BITAmin

10주차 정규Session

비타민 6기 2조

이수진 김언지 홍진수 이상연

CONTETNS

1.분류모델

분류모델

분류 및 예측을 위한 모델의 일반화를 위한 두 가지 접근 법
<The true goal is to perform well on new instances.>

Model-based Learning (모델 기반 학습)

데이터로 모델을 만들고 새로운 데이터에 대한 분류/예측을 진행
훈련 데이터를 이용해 모델의 파라미터를 찾고 모델이 만들어진 후 훈련 데이터는 불필요
모델 생성 후엔 새로운 데이터에 대한 분류/예측 빠름

- 선형/비선형모델 (ex. 선형회귀, 로지스틱 회귀
- Neural network
- 의사결정나무
- Support vector machine (SVM)

KNN, SVM 모두 분류뿐만 아니라 예측 목적으로도 활용 가능하지만
이번 시간엔 '분류'모델을 살펴볼 것

Instance-based Learning (사례 기반 학습)

별도의 모델 생성 없이 인접 데이터를 분류/예측에 사용
모델을 만들 필요 없이 새로운 데이터에 대한 분류/예측을 위해 훈련 데이터셋을 곧바로 활용

- K-nearest neighbor (KNN)

2. K 최근접 이웃 (K-Nearest Neighbor)

K 최근접 이웃 (K-Nearest Neighbor)

<유유상종> 알고리즘

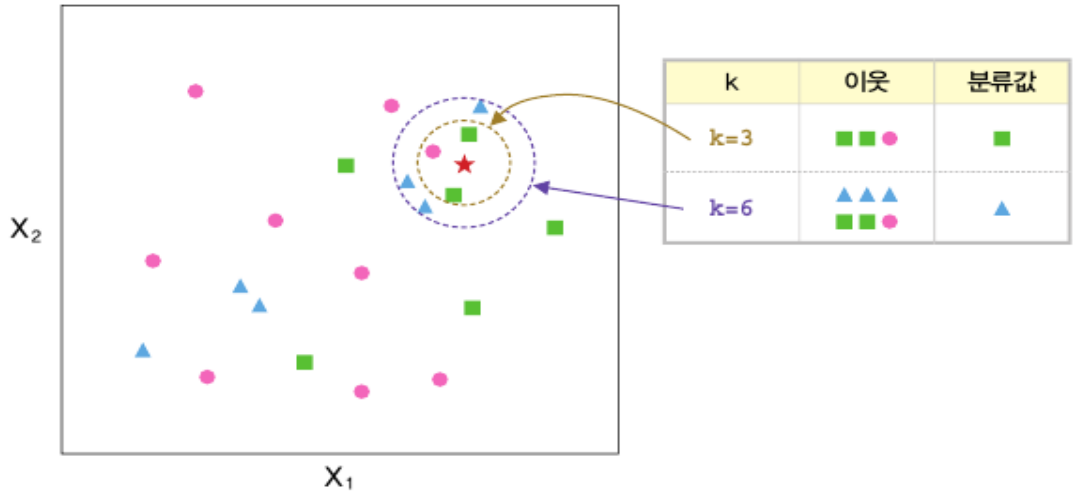
유유상종: 같은(Nearest) 무리(Neighbor)끼리 사로 사귐 (비슷한 애들끼리 몰려있다)

K: '이웃의 수'
Nearest: '거리'

새로운 데이터가 들어왔을 때 해당 데이터와 거리가 가까운 데이터들의 특성을 기반으로 분류

	X_1	X_2	X_3	X_4	X_5	X_6	⋯	X_n	Y
0	4	1	9	0	0.05	M	⋯	60	0
1	6	1	22	0	0.6	F	⋯	55	1
2	3	1	73	1	0.75	F	⋯	77	1
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
100	3	0	43	0	0.01	M	⋯	32	0
	2	1	33	1	0.5	M	⋯	29	?

[그림 7-5] 분류 알고리즘

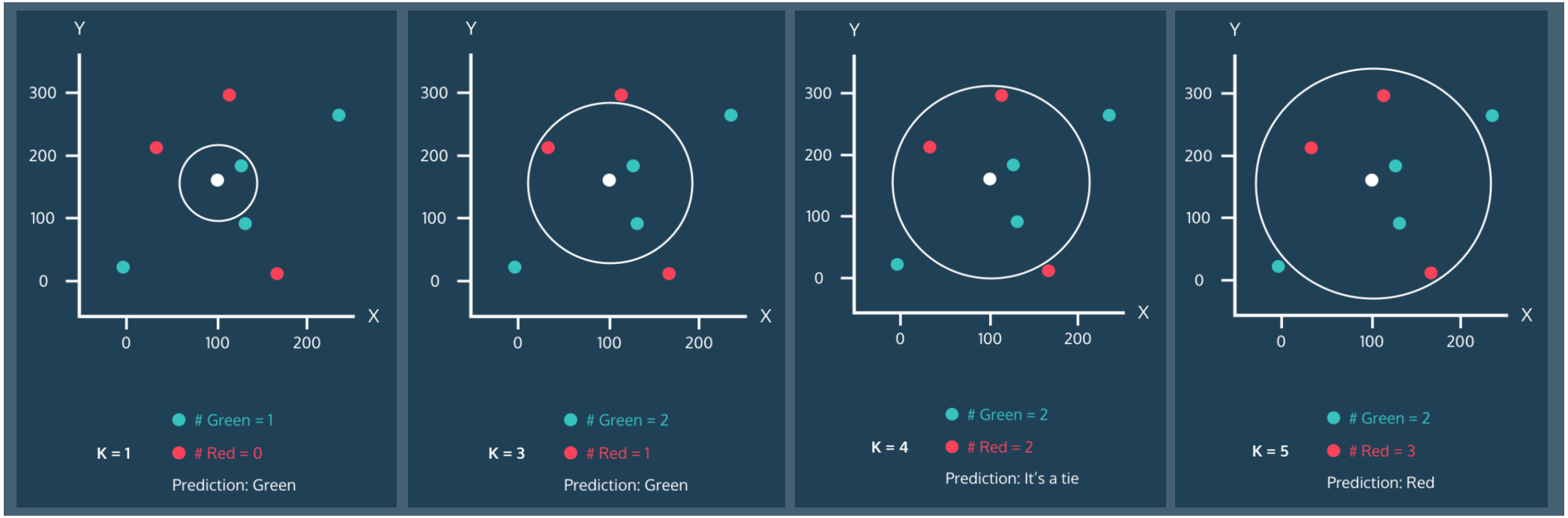


k	이웃	분류값
k=3	■■●	■
k=6	▲▲▲ ■■●	▲

[그림 7-6] KNN 모형

K 최근접 이웃 (K-Nearest Neighbor) 작동방식

1. 분류할 관측치 x 선택 : **test data**, 새로운 data를 의미
2. X로부터 인접한 k개의 **학습 데이터**를 탐색
3. 탐색된 k개 학습 데이터의 **majority class 를 정의**
4. majority class를 x의 **분류결과로 반환**



http://hleecaster.com/ml-knn-concept/

K 최근접 이웃 (K-Nearest Neighbor) 특징

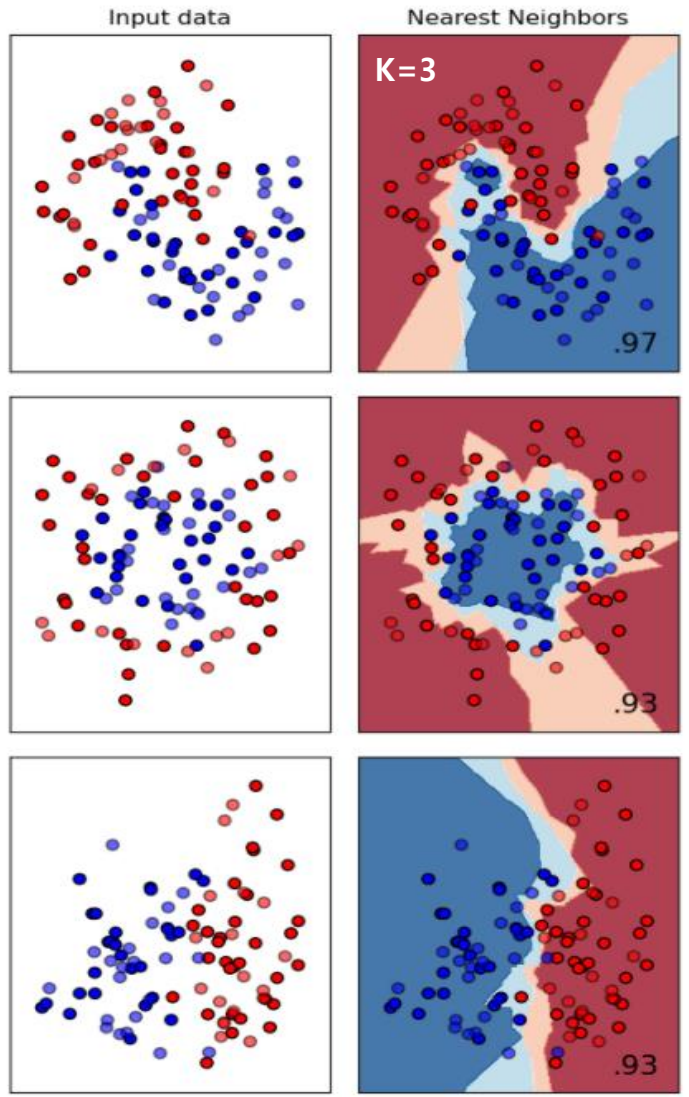
Instance-based Learning
- 각각의 관측치 (instance)만을 이용해 새로운 데이터 예측 진행

Lazy learning
- 모델을 별도로 학습하지 않고 테스팅 데이터가 들어와야 비로소 작동하는 게으른 알고리즘 (단순하고 효율적)

KNN은 비선형 형태로 결정 경계선이 나타나게 됨

거리계산이 이루어지기 때문에 연속형 변수들의 단위 차이가 크면 정규화 진행

명목형 변수의 경우 더미변수로 변환



https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html#sphx-glr-auto-examples-classification-plot-classifier-comparison-py

K 최근접 이웃 (K-Nearest Neighbor) 활용 사례

패턴 인식 분야에서 많이 사용

유전자 데이터 패턴 식별, 질병진단

스펨 메일 필터링, 개인영화추천

글자/ 얼굴인식

KNN 하이퍼파라미터

sklearn.neighbors.KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2, metric='minkowski', metric_params=None, n_jobs=None)

