우 변수 값에 이미 들어가 있는 경로 뒤에 세미콜론 ; 가 있는지 확인하고 없으면 ; 뒤에
복사해둔 경로 C:₩Program Files₩Java₩jre1.8.0_261 를 입력한 후 확인을 눌러준다.
(세미콜론은 2가지 이상의 경로를 구분하는 구분자 역할)



5. 위쪽 사용자 JAVA_HOME 설치
위쪽 사용자 변수에서 PATH를 설정한다. 대부분의 경우 PATH는 이미 있는 경우가 많은데,
이미 있는 경우 PATH를 선택하여 편집을 클릭한다. 만약 없으면 새로 만들기를 한다.
편집인 경우 변수 값에 이미 들어가 있는 경로 뒤에 세미콜론 ; 가 있는지 확인하고 없으면 ; 뒤에
복사해둔 경로 C:₩Program Files₩Java₩jre1.8.0_261를 입력한 후 추가로 ₩bin을 붙여준다.
다시 말해 C:₩Program Files₩Java₩jre1.8.0_261₩bin을 입력하면 된다.



6. R에서 library(RWeka)를 다시 실행한다. 만약 그래도 안되면 R script 창에
Sys.setenv(JAVA_HOME='C:/Program Files/Java/jre1.8.0_261')