분류 알고리즘 - 1

Kyungsik Han

본 영상에서 다룰 내용

CODING

- 지도학습 분류(classification) 알고리즘 학습

분류 알고리즘 종류

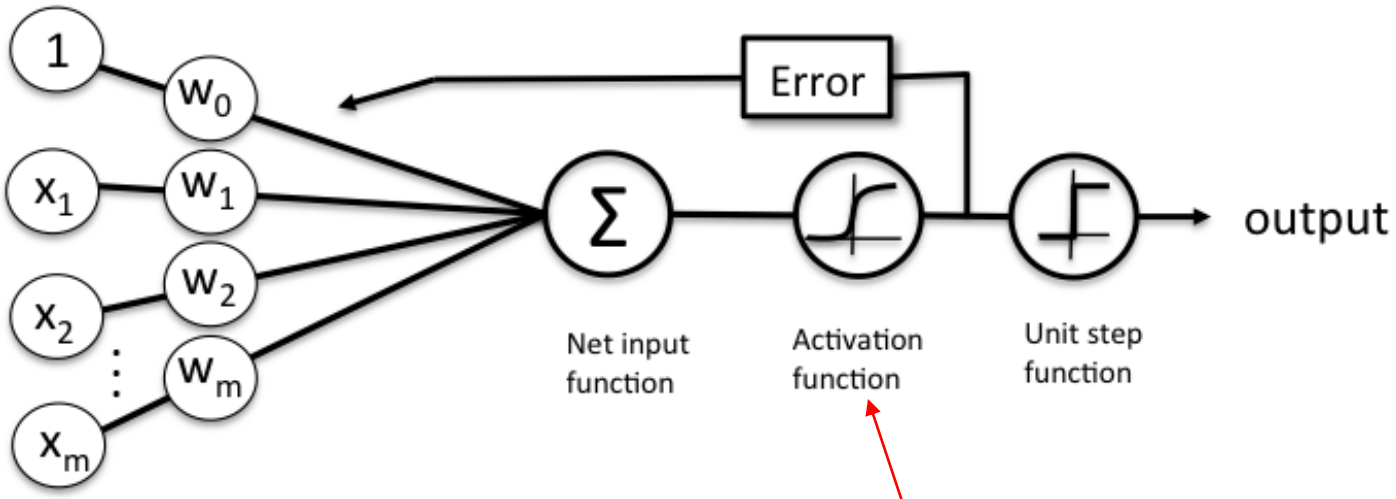
- Logistic Regression
- Support Vector Machine
- Decision Tree
- Random Forest
- kNN
- Naïve Bayes
- More ...

Logistic Regression
(로지스틱 회귀)

Logistic Regression

- 선형 바이너리 분류의 강력한 알고리즘 중 하나
- 로지스틱 회귀 분석은 일반 회귀 분석이 아니라 분류를 위한 모델임을 주의

Logistic Regression



Schematic of a logistic regression classifier.

활성화함수
= 순입력 함수(Net input function)의 리턴값과 실제 결과값을 비교하여 오차가 최소가 되도록 가중치 업데이트
→ Logistic Regression에서는 sigmoid 함수 사용

Odds Ratio

- Odds: 어떤 일이 일어날 승산
- Odds ratio: 특정 사건의 승산률
  - 어떤 특정 사건이 일어날 확률(*p*)과 일어나지 않을 확률의 비율(*1-p*)