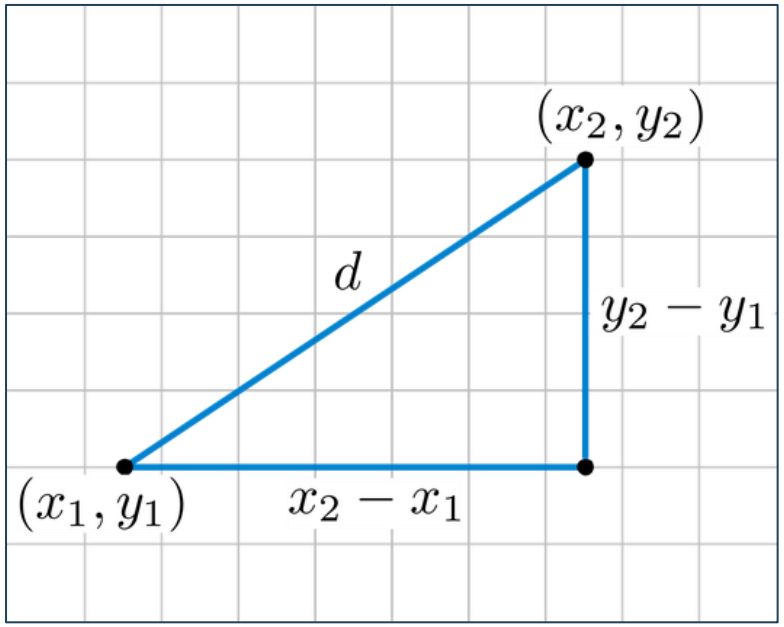
n_neighbors	k를 의미 (int가 들어옴)
weights	‘uniform’: 모든 이웃에 대해 동일한 가중치 부여(즉, 거리에 따른 가중치 없음) ‘distance’: 거리의 역수를 이용해 거리가 가까운 이웃에 대해 더 큰 가중치를 부여 혹은 개인이 직접 설정도 가능
Algorithm	‘auto’: 들어온 데이터 기반 가장 적합한 알고리즘으로 자동 선택 ‘ball_tree’, ‘kd_tree’, ‘brute’
P	Power parameter for the Minkowski metric (1이면 맨해튼 거리, 2면 유클리드 거리와 동일)
Metric	거리 척도

https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html#sklearn.neighbors.KNeighborsClassifier.fit

다양한 거리 개념

유클리드 거리 (Euclidean Distance)

두 지점의 직선이자 최단거리

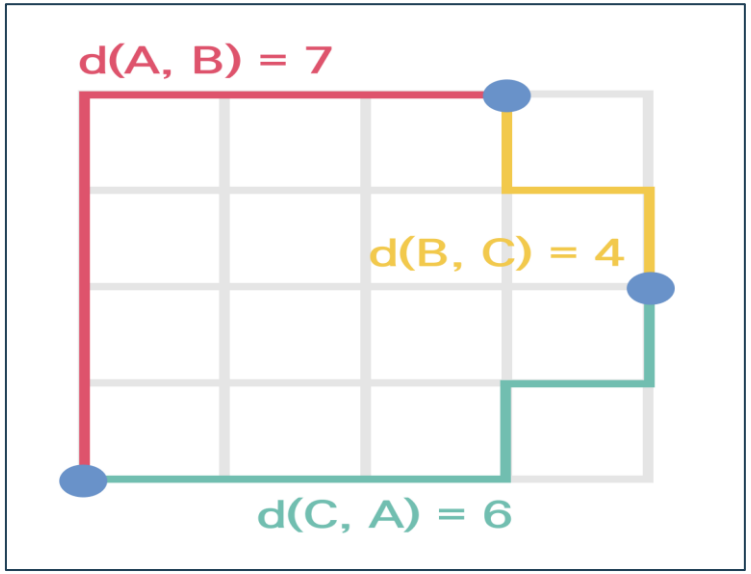


$$d(x,y) = \sqrt{(x_1 - y_1)^2 + \ldots + (x_p - y_p)^2}$$

$$= \sqrt{(x-y)'(x-y)}$$

맨해튼 거리 (Manhattan Distance / Taxicab Distance)

직선 거리가 아닌 격자 형태의 거리
항상 맨해튼>= 유클리드 거리

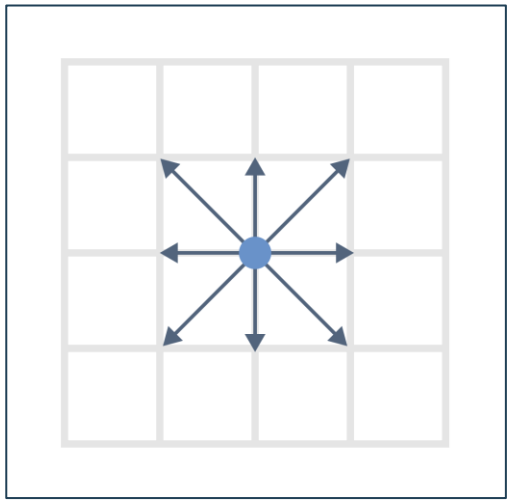


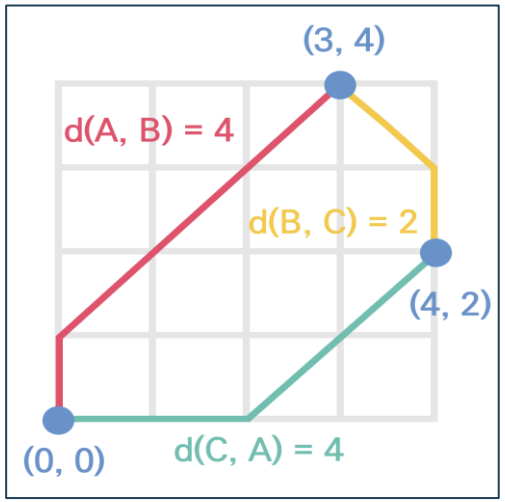
$$d(x,y) = \sum_{i=1}^{p} |x_i - y_i|$$

https://dimenchoi.tistory.com/81 https://tomining.tistory.com/35

다양한 거리 개념

체비셰프 거리
(Chebyshev Distance)





$$d(x,y) = \max_i |x_i - y_i|$$

민코프스키 거리
(Minkowski Distance)

이전의 거리 지표들을 일반화한 것

n 차원 점 X, Y에 대해 p차 민코프스키 거리는 다음과 같음

p = 1일 경우 맨해튼 거리와 동일 (L1 norm이라고도 함)
p = 2일 경우 유클리드 거리와 동일 (L2 norm이라고도 함)
p = ∞일 경우 체비쇼프 거리와 동일 (L max norm이라고도 함)

$$D(X,Y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{1/p}$$

https://tomining.tistory.com/35

https://goofcode.github.io/similarity-measure

다양한 거리 개념

마할라노비스 거리

인도 통계학자 마할라노비스

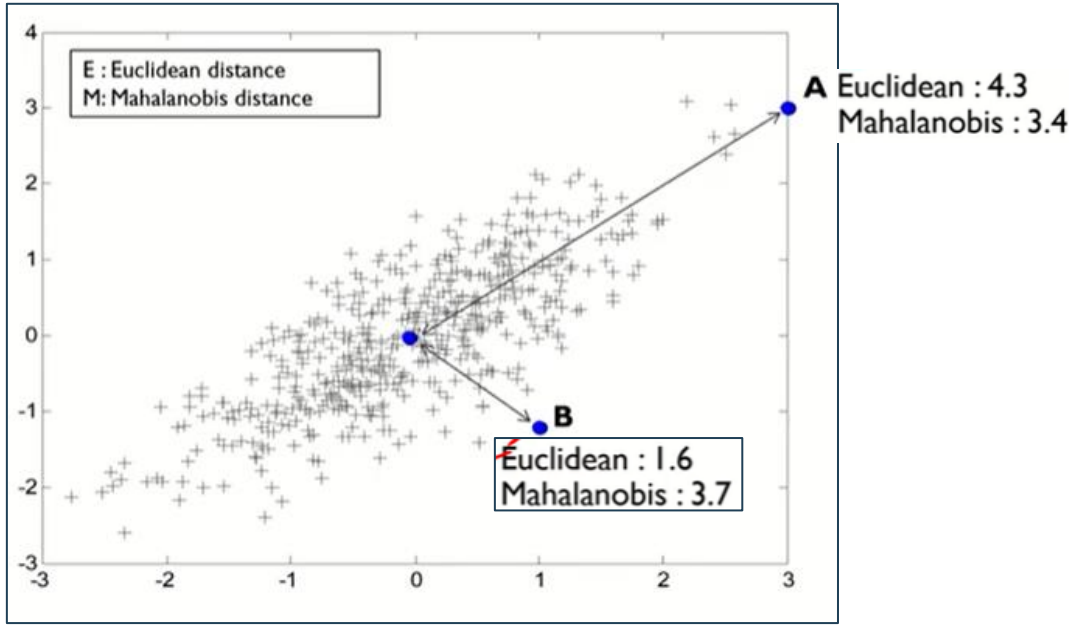
변수 내 분산, 변수 간 공분산을 고려한 거리로 거리가 타원의 형태를 띄게됨, S는 공분산 행렬

S가 단위행렬(공분산=0, 분산=1)이면 거리는 유클리드 거리와 동일해짐

$$d(x,y) = \sqrt{(x-y)'S^{-1}(x-y)}$$

$$S = \begin{pmatrix} \sigma_{11} & \sigma_{12} & \sigma_{13} \\ \sigma_{21} & \sigma_{22} & \sigma_{23} \\ \sigma_{31} & \sigma_{32} & \sigma_{33} \end{pmatrix}$$

이외에도 Correlation Distance, Spearman Rank Correlation Distance, Cosine Distance 도 존재



https://tomining.tistory.com/35 https://www.youtube.com/watch?v=W-DNu8nardo

KNN 하이퍼파라미터 metric

class sklearn.neighbors.DistanceMetric

identifier	class name	args	distance function
“euclidean”	EuclideanDistance	•	`sqrt(sum((x - y)^2))`
“manhattan”	ManhattanDistance	•	`sum(|x - y|)`
“chebyshev”	ChebyshevDistance	•	`max(|x - y|)`
“minkowski”	MinkowskiDistance	p	`sum(|x - y|^p)^(1/p)`
“wminkowski”	WMinkowskiDistance	p, w	`sum(|w * (x - y)|^p)^(1/p)`
“seuclidean”	SEuclideanDistance	V	`sqrt(sum((x - y)^2 / V))`
“mahalanobis”	MahalanobisDistance	V or VI	`sqrt((x - y)' V^-1 (x - y))`

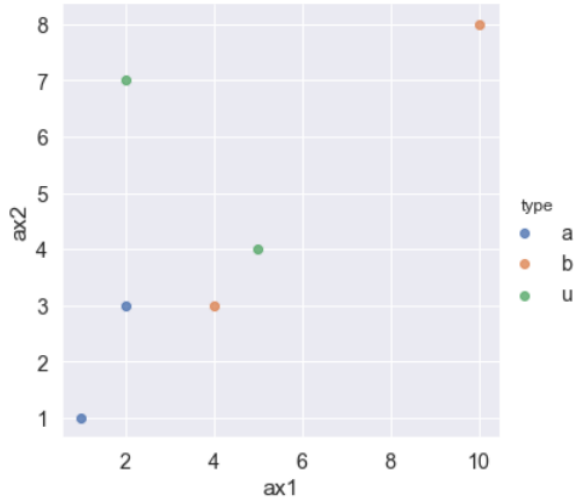
https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.BallTree.html#sklearn.neighbors.BallTree

KNN 코드 예시

kneighbors(X=None, n_neighbors=None, return_distance=True)
입력한 데이터와 이웃들의 거리와 인덱스를 반환

```
from sklearn.neighbors import KNeighborsClassifier
import numpy as np
import pandas as pd

# Packages for visuals
import matplotlib.pyplot as plt
import seaborn as sns; sns.set(font_scale=1.3)
%matplotlib inline

import warnings
warnings.filterwarnings('ignore')


# 2차원 공간에 4개 train data 설정
x = pd.DataFrame([[1,1],[2,3],[4,3],[10,8]], columns=['ax1','ax2'])

# train data의 label 설정
y = pd.DataFrame(['a','a','b','b'], columns=['type'])

# 3-NN classifier
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(x, y)

# test data
x_test=pd.DataFrame([[2,7],[5,4]], columns=['ax1','ax2'])
pred = knn.predict(x_test)
print('분류:',pred,'₩n')

# test data와 train data와의 거리
print(knn.kneighbors(x_test,4))

from math import sqrt
print('test data(2,7)와 (10,8)data와의 거리: ',sqrt((2-10)**2+(7-8)**2))
sample = pd.concat([x.join(y), x_test])
sample.reset_index(inplace=True)
sample.drop('index',axis=1,inplace=True)
sample.fillna('u',inplace =True)

sns.lmplot('ax1','ax2', data=sample, hue='type',fit_reg=False);

print('₩n')
print(knn.predict_proba(x_test))
```

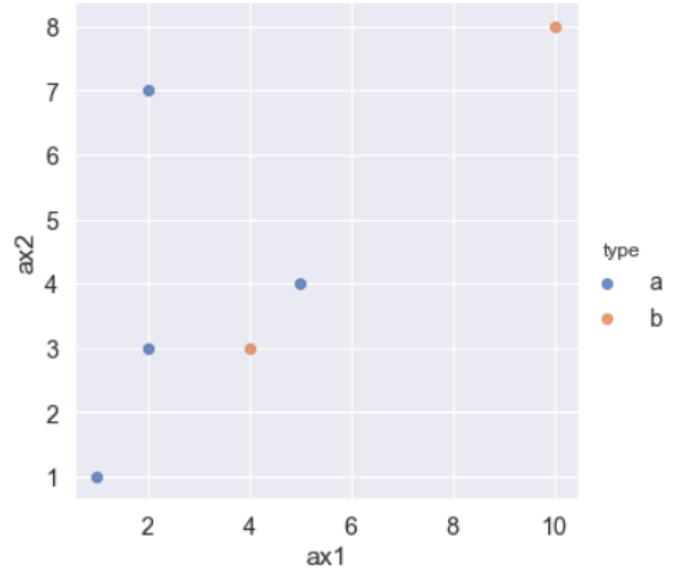
```
분류: ['a' 'a']

(array([[4.        , 4.47213595, 6.08276253, 8.06225775],
       [1.41421356, 3.16227766, 5.        , 6.40312424]]), array([[1, 2, 0, 3],
       [2, 1, 0, 3]], dtype=int64))
test data(2,7)와 (10,8)data와의 거리: 8.06225774829855


[[0.66666667 0.33333333]
 [0.66666667 0.33333333]]
```



KNN 코드 예시

```python
# 3-NN classifier
# 맨해튼 거리
knn = KNeighborsClassifier(n_neighbors=3, weights='uniform', metric='manhattan')
knn.fit(x, y)

# test data
x_test=pd.DataFrame([[2,7],[5,4]], columns=['ax1','ax2'])
pred = knn.predict(x_test)
print('분류:',pred,'₩n')

# test data와 train data와의 거리
print(knn.kneighbors(x_test,4))

sample = pd.concat([x.join(y), x_test.join(pd.DataFrame(pred,columns=['type']))])
sample.reset_index(inplace=True)
sample.drop('index',axis=1,inplace=True)
sample.fillna('u',inplace =True)

sns.lmplot('ax1','ax2', data=sample, hue='type',fit_reg=False);

print('₩n')
print(knn.predict_proba(x_test))
```

```
분류: ['a' 'a']

(array([[4., 6., 7., 9.],
       [2., 4., 7., 9.]]), array([[1, 2, 0, 3],
       [2, 1, 0, 3]], dtype=int64))


[[0.66666667 0.33333333]
 [0.66666667 0.33333333]]
```



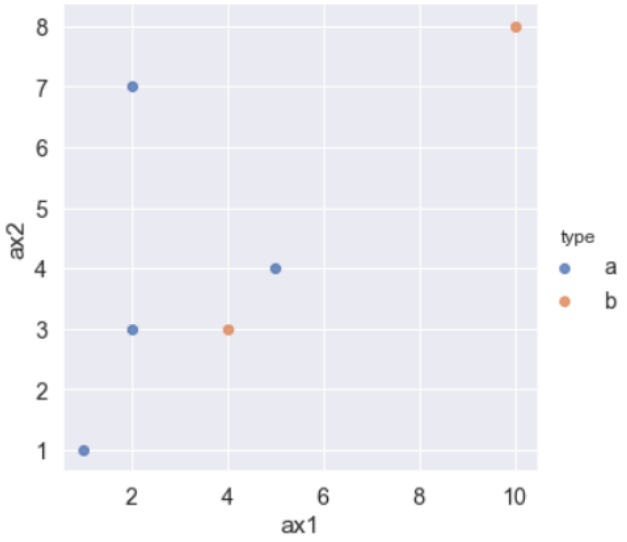
앞의 유클리드 거리로 했을 때와 결과 동일

test 데이터의 분류 결과도 시각화에 반영

KNN 코드 예시

```
# 2-NN classifier
knn = KNeighborsClassifier(n_neighbors=2)
knn.fit(x, y)

# test data
x_test=pd.DataFrame([[2,7],[5,4]], columns=['ax1','ax2'])
pred = knn.predict(x_test)
print('분류:',pred,'₩n')

# test data와 train data와의 거리
print(knn.kneighbors(x_test,4))

sample = pd.concat([x.join(y), x_test.join(pd.DataFrame(pred,columns=['type']))])
sample.reset_index(inplace=True)
sample.drop('index',axis=1,inplace=True)
sample.fillna('u',inplace =True)

sns.lmplot('ax1','ax2', data=sample, hue='type',fit_reg=False);

print('₩n')
print(knn.predict_proba(x_test))
```

```
분류: ['a' 'a']

(array([[4.        , 4.47213595, 6.08276253, 8.06225775],
       [1.41421356, 3.16227766, 5.        , 6.40312424]]), array([[1, 2, 0, 3],
       [2, 1, 0, 3]], dtype=int64))


[[0.5 0.5]
 [0.5 0.5]]
```

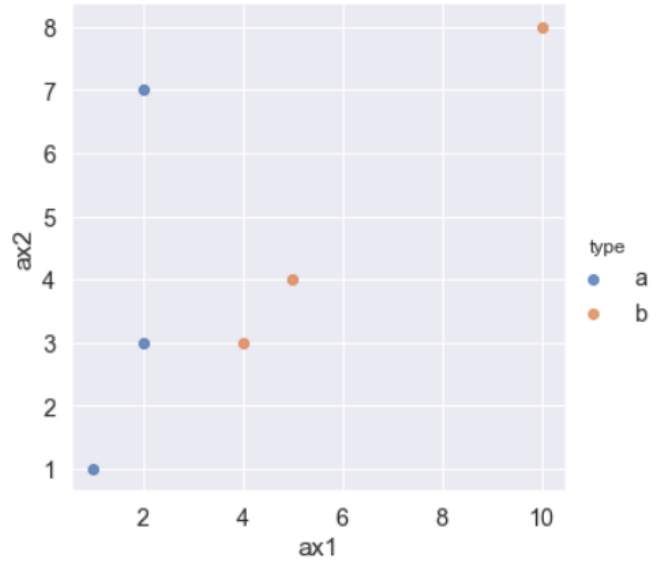


K=2, 샘플 데이터 둘 다 인덱스가 1인 (2,3) ‘a’ , 2인 (4,3) ’b’ 데이터를 이웃으로 인식해 동률(0.5)이지만 ‘a’ type으로 분류

KNN 코드 예시

```
# 2-NN classifier
# 가중치 부여
knn = KNeighborsClassifier(n_neighbors=2, weights='distance')
knn.fit(x, y)

# test data
x_test=pd.DataFrame([[2,7],[5,4]], columns=['ax1','ax2'])
pred = knn.predict(x_test)
print('분류:',pred,'₩n')

# test data와 train data와의 거리
print(knn.kneighbors(x_test,4))

sample = pd.concat([x.join(y), x_test.join(pd.DataFrame(pred,columns=['type']))])
sample.reset_index(inplace=True)
sample.drop('index',axis=1,inplace=True)
sample.fillna('u',inplace =True)

sns.lmplot('ax1','ax2', data=sample, hue='type',fit_reg=False);

print('₩n')
print(knn.predict_proba(x_test))

ap = 1/4
bp = 1/4.47213595
n = 1/(ap+bp)
n
print(ap*n,bp*n)
```

```
분류: ['a' 'b']

(array([[4.        , 4.47213595, 6.08276253, 8.06225775],
       [1.41421356, 3.16227766, 5.        , 6.40312424]]), array([[1, 2, 0, 3],
       [2, 1, 0, 3]], dtype=int64))


[[0.52786405 0.47213595]
 [0.30901699 0.69098301]]
0.5278640447218036 0.47213595527819635
```