$$odds\ ratio = \frac{p}{1-p}$$

Odds ratio 정의

↓

$$f(p) = log\frac{p}{1-p}$$

로그를 취한 함수 정의

Odds Ratio

CODING

$$z = \sum_j w_j x_j$$

순입력 함수의 리턴값을
z로 표기

↓

$$z = log \frac{p}{1-p}$$

z 값에 따라서 입력된 트레이닝
데이터 X가 어떤 집단에 속하는지
결정

↓

$$p = \frac{1}{1+e^{-z}}$$

p를 z에 관한 식으로 표현

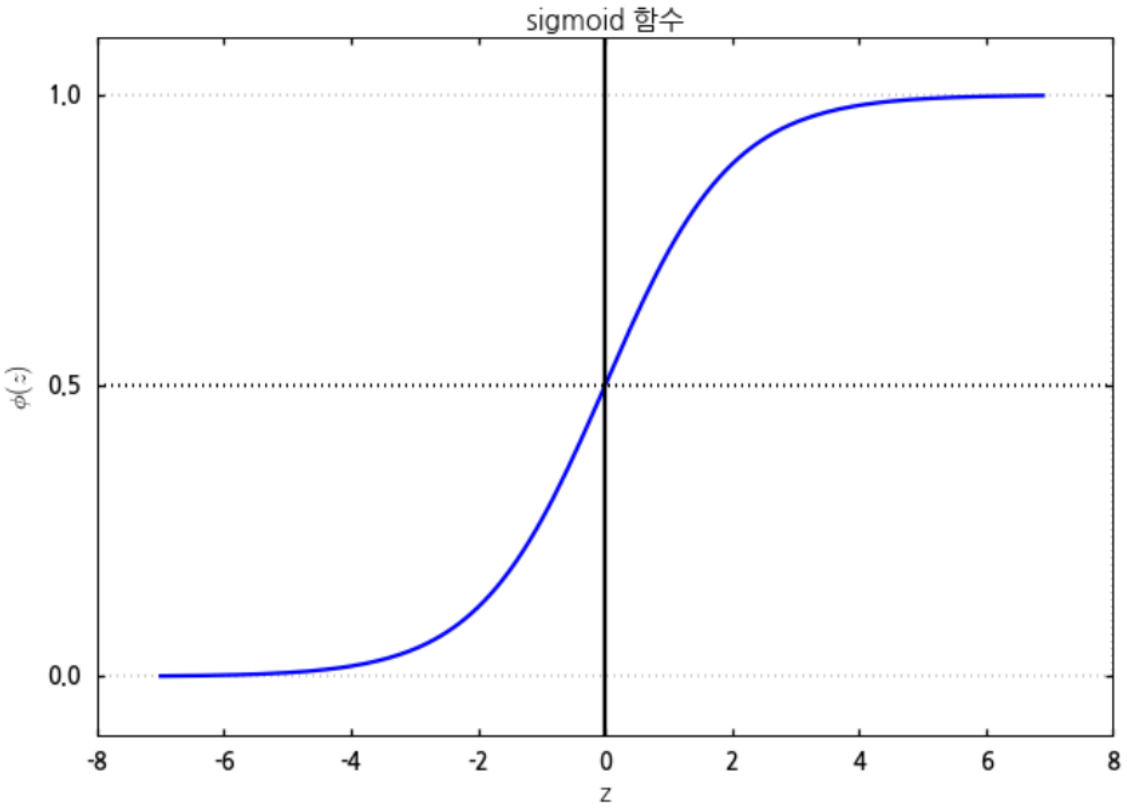
↓

$$\Phi(z) = \frac{1}{1+e^{-z}}$$

p를 z에 관한 함수로 표현

Logistic Regression

로지스틱 회귀는 순입력 함수의 리턴값에 대해 가중치 업데이트 여부를 결정하는 활성화 함수로 sigmoid 함수를 이용



Logistic Regression

CODING

Data preparation

```
iris = datasets.load_iris()
X = iris.data[:, [2,3]]
y = iris.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

```
sc = StandardScaler()
sc.fit(X_train) # get mean and sd of X_train
X_train_std = sc.transform(X_train) # training 데이터 표준화
X_test_std = sc.transform(X_test) # test 데이터 표준화
```

Logistic Regression

CODING

Logistic Regression

```
lr = LogisticRegression(C=1000.0, random_state=0)
lr.fit(X_train_std, y_train)
```

```
LogisticRegression(C=1000.0, class_weight=None, dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='ovr', n_jobs=1, penalty='l2', random_state=0,
          solver='liblinear', tol=0.0001, verbose=0, warm_start=False)
```

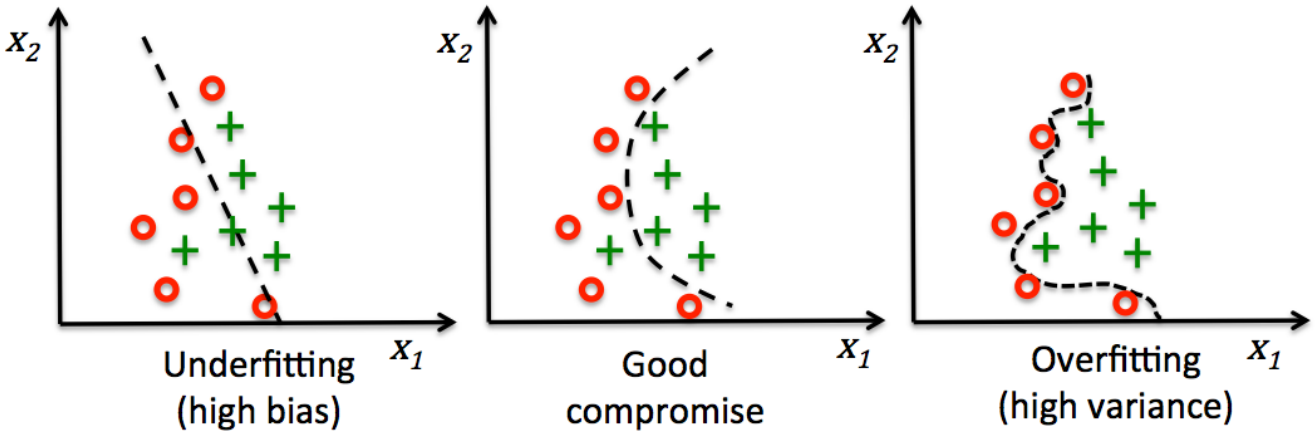
```
# X 테스트 데이터의 예측값 구하기
y_pred = lr.predict(X_test_std)
```