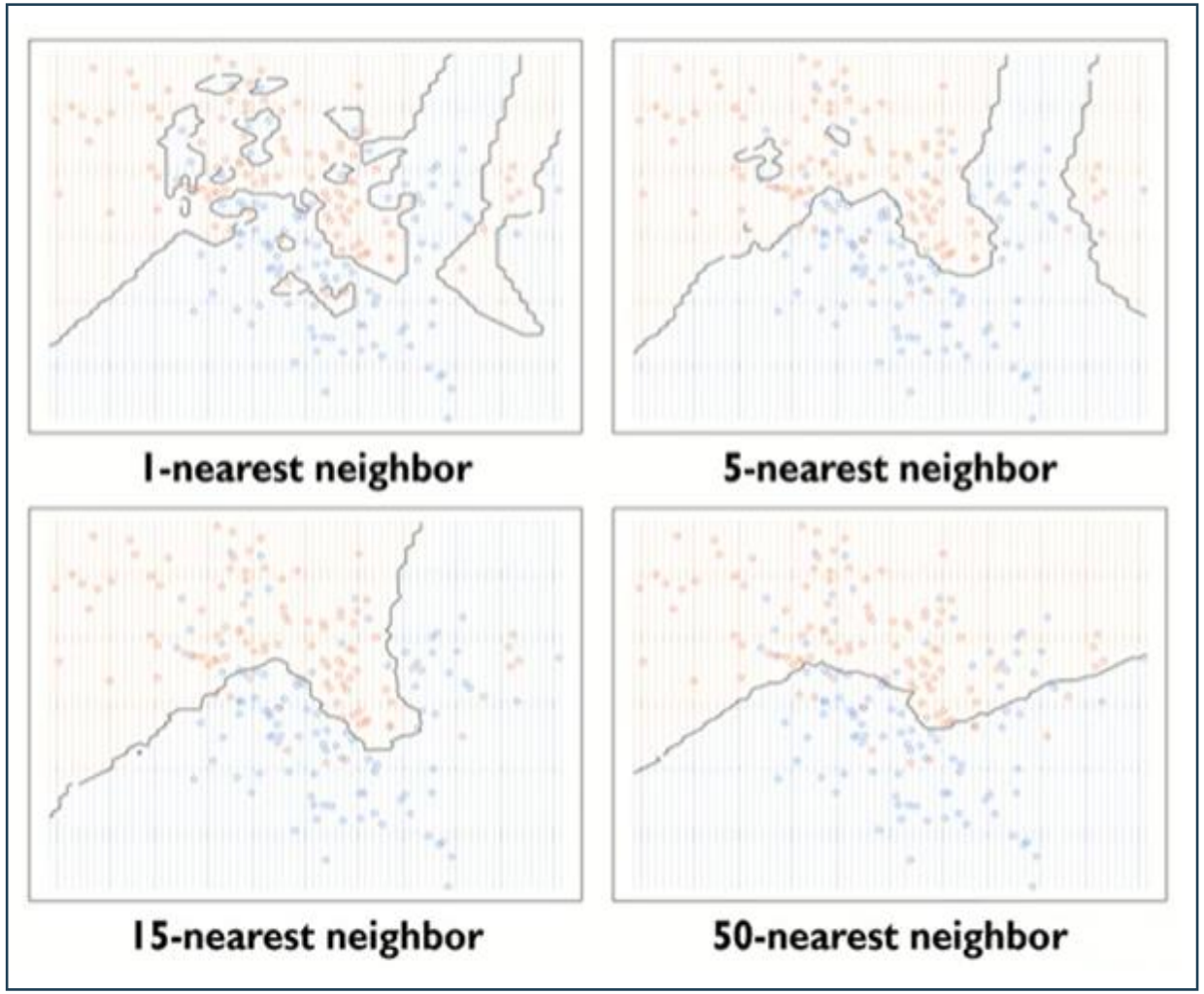


거리의 역수를 이용해 분류

가중치를 부여해 동률이더라도 더 가까운 이웃으로 분류

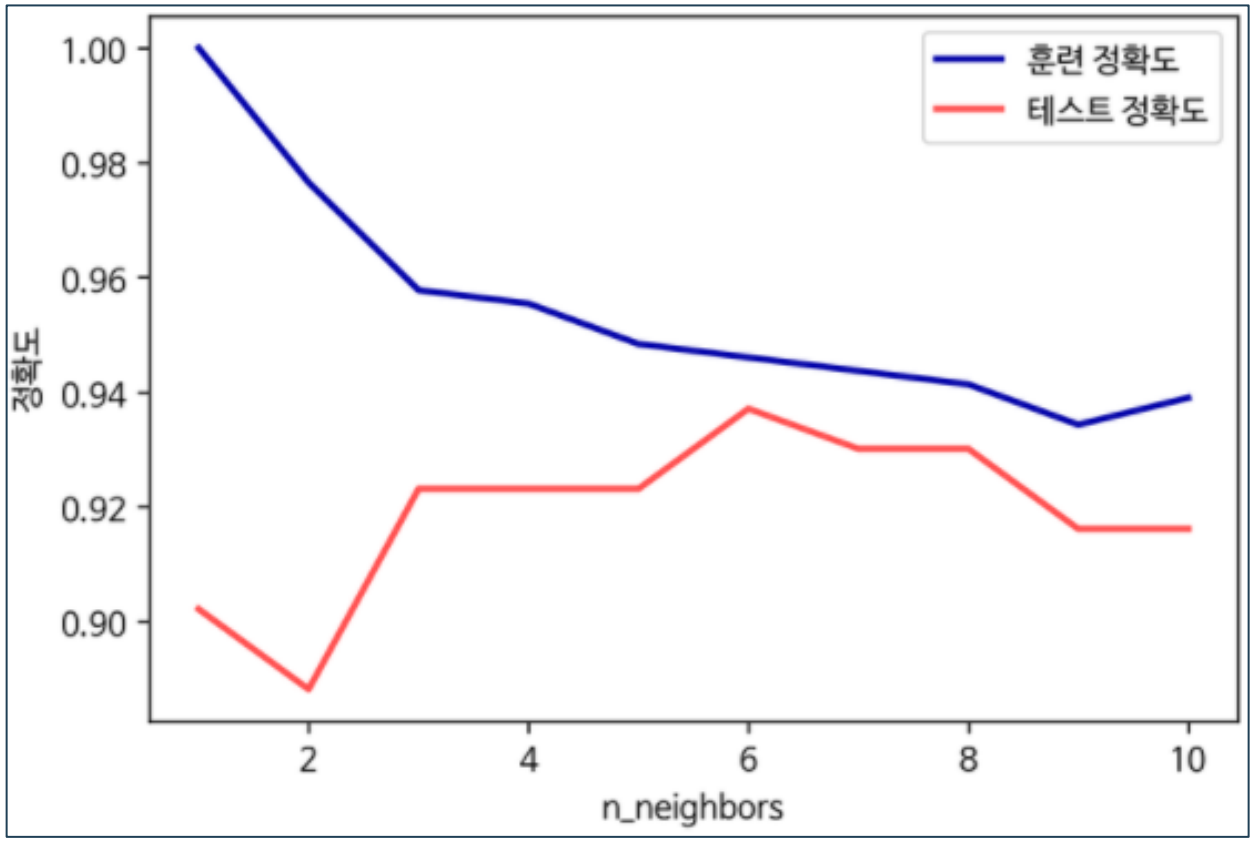
KNN 한계 및 고려할 점

- 최적의 k가 명확히 존재하지 않아 **Trial and Error** (다만 k가 작으면 overfitting 가능성 존재)이 존재함
- 어떤 거리 척도가 좋은 지 확정적이진 않아. 데이터 특성에 맞는 **거리 척도를 임의로 선정해야 함**
- Class가 2개인 문제에선 k를 홀수로 설정하는 게 좋아 **동점 발생**시 분류 할 수 없는 경우가 발생 (scikit learn에선 임의로 분류 시행, 오류가 발생하진 않음)
- 새로운 관측치와 각각 학습 데이터 간 거리를 전부 측정해야 해서 **계산시간이 오래 걸림**
- 그룹 간 명확한 차이가 없으면 **그룹 간 경계를 구분하는데 효과적이지 못함**
- **고차원**에서는 거리 개념이 모호해져서 KNN이 잘 작동하지 않는다고 알려짐. -> 해결방안: 차원 축소
- 결과를 해석해 피처와 클래스 간 관계를 이해하는 것이 아니라 분석가가 선험적으로 각 **변수와 클래스 간 관계를 파악**하여 알고리즘을 적용해야 효과적



KNN 한계 및 고려할 점

- **최적 k**를 구하는데 도움이 될 수 있는 지표
- K가 적을 수록 **overfitting**이 되기 때문에 k가 1일때 훈련 **정확도는 100%**에 가까움
- K가 커질 수록 **underfitting**(모델이 단순)되며 **training error**는 커짐
- 이렇게 분포가 예쁘게 나올 땐 훈련 정확도와 테스트 정확도가 동시에 커지는 지점이 **best K**라고 볼 수 있음 (오른쪽의 경우 6)
- 하지만 분석 데이터의 분포에 따라 오른쪽과 같은 선이 나오지 않고 **변동성이 큰 분포**도 나올 수 있음



https://tensorflow.blog/%ED%8C%8C%EC%9D%B4%EC%8D%AC-%EB%A8%B8%EC%8B%A0%EB%9F%AC%EB%8B%9D/2-3-2-k-%EC%B5%9C%EA%B7%BC%EC%A0%91-%EC%9D%B4%EC%9B%83/

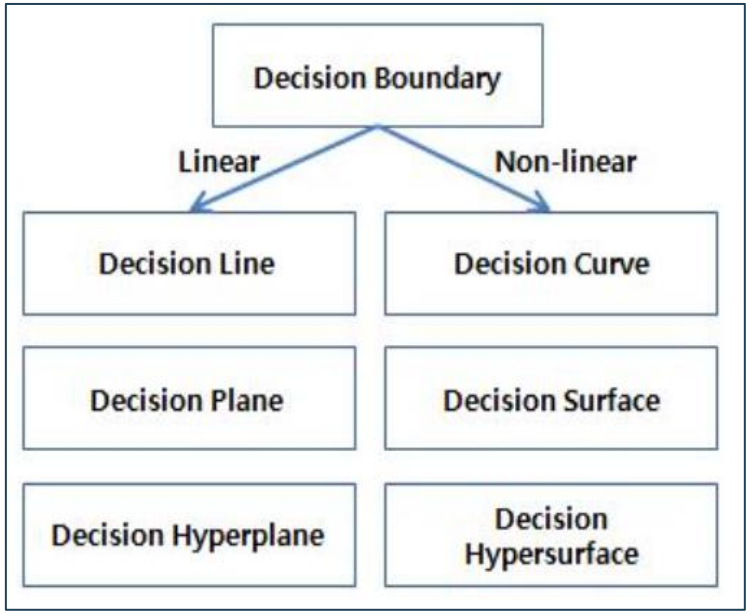
3. 서포트 벡터 머신(Support Vector Machin/SVM)

SVM이란?

SVM(Support Vector Machine)

- 지지벡터를 활용해 Margin을 극대화하는 결정경계를 찾는 것
- 벡터 공간에 위치한 훈련 데이터의 좌표와 각 데이터가 어떤 분류 값을 가져야 하는지 정답을 입력 받아서 학습한다.

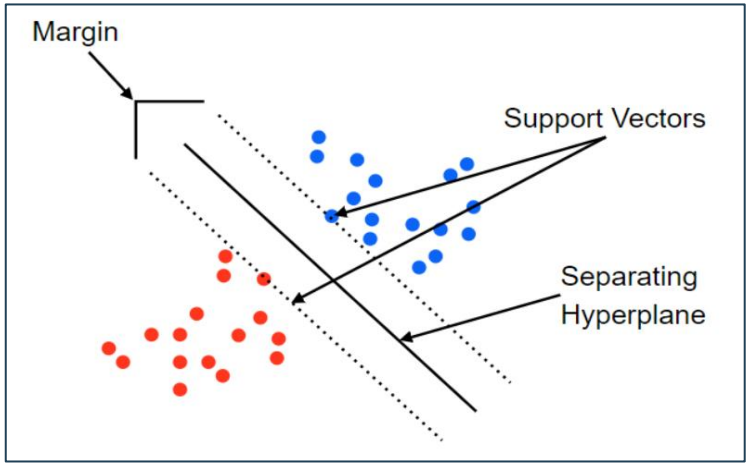
→ 데이터셋의 여러 속성을 나타내는 데이터프레임의 각 열은 열 벡터 형태로 구현된다.