```
# 정답값과 예측값의 비교
accuracy_score(y_test, y_pred)
```

```
0.9666666666666667
```

Logistic Regression

- 평가
  - 계산비용이 적고, 구현하기 쉬우며, 결과 해석을 위한 지식 표현 쉬움
  - 선형 외의 경우 정확도가 낮을 수 있음
  - 언더피팅 경향이 있어서 정확도가 낮게 나올 수 있음



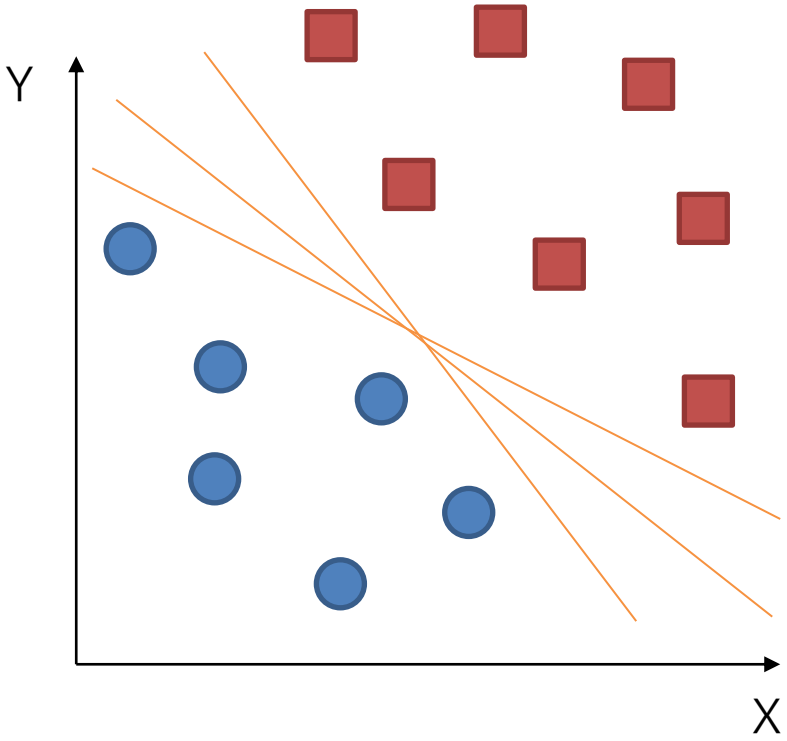
Support Vector Machine
(서포트 벡터 머신)

SVM

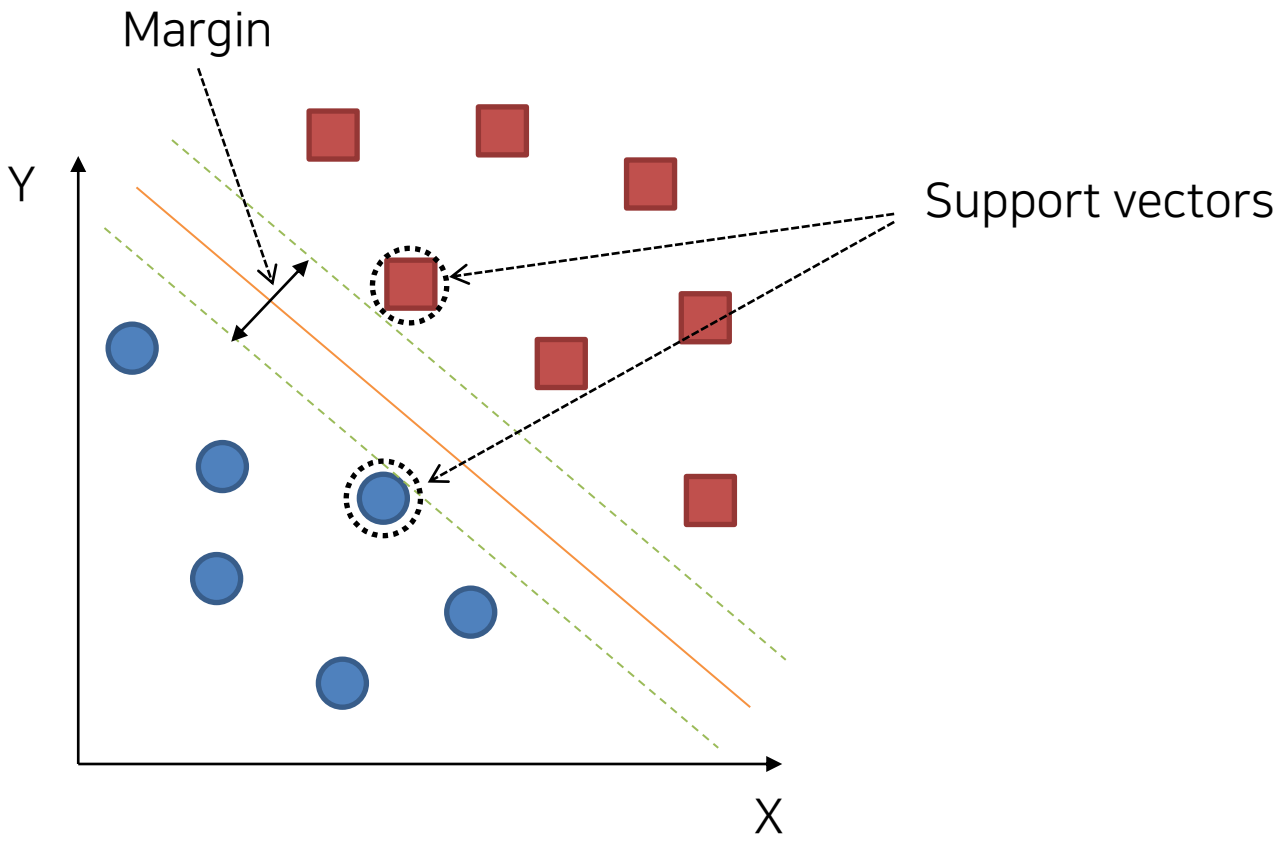
CODING

- SVM은 높은 인식 성능을 보여주는 지도학습의 대표적인 알고리즘
- SVM은 선을 구성하는 매개변수를 조정해서 요소들을 구분하는 선을 찾고, 이를 기반으로 패턴을 인식하는 방법

데이터 구분 선



마진을 최소화



SVM의 특징 ➔ 마진 최대화

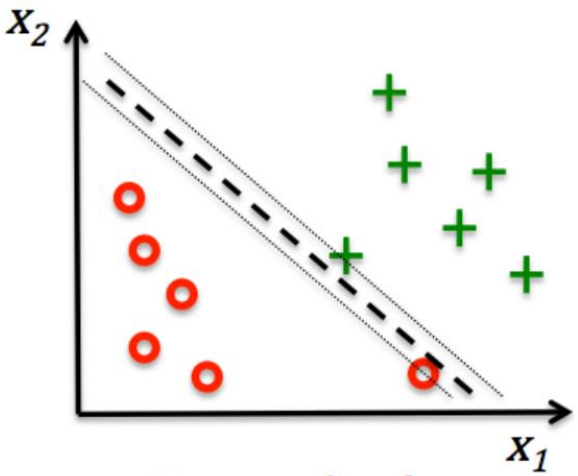
SVM