결정 경계

- 결정 경계는 기본 벡터공간을 각 클래스에 대하여 하나씩 두 개의 집합으로 나누는 초평면 이다

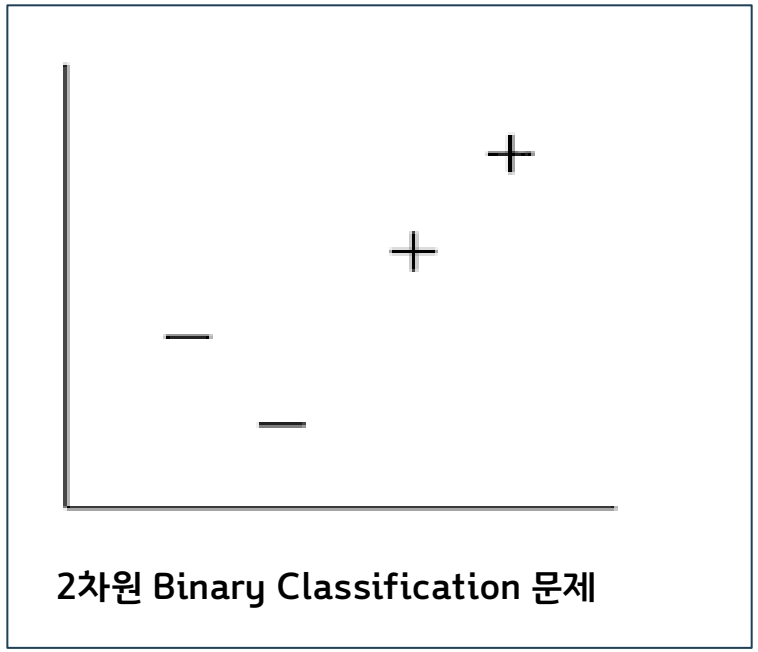
→ 어느 한 쪽에도 치우치지 않고, 양쪽 데이터와 균등한 위치에 분류 기준을 세워줘야 한다.



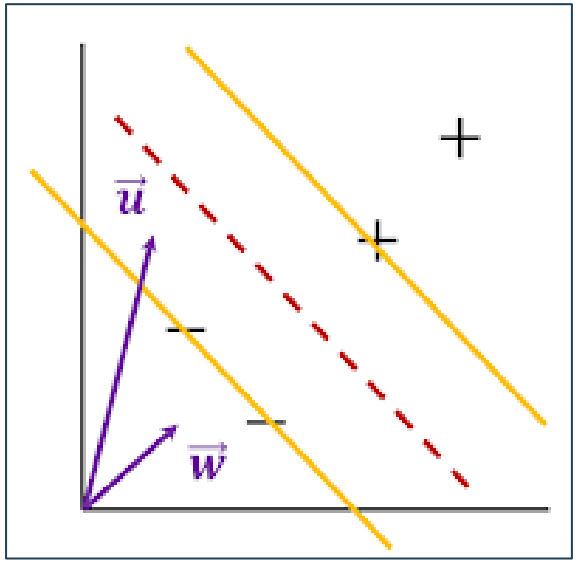
SVM 결정 경계를 찾는 방식

제약하(지지벡터) 최적화(마진 극대화) 문제

학습데이터 기반 마진 극대화를 통해 testing(generalization) error를 최소화



2차원 Binary Classification 문제



결정경계: $w^T x + b = 0$

결정규칙 (Decision Rule)

$w^T x + b \leq 0$ 이면 -클래스 $w^T x + b \geq 0$ 이면 +클래스

하지만 w와 b의 값을 모름 (w는 결정경계에 수직인 **법선 벡터**)

http://jaejunyoo.blogspot.com/2018/01/support-vector-machine-1.html

SVM 결정 경계를 찾는 방식

1. 인위적인 제약을 부여

x+를 +샘플 x−가 −의 샘플일때
+는 1보다 크게 -는 -1보다 작아야 한다고 가정

$$\vec{w} \cdot \vec{x}_+ + b \geq 1$$
$$\vec{w} \cdot \vec{x}_- + b \leq -1$$

2. 계산의 편리함을 위해 변수 추가

$$y_i = \begin{cases} 1 & for \quad `+' \\ -1 & for \quad `-' \end{cases}$$

3. 결국 이렇게 표현 가능

$$y_i(\vec{w} \cdot \vec{x}_i + b) \geq 1$$
$$y_i(\vec{w} \cdot \vec{x}_i + b) - 1 \geq 0$$

4. 정확히 노란 경계선에 걸치면 아래 식이 0이 되도록 제약

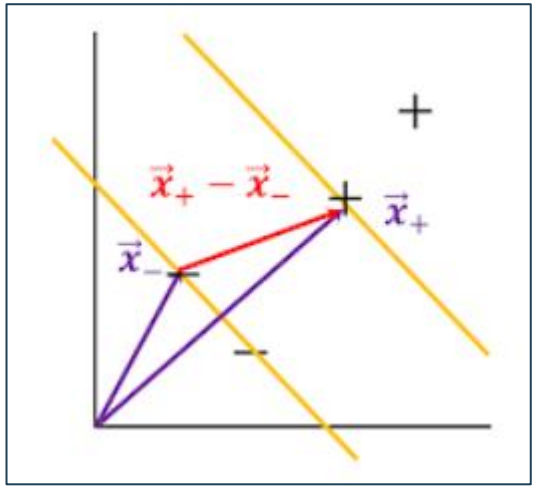
$$y_i(\vec{w} \cdot \vec{x}_i + b) - 1 = 0 \qquad for \quad \vec{x}_i \in \text{노란선 } (gutters)$$

5. 지지벡터 사이의 거리 벡터를 이용해 마진의 크기 도출

$$WIDTH = (x_+ - x_-) \cdot \frac{\vec{w}}{||\vec{w}||} \qquad WIDTH = \frac{2}{||\vec{w}||}$$

6. 또다시 수학적 편리함을 위해 최소화 문제로 변형

$$\max \frac{1}{||\vec{w}||} \leftrightarrow \min ||\vec{w}|| \leftrightarrow \min \frac{1}{2}||\vec{w}||^2$$



SVM 결정 경계를 찾는 방식

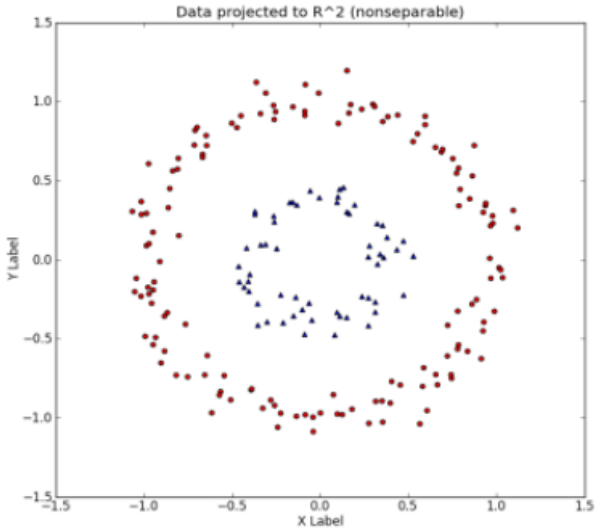
7. 앞의 4번에 등장한 제약을 고려한 최적화 방식인 라그랑지 승수법을 적용

이차 목적식이 만들어짐, global solution 존재

카루쉬 쿤 터커(KKT) 조건, 라그랑지 프라임, 라그랑지 듀얼 등을 이용해 최적의 w와 b를 구해 마진을 극대화하는 결정경계를 도출

$$\mathcal{L}(w,b,\alpha) = \frac{1}{2}||\vec{w}||^2 - \sum_{i=1}^{N} \alpha_i[y_i(\vec{w}\cdot\vec{x}_i + b) - 1]$$

$$minimize \quad w.r.t. \quad \vec{w}\,and\,b \qquad maximize \quad w.r.t. \quad \alpha_i \geq 0 \quad \forall i$$



8. 선형 분리가 불가한 데이터들의 경우 커널 함수를 이용해 저차원 공간에서 선형 분리가 안되는 테이터들을 고차원 공간에 매핑시켜 비선형성을 처리

선형 서포트 벡터 머신

$$k(x_1, x_2) = x_1^T x_2$$

다항 커널 (Polynomial Kernel)

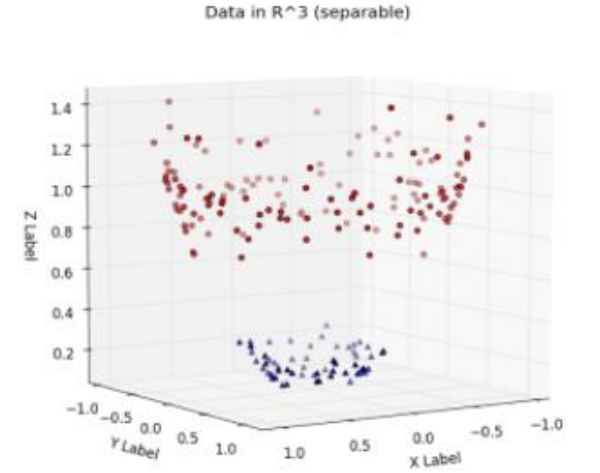
$$k(x_1, x_2) = (\gamma(x_1^T x_2) + \theta)^d$$

RBF(Radial Basis Function) 또는 가우시안 커널(Gaussian Kernel)

$$k(x_1, x_2) = \exp\left(-\gamma||x_1 - x_2||^2\right)$$

시그모이드 커널 (Sigmoid Kernel)

$$k(x_1, x_2) = \tanh(\gamma(x_1^T x_2) + \theta)$$



http://jaejunyoo.blogspot.com/2018/01/support-vector-machine-1.html

SVM 결정 경계를 찾는 방식

For more studies...

(글)
구글에 다음과 같이 검색
a tutorial on support vector machines for pattern recognition chirstopher J.C. burges

수식 설명, 파이썬 코드
https://datascienceschool.net/03%20machine%20learning/13.02%20%EC%84%9C%ED%8F%AC%ED%8A%B8%20%EB%B2%A1%ED%84%B0%20%EB%A8%B8%EC%8B%A0.html
https://datascienceschool.net/03%20machine%20learning/13.03%20%EC%BB%A4%EB%84%90%20%EC%84%9C%ED%8F%AC%ED%8A%B8%20%EB%B2%A1%ED%84%B0%20%EB%A8%B8%EC%8B%A0.html

(영상)
김성범 고려대 교수 SVM 강의
https://www.youtube.com/watch?v=qFg8cDnqYCI
https://www.youtube.com/watch?v=ltjhyLkHMls

SVM 특징 및 장단점

장점

새로운 데이터에 대한 분류를 정확히 할 수 있음

범주, 수치 예측 문제에 사용이 가능함

오류 데이터 영향이 적음

신경망보다 사용하기 쉬움

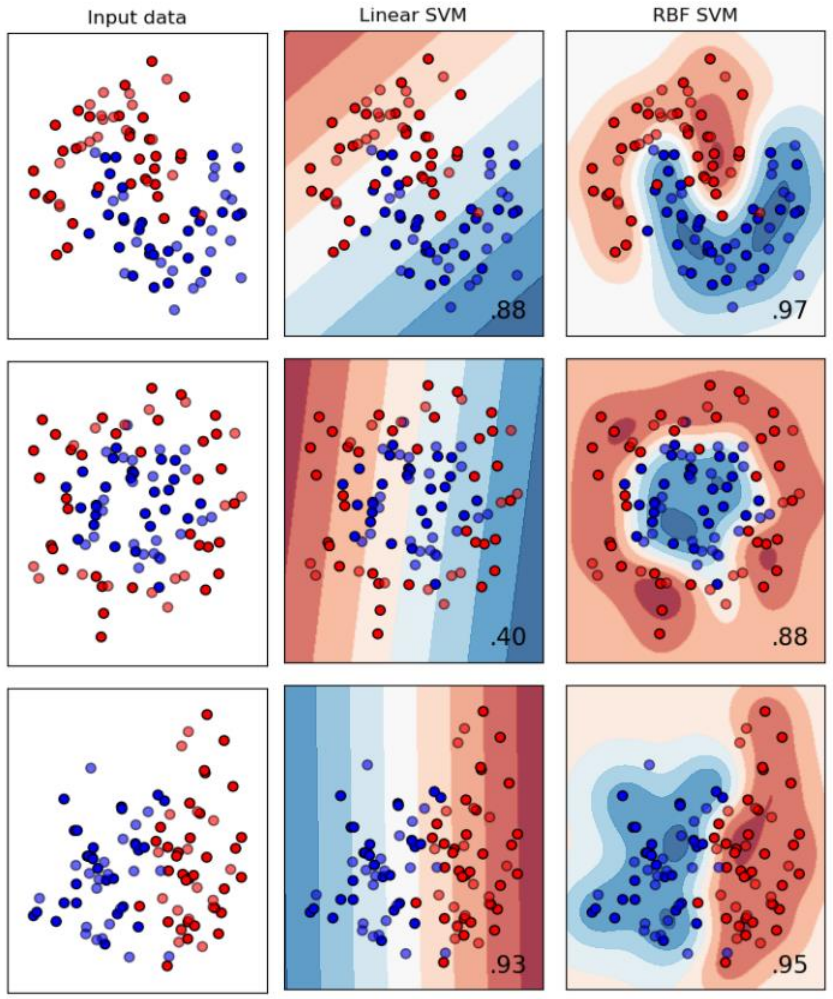
단점

여러 개의 조합 테스트 필요

학습 속도가 느림

해석이 어렵고 복잡한 블랙박스 형태로 되어있음

SVC(kernel="linear", C=0.025), SVC(gamma=2, C=1),



https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html#sphx-glr-auto-examples-classification-plot-classifier-comparison-py

SVM 활용사례

고지혈증 유병 예측모형 개발

텍스트 분류

스펨 메일 필터링

소비자 소비 상품 분류

SVM 하이퍼파라미터

SVC(*, C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', break_ties=False, random_state=None)

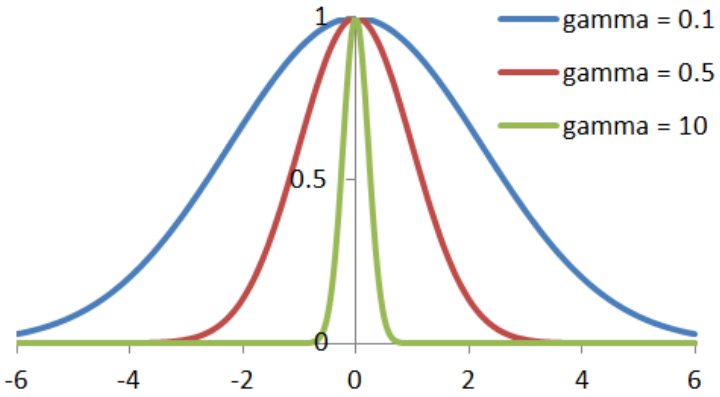
C	마진과 training error에 대한 trade-off를 결정하는 tuning parameter
Kernel	‘linear’, ‘poly’, ‘rbf’, ‘sigmoid’
gamma	Kernel coefficient for ‘rbf’, ‘poly’ and ‘sigmoid’ float를 넣어 조정 ‘scale’: uses 1 / (n_features * X.var()) ‘auto’: uses 1 / n_features

https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

SVM 파라미터 C,Gamma

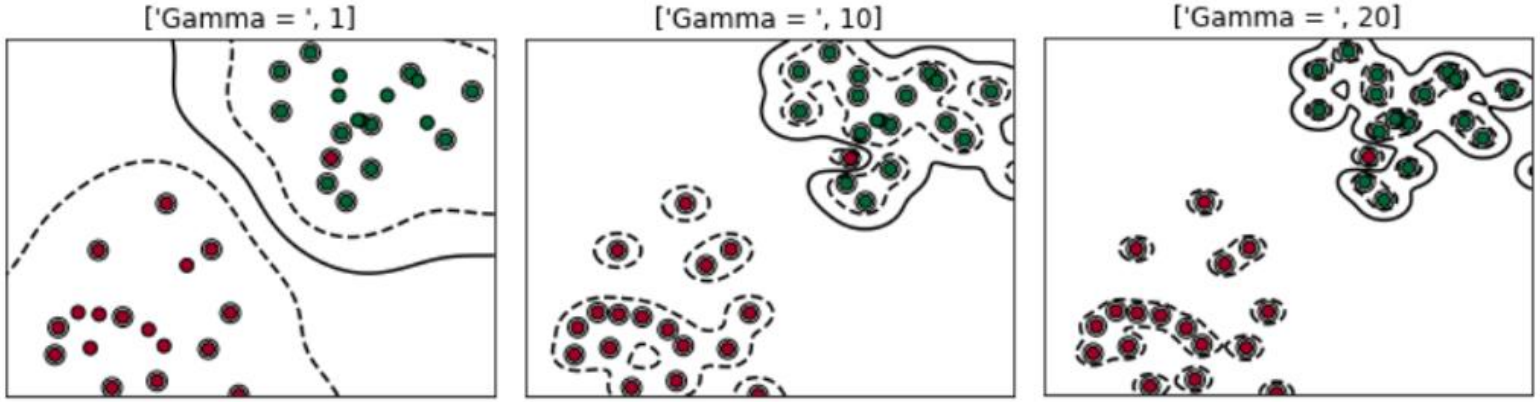
C

C 는 Training Data를 정확히 구분할지, 결정 경계를 일반화할지를 결정해준다.
C가 클수록 training error 허용하지 않아(overfit)
작을수록 training error 허용 (underfit)



Gamma

하나의 데이터 샘플이 영향력을 행하사는 거리를 결정
Gamma가 크면 결정경계의 곡률이 커지면서 학습데이터 의존도가 높아지고 (overfit) 작으면 학습 데이터에 대한 의존도가 낮아진다. (underfit)



SVM Soft Margin VS Hard Margin

Hard Margin

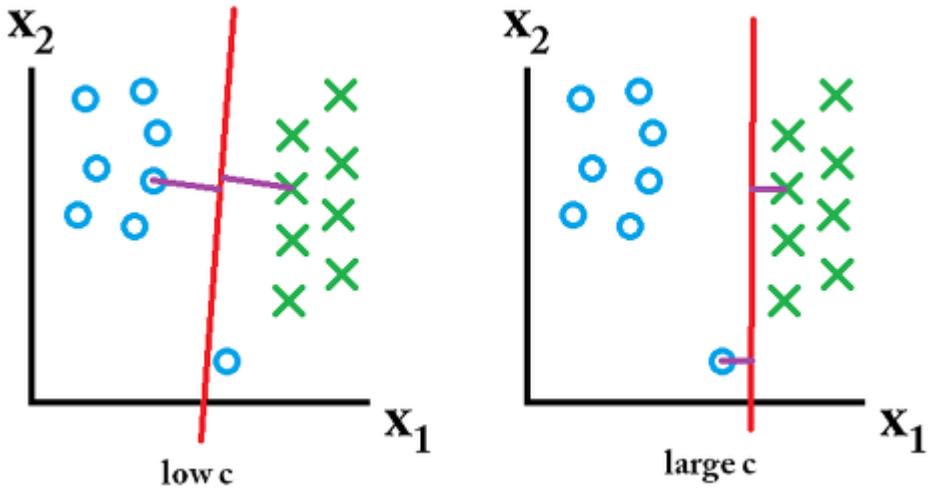
두 개의 클래스를 엄격하게 분리하는 방법
→ 결정 경계를 사이에 두고 무조건 한 클래스에 속해야 함

1. 몇 개의 노이즈로 인해 두 그룹을 구별하는 분리초평면(결정 경계)를 잘 못 구할 수 있다.
2. 노이즈로 인해 결정경계를 찾지 못할 수 있다.

Soft Margin

결정 경계를 사이로 여유 변수가 존재하는 것을 용인해주는 방법
→ C가 이것을 결정함

1. Outlier를 배제하고 결정 경계를 찾을 수 있다.
2. 비용과 시간을 절약할 수 있다.



SVM 파라미터 커널(kernel)

커널(kernel)

: 선형에서 구분하지 못하는 구조를 Kenel을 사용해 데이터를 변환해 구분시킨다.

1. 선형 (linear)

Kernel을 적용하지 않은 기본SVM으로 Soft Margin을 계산하는데만 사용된다.