CODING

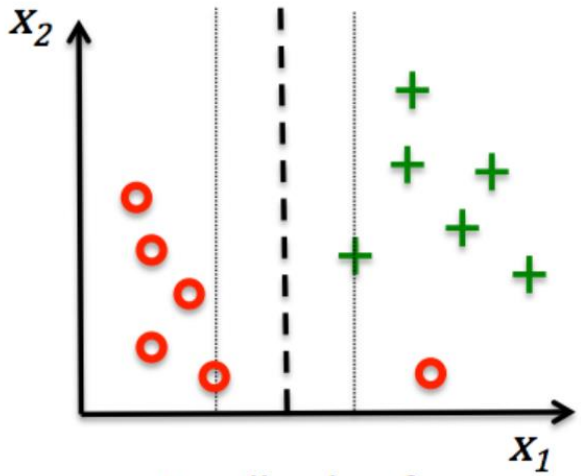
- sklearn 에서 크게 2가지 종류의 SVM 지원
  - SVC: 표준적으로 구현된 SVM
  - LinearSVC: 선형에 특화된 SVM
- Why SVC?
  - **C** represents '**C**lassification'

SVM

- sklearn의 SVM 에는 C 파라미터가 있음
- C 값에 따라서 경계선이 달라짐
  - 모델 overfitting or underfitting이 될 수 있음
  - 보통 [0.01, 0.1, 1.0, 10.0] 값을 많이 사용



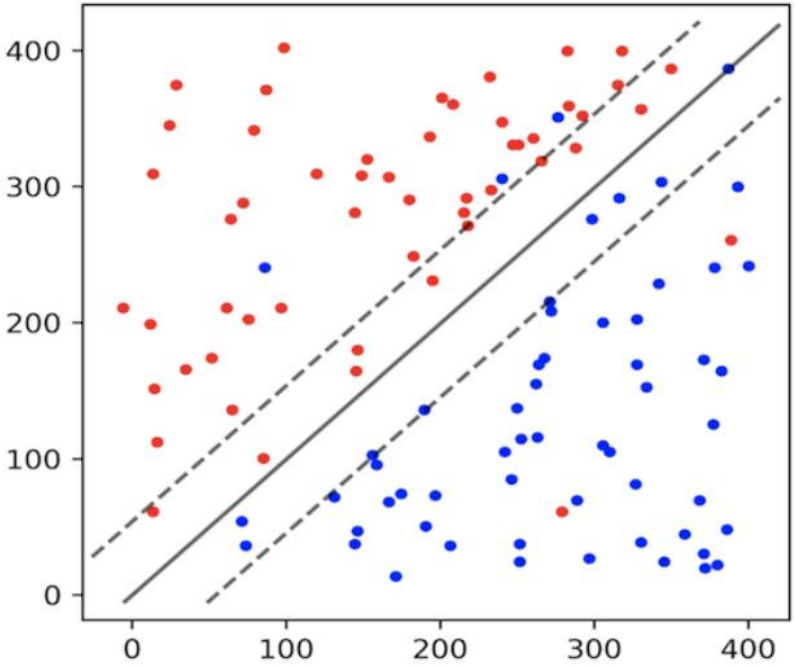
Large value for parameter *C*



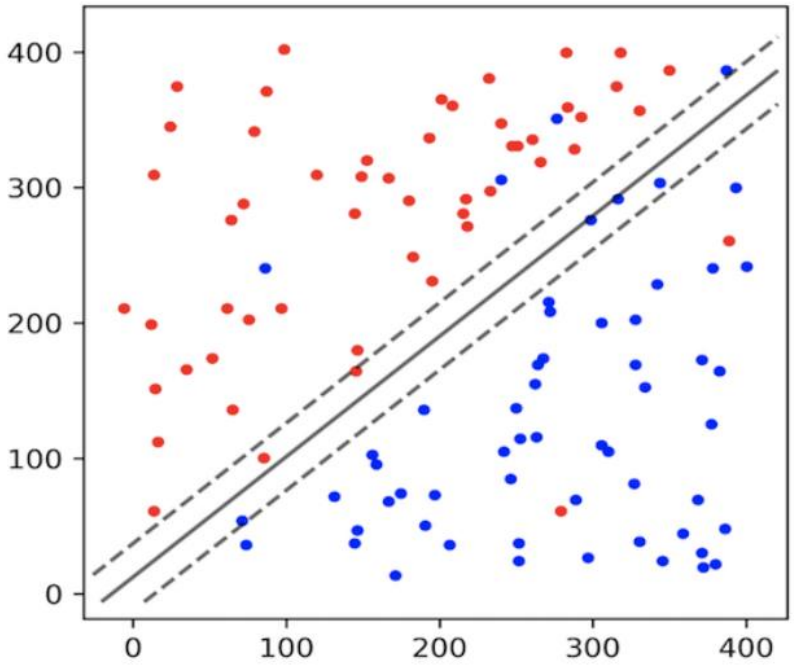
Small value for parameter *C*

SVM

SVM Parameter C



C = 1



C = 100

Support Vector Machine