2. 다항식(poly)

Parameter : Cost, Gamma(지수의 분모에 해당하는 파라미터), Coef, Degree

3. 가우시안 (RBF)

Parameter : Cost, Gamma

4. 시그모이드(sigmoid)

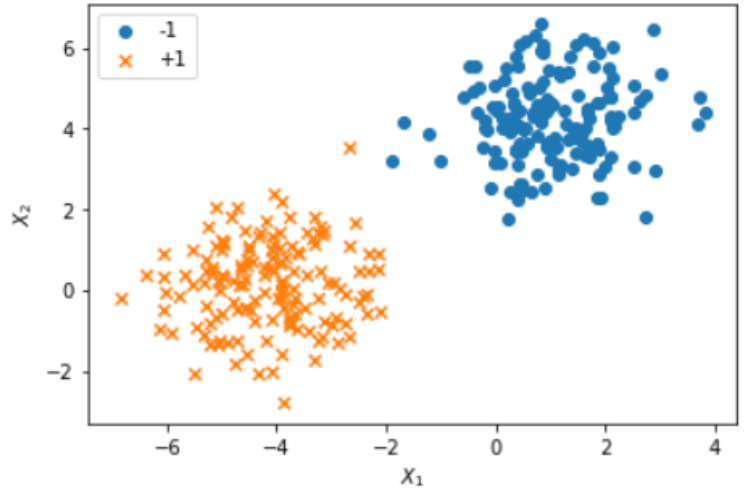
Parameter : Cost , Gmma, Coef

- **선형(linear)** : $K(\mathbf{a}, \mathbf{b}) = \mathbf{a}^T \cdot \mathbf{b}$
- **다항식(poly)** : $K(\mathbf{a}, \mathbf{b}) = \left(\gamma \mathbf{a}^T \cdot \mathbf{b} + r\right)^d$
- **가우시안 RBF(rbf)** : $K(\mathbf{a}, \mathbf{b}) = \exp\left(-\gamma \|\mathbf{a} - \mathbf{b}\|^2\right)$
- **시그모이드(sigmoid)** : $K(\mathbf{a}, \mathbf{b}) = \tanh\left(\gamma \mathbf{a}^T \cdot \mathbf{b} + r\right)$

SVM 예시

데이터 가공

```python
from sklearn.datasets import make_blobs
from matplotlib import pyplot as plt

X, y = make_blobs(n_samples=300, n_features=2, centers=2, random_state=3, cluster_std=1)

y=2*y-1 #+1, -1로 바꿔주는 함수

plt.scatter(X[y == -1, 0], X[y == -1, 1], marker='o', label="-1")
plt.scatter(X[y == +1, 0], X[y == +1, 1], marker='x', label="+1")
plt.xlabel("$X_1$")
plt.ylabel("$X_2$")
plt.legend()
plt.show()
```

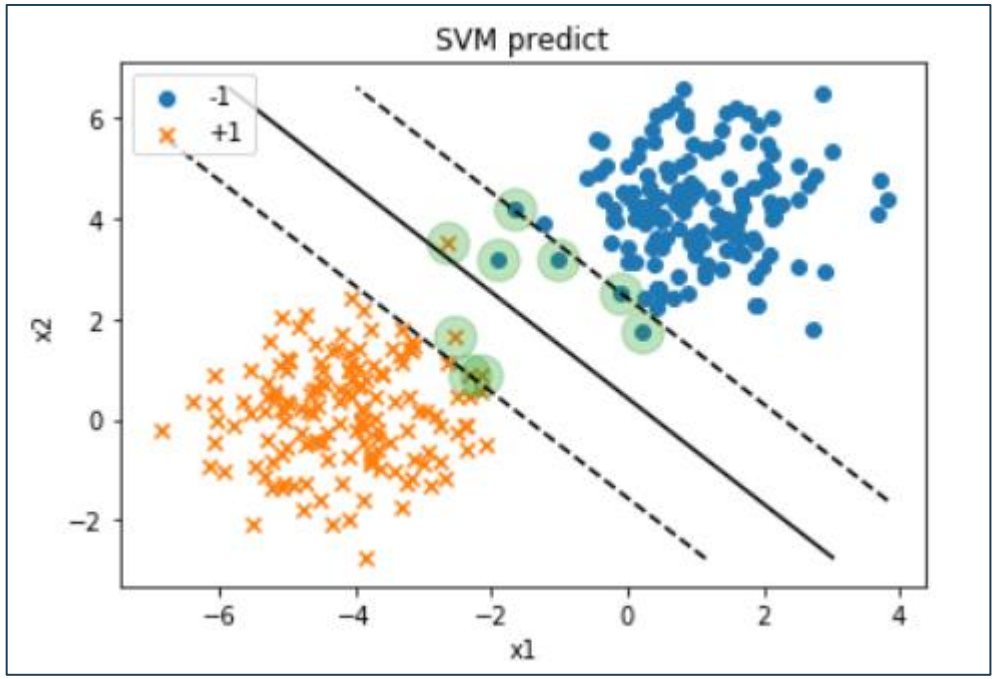


SVM 예시

```
from sklearn.svm import SVC
import numpy as np

model = SVC(kernel='linear', C=0.1).fit(X, y)

xmin = X[:, 0].min()
xmax = X[:, 0].max()
ymin = X[:, 1].min()
ymax = X[:, 1].max()
xx = np.linspace(xmin, xmax, 10)
yy = np.linspace(ymin, ymax, 10)
X1, X2 = np.meshgrid(xx, yy)

Z = np.empty(X1.shape)
for (i, j), val in np.ndenumerate(X1):
    x1 = val
    x2 = X2[i, j]
    p = model.decision_function([[x1, x2]])
    Z[i, j] = p[0]
levels = [-1, 0, 1]
linestyles = ['dashed', 'solid', 'dashed']
plt.scatter(X[y == -1, 0], X[y == -1, 1], marker='o', label="-1")
plt.scatter(X[y == +1, 0], X[y == +1, 1], marker='x', label="+1")
plt.contour(X1, X2, Z, levels, colors='k', linestyles=linestyles)
plt.scatter(model.support_vectors_[:, 0], model.support_vectors_[:, 1], s=300, alpha=0.3)

plt.xlabel("x1")
plt.ylabel("x2")
plt.legend()
plt.title("SVM predict")

plt.show()
```

Kernel : linear

C : 0.1



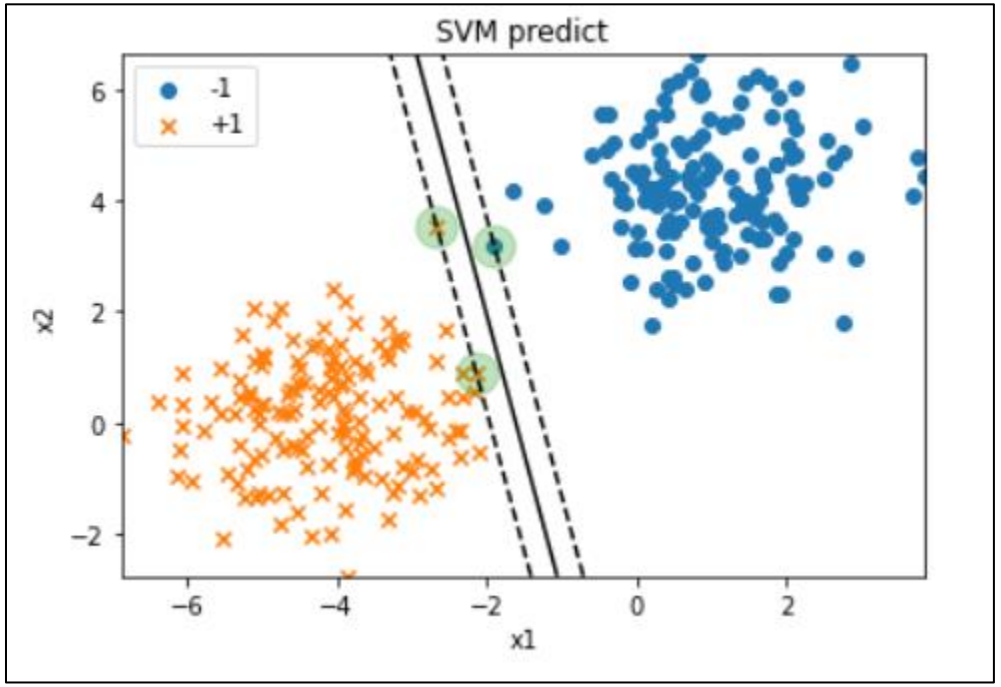
SVM 예시

```
from sklearn.svm import SVC
import numpy as np

model = SVC(kernel='linear', C=10).fit(X, y)

xmin = X[:, 0].min()
xmax = X[:, 0].max()
ymin = X[:, 1].min()
ymax = X[:, 1].max()
xx = np.linspace(xmin, xmax, 10)
yy = np.linspace(ymin, ymax, 10)
X1, X2 = np.meshgrid(xx, yy)

Z = np.empty(X1.shape)
for (i, j), val in np.ndenumerate(X1):
    x1 = val
    x2 = X2[i, j]
    p = model.decision_function([[x1, x2]])
    Z[i, j] = p[0]
levels = [-1, 0, 1]
linestyles = ['dashed', 'solid', 'dashed']
plt.scatter(X[y == -1, 0], X[y == -1, 1], marker='o', label="-1")
plt.scatter(X[y == +1, 0], X[y == +1, 1], marker='x', label="+1")
plt.contour(X1, X2, Z, levels, colors='k', linestyles=linestyles)
plt.scatter(model.support_vectors_[:, 0], model.support_vectors_[:, 1], s=300, alpha=0.3)

plt.xlabel("x1")
plt.ylabel("x2")
plt.legend()
plt.title("SVM predict")

plt.show()
```

C 부분만 바꿔보세요

Kernel : linear

C : 10



4. 앙상블(Ensemble)

앙상블이란?

여러 개의 분류기를 생성하고 그 예측을 결합하는 학습방법임

=>보다 정확한 최종 예측 도출이 가능함

다양한 분류기의 예측결과를 결합함으로써 단일 분류기보다 신뢰성이 높은 예측값을 얻는 걸 목적으로 함

=>**집단 지성**과 같은 효과를 얻을 수 있음 (wisdom of crowd)

=>하나의 강한 머신러닝 알고리즘보다 **여러 개의 약한 머신러닝** 알고리즘이 낫다

*앙상블의 어원: 앙상블은 프랑스어로 '함께', '동시에'라는 의미에서 진화하여 '조화'의 의미를 갖는 음악 용어

앙상블이란?

특징

이미지, 영상, 음성 등의 비정형 데이터의 분류는 딥러닝이 뛰어난 성능을 보이지만,

대부분 정형 데이터의 분류에서는 앙상블이 뛰어난 성능을 보이고 있음

*정형 데이터: 데이터를 정제해 구조화한 데이터

*비정형 데이터: 이미지, 영상, 음성 등과 같이 구조화가 되어 있지 않은 데이터

장점

단일 모델에 비해서 분류 성능이 우수함

단점

모델 결과의 해석이 상대적으로 어렵고, 예측 시간이 많이 소요될 수 있음

앙상블이란?