```
from sklearn.svm import SVC
```

```
svm = SVC(kernel='linear', C=1.0, random_state=0)
svm.fit(X_train_std, y_train)
```

```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
  max_iter=-1, probability=False, random_state=0, shrinking=True,
  tol=0.001, verbose=False)
```

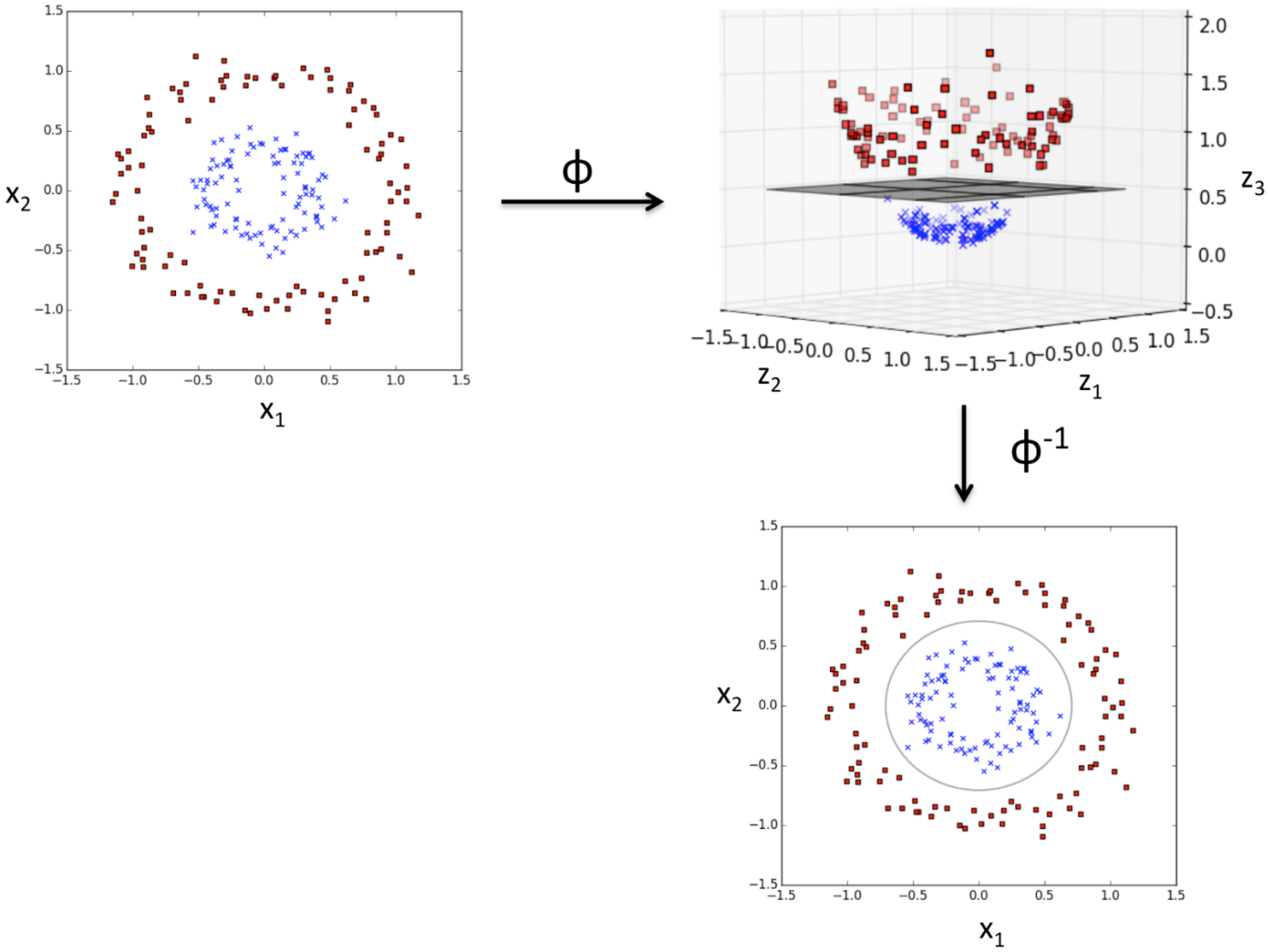
```
# X 테스트 데이터의 예측값 구하기
y_pred = svm.predict(X_test_std)

# 정답값과 예측값의 비교
accuracy_score(y_test, y_pred)
```

```
0.9666666666666667
```

SVM

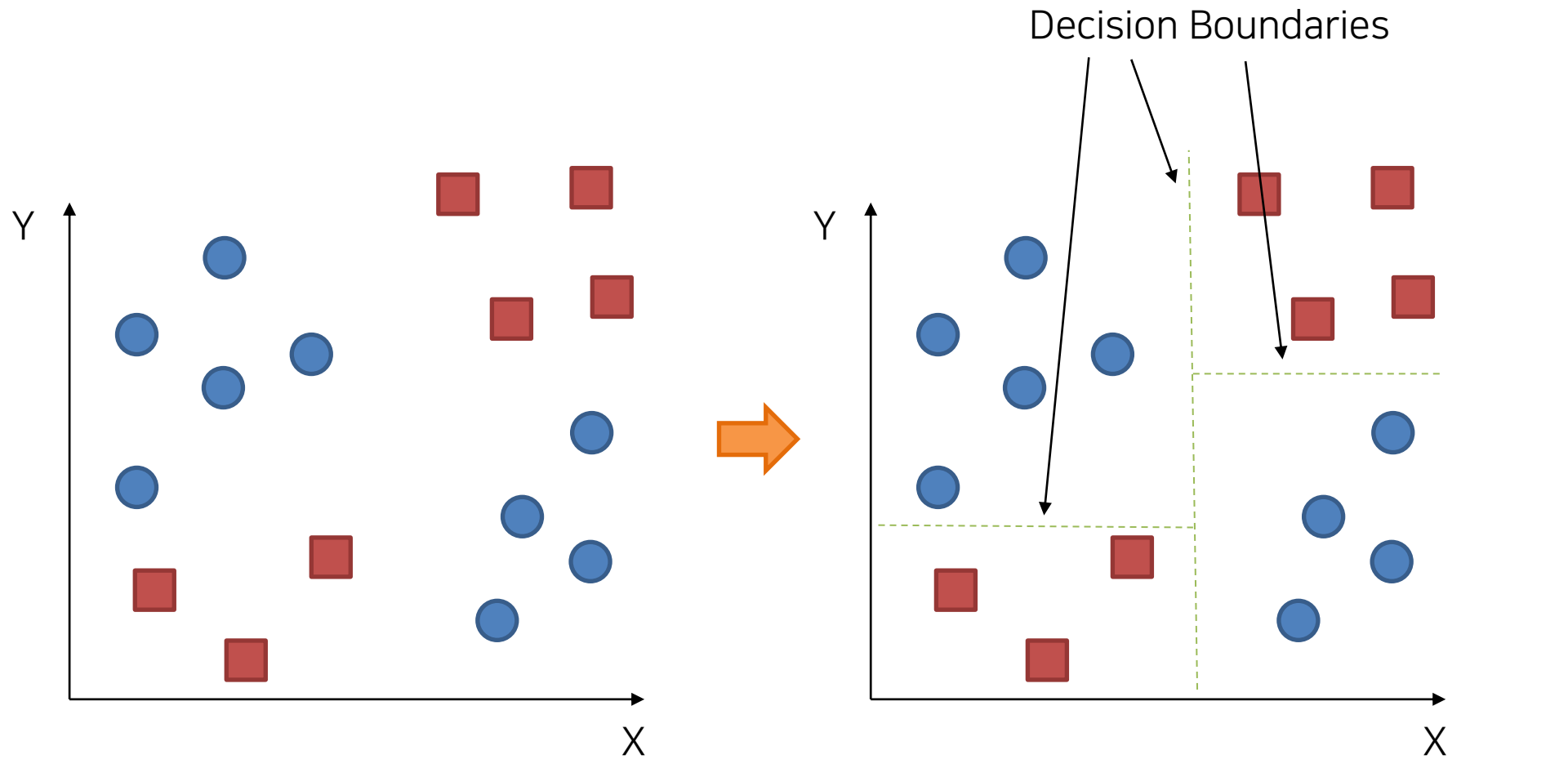
- 평가
  - 학습데이터의 오버피팅을 방지함
  - 비선형 데이터 분류 가능 (kernel method 활용)
  - 고차원 데이터에서 효율적
  - 학습 데이터의 margin이 적을 때 문제 발생 가능
  - Support vector (margin) 근처의 데이터만 고려

SVM 은 가능



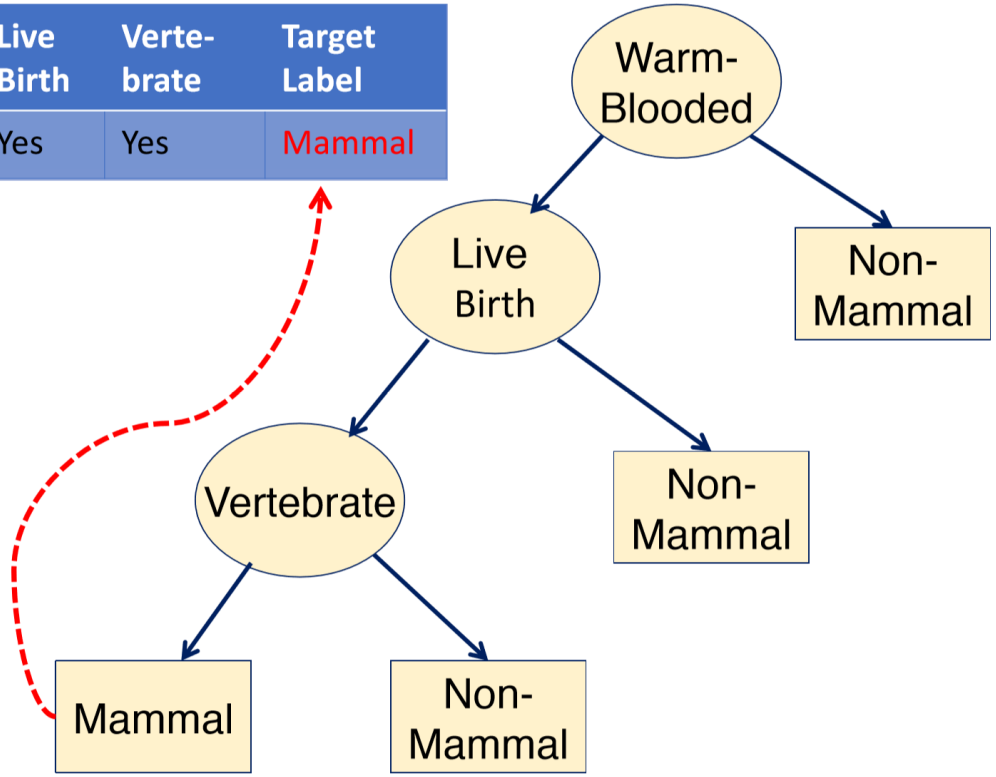
Decision Tree

Decision Tree



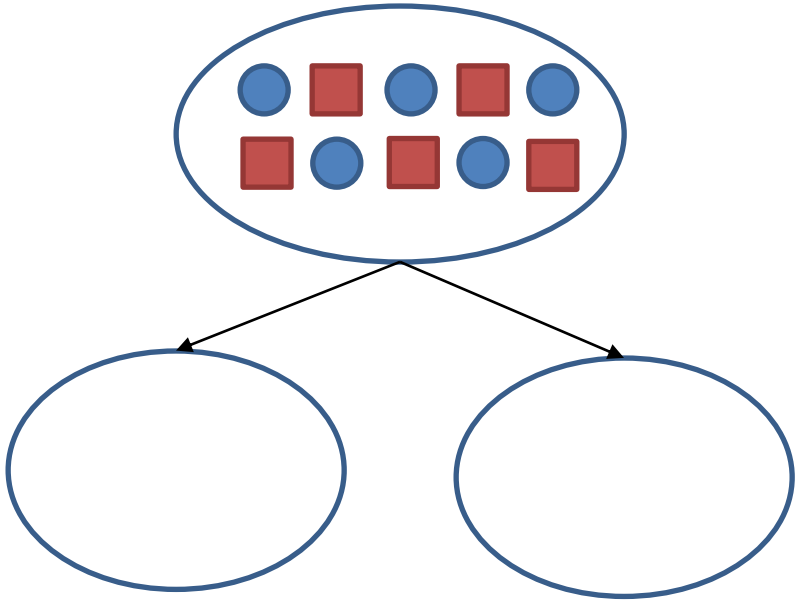
Decision Tree

Warm-Blooded	Live Birth	Verte-brate	Target Label
Yes	Yes	Yes	Mammal

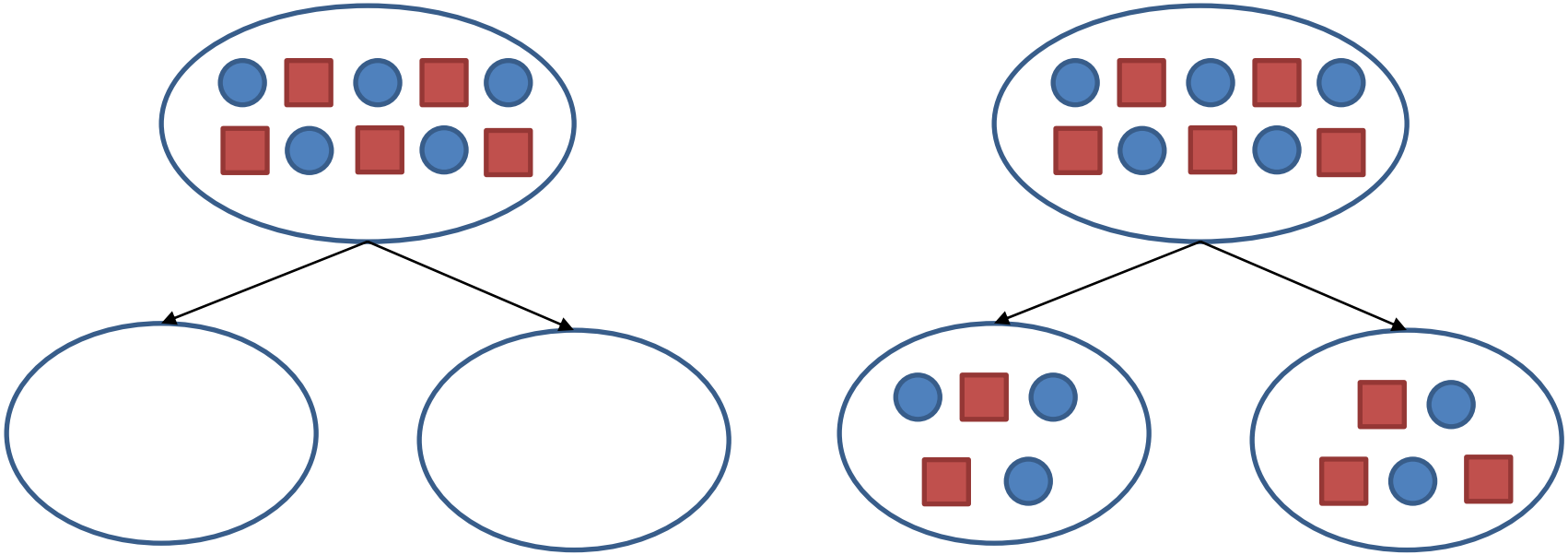


Decision Tree

- Start with all samples at a node (모든 샘플에서 시작)
- Partition samples based on input to create purest subsets (샘플 나누기 ➔ 기준: 불순도)
- Repeat to partition data into successively purer subsets



Greedy 접근

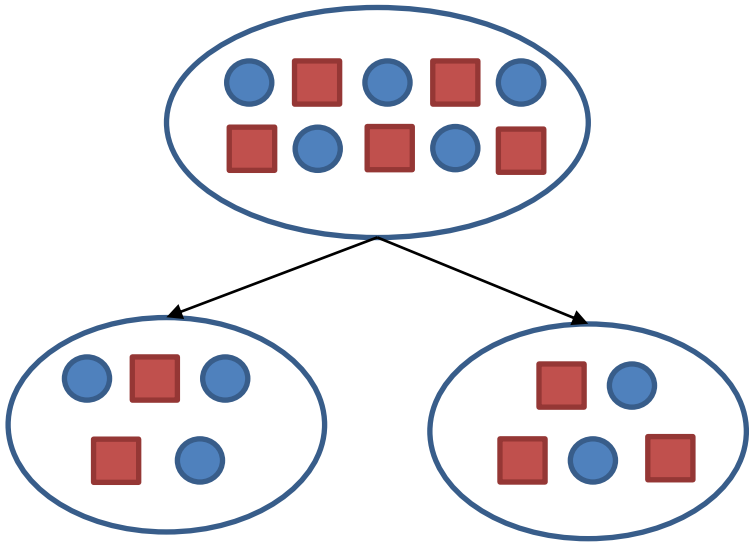


What's the best way to split the current node?

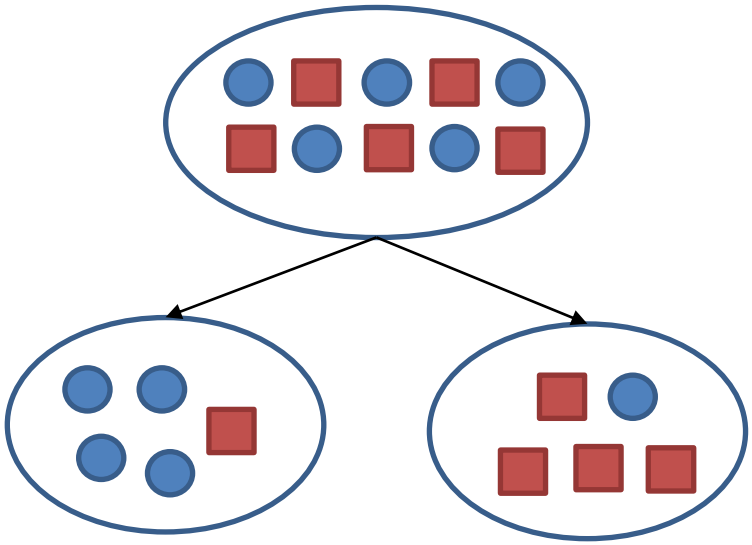
최적의 구분

CODING

We want subsets to be as homogeneous as possible



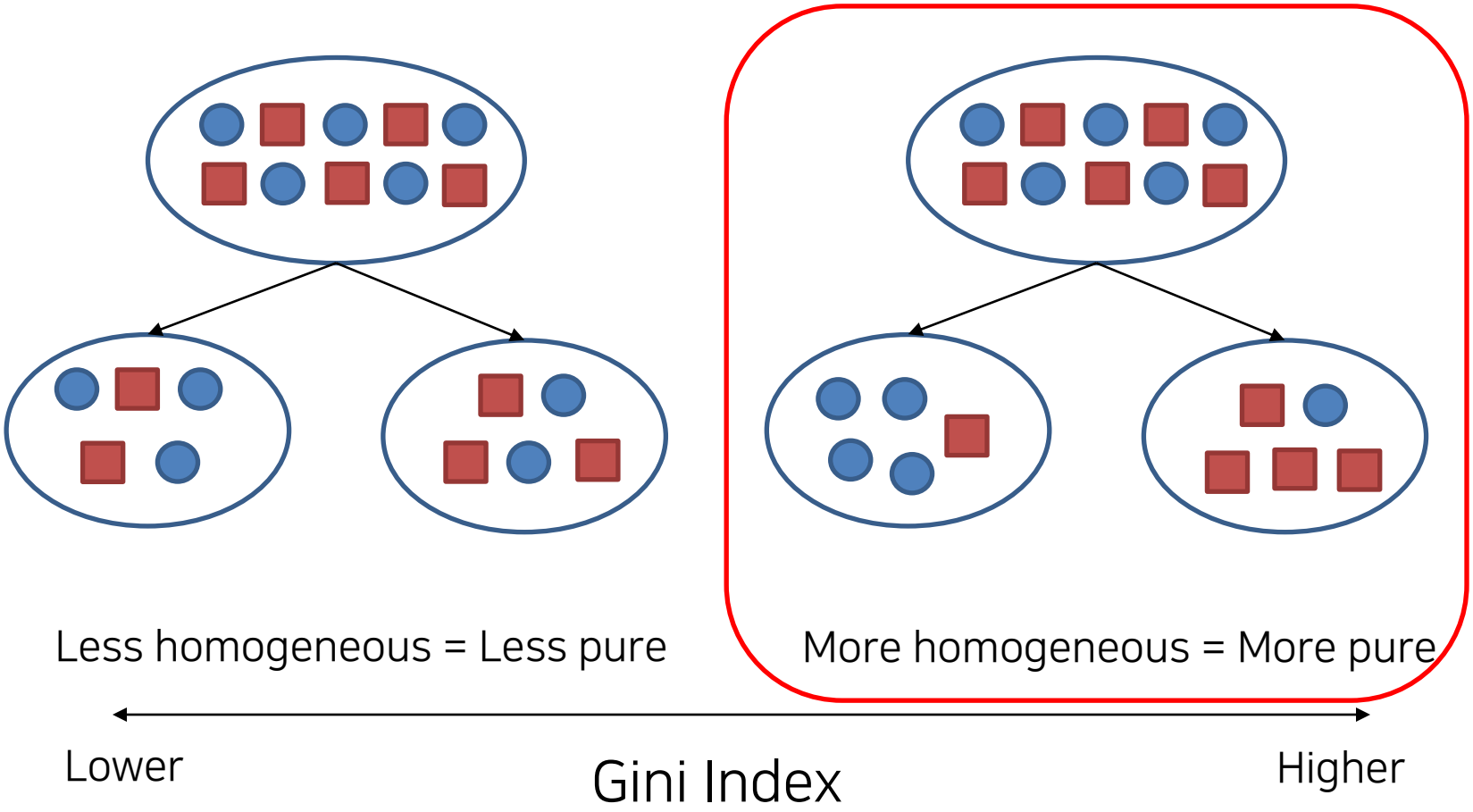
Less homogeneous = Less pure



More homogeneous = More pure

최적의 구분

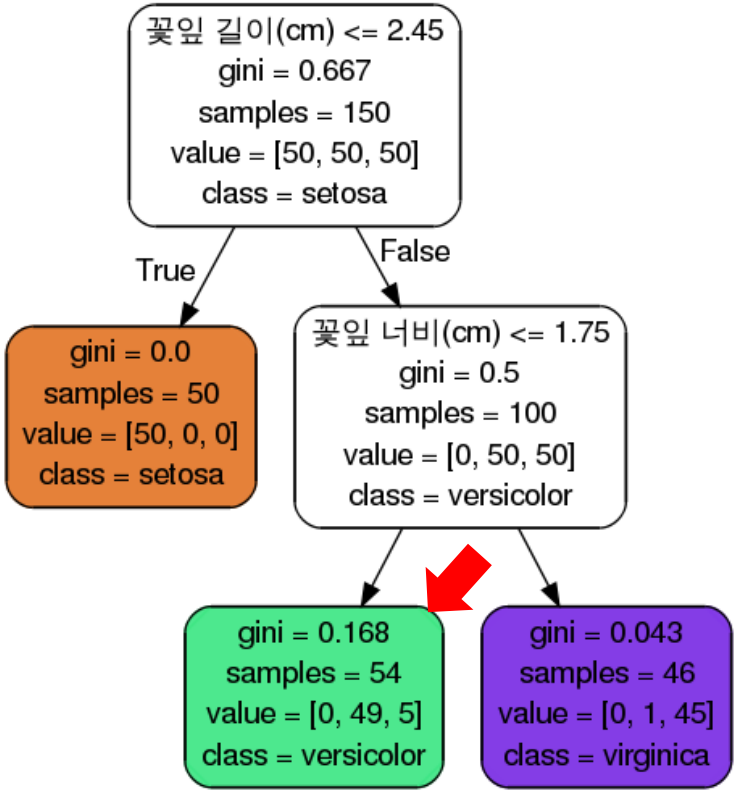
We want subsets to be as homogeneous as possible



Gini (지니) 계수

$$G_i = 1 - \sum_{k=1}^{n} {P_{i,k}}^2$$

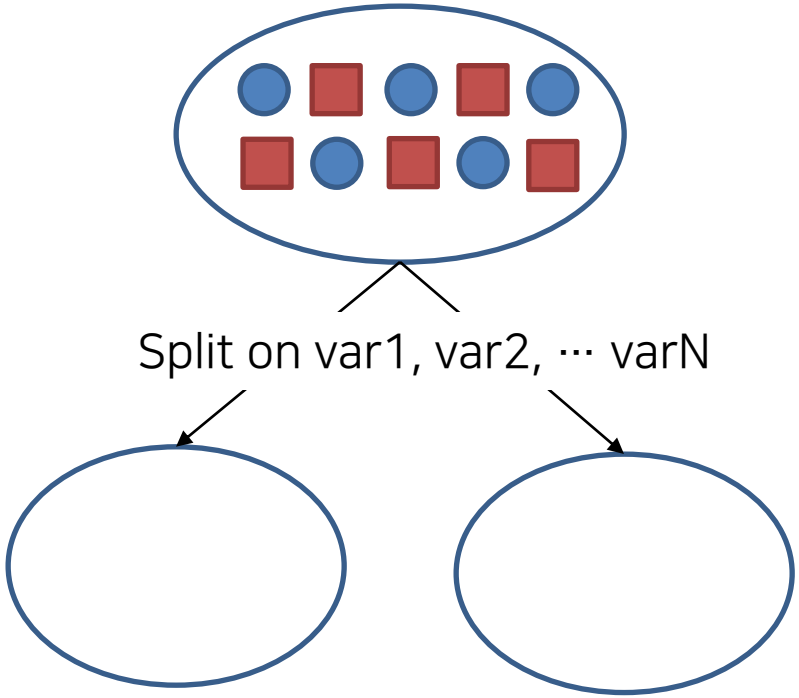
$P_{i,k}$ 는 i 번째 노드에 있는 훈련 샘플 중 클래스 k 에 속하는 샘플의 비율



Gini 점수 = $1 - (0/54)^2 - (49/54)^2 - (5/54)^2$
≈ 0.168

어떤 변수를 구분?

Splits on all variables are tested



Can take a lot of time

언제까지 구분?

- 특정 비율 만큼 샘플이 들어가는 경우: All (or X% of) samples have same class label
- 한 그룹당 최소 샘플 갯수가 채워진 경우: Number of samples in node reaches minimum
- 특정 기준 보다 불순도가 낮아진 경우: Change in impurity measure is smaller than threshold
- 최대 트리 깊이에 도달한 경우: Max tree depth is reached
- Others …

Decision Tree

CODING

- 평가
  - Tree를 만드는 과정이 직관적이고 이해하기 쉽다
  - 데이터 전처리 과정이 필수는 아니다
  - Greedy 접근법은 항상 Best Solution을 제공하지 않는다
  - 직선 boundary만 가능하다

Decision Tree

CODING

Decision Tree

```
from sklearn.tree import DecisionTreeClassifier
```

```
dt = DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state=0)
dt.fit(X_train_std, y_train)
```

```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=3,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=0,
            splitter='best')
```

```
# X 테스트 데이터의 예측값 구하기
y_pred = dt.predict(X_test_std)

# 정답값과 예측값의 비교
accuracy_score(y_test, y_pred)
```

```
0.96666666666666667
```