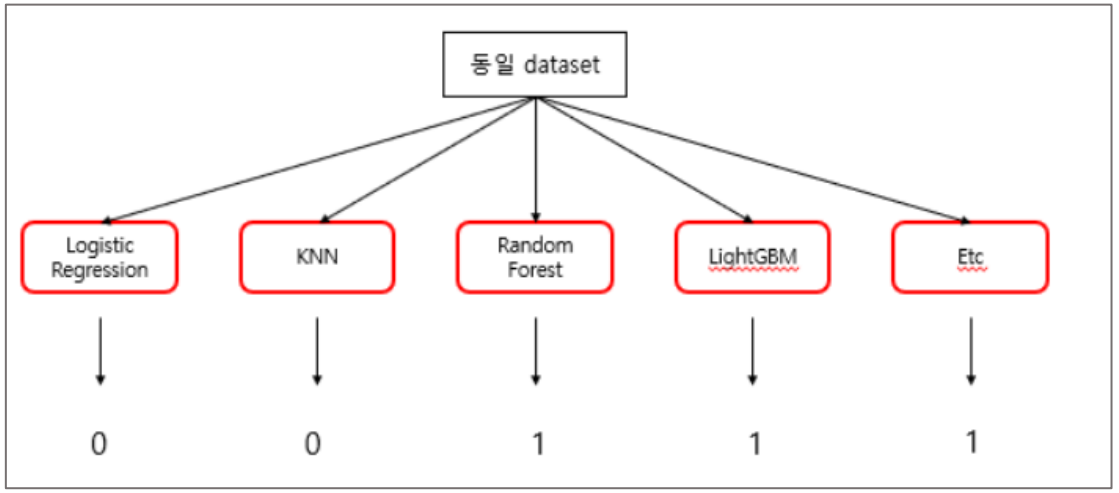
보팅	여러 개의 분류기가 투표를 통해 최종 결과를 예측함	서로 다른 알고리즘을 가진 분류기를 결합함
배깅		같은 유형의 알고리즘을 기반으로 하지만 데이터 샘플링을 서로 다르게 가져감
부스팅	여러 개의 분류기가 순차적으로 학습을 수행하되, 앞에서 학습한 분류기가 예측이 틀린 데이터에 대해서는 올바르게 예측하도록 다음 분류기에는 가중치를 부여하며 학습과 예측을 진행함	
스태깅	여러가지 다른 모델의 예측 결과값을 다시 학습 데이터로 만들어서 다른 모델로 재학습시켜 결과를 예측함	

보팅이란?

여러 종류의 알고리즘을 사용한 각각의 결과에 대해 투표를 통해 최종 결과를 예측하는 방식

일반적으로 서로 다른 알고리즘을 가진 분류기를 결합함

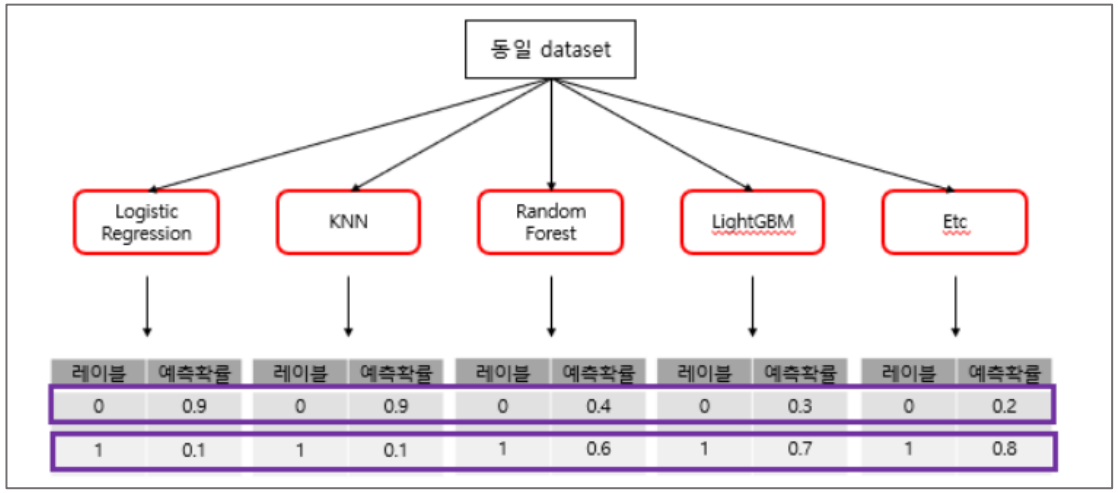
<하드 보팅(Hard Voting)>



다수결의 원칙과 비슷함

다수의 분류기가 결정한 예측값을 최종 보팅 결괏값으로 선정함

<소프트 보팅(Soft Voting)>



레이블 값 결정 확률을 모두 더하고 이들의 평균값을 구해

확률이 가장 높은 레이블 값을 최종 보팅 결괏값으로 선정함

이미지 출처: 바람돌이의 빅데이터 http://blog.naver.com/winddori2002/221829427442

보팅 예제

사이킷런에 있는 유방암 예제 활용!

```
import pandas as pd

from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

cancer = load_breast_cancer()

dta_df = pd.DataFrame(cancer.data, columns=cancer.feature_names)
dta_df.head(3)
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	worst texture	worst perimeter	worst area	worst smoothness	worst compactness	worst concavity	worst concave points	worst symmetry	worst fractal dimension
0	17.99	10.38	122.8	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	25.38	17.33	184.6	2019.0	0.1622	0.6656	0.7119	0.2654	0.4601	0.11890
1	20.57	17.77	132.9	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	24.99	23.41	158.8	1956.0	0.1238	0.1866	0.2416	0.1860	0.2750	0.08902
2	19.69	21.25	130.0	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	23.57	25.53	152.5	1709.0	0.1444	0.4245	0.4504	0.2430	0.3613	0.08758

3 rows × 30 columns

보팅 예제

1. 로지스틱 회귀와 **KNN**을 활용해 **앙상블(보팅)** 모델 구현
2. **VotingClassifier**의 정확도 평가

```
#개별 모델은 로지스틱 회귀와 KNN임
lr_clf = LogisticRegression()
knn_clf = KNeighborsClassifier(n_neighbors=8)

#개별 모델을 소프트 보팅 기반의 앙상블 모델로 구현한 분류기
vo_clf = VotingClassifier(estimators=[('LR', lr_clf), ('KNN', knn_clf)],voting='soft')

X_train, X_test, y_train, y_test = train_test_split(cancer.data, cancer.target,
                                                    test_size=0.2, random_state=156)

#VotingClassfier 학습/예측/평가
vo_clf.fit(X_train, y_train)
pred = vo_clf.predict(X_test)
print('Voting 분류기 정확도: {0:.4f}'.format(accuracy_score(y_test,pred)))
```

```
Voting 분류기 정확도: 0.9474
```

보팅 예제

3. 개별 모델(로지스틱 회귀/KNN)의 정확도 측정

```
#개별 모델의 학습/예측/평가
classifiers = [lr_clf, knn_clf]
for classifier in classifiers:
    classifier.fit(X_train, y_train)
    pred = classifier.predict(X_test)
    class_name = classifier.__class__.__name__
    print('{0} 정확도:{1:.4f}'.format(class_name, accuracy_score(y_test,pred)))
```

```
LogisticRegression 정확도:0.9386
KNeighborsClassifier 정확도:0.9386
```

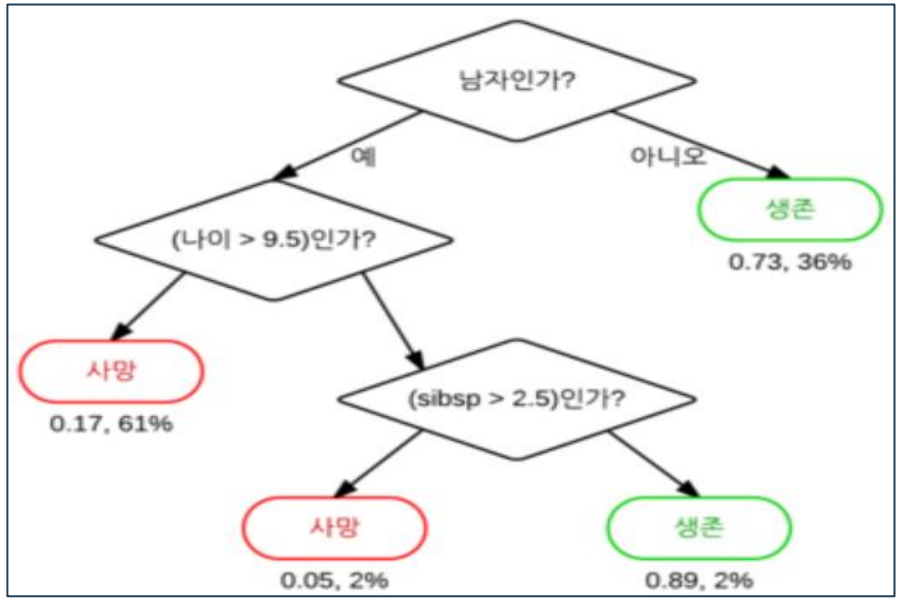
VotingClassifier의 정확도(=0.9474) > **로지스틱 회귀/KNN의 정확도(=0.9386)**

따라서, 개별 모델을 통한 예측보다 앙상블 모델(보팅)을 통한 예측의 성능이 더 좋음을 확인할 수 있음

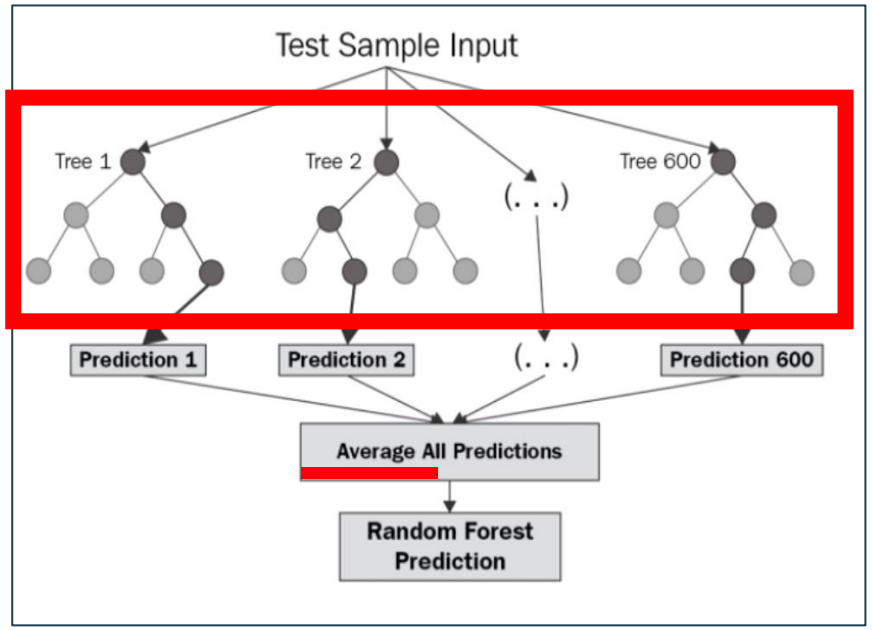
랜덤포레스트 Random Forest

랜덤포레스트 : **여러개의 Decision Tree**를 만들고 연결하여 결과를 취합한 후 평균을 내어 성능을 높인 모델

Decision Tree
: 의사결정을 하는데 도움을 주는
나무 모양의 예측 모델



여러 트리들의 결과를 평균내면
과적합 문제가 상당부분 해결되어 성능이 향상됨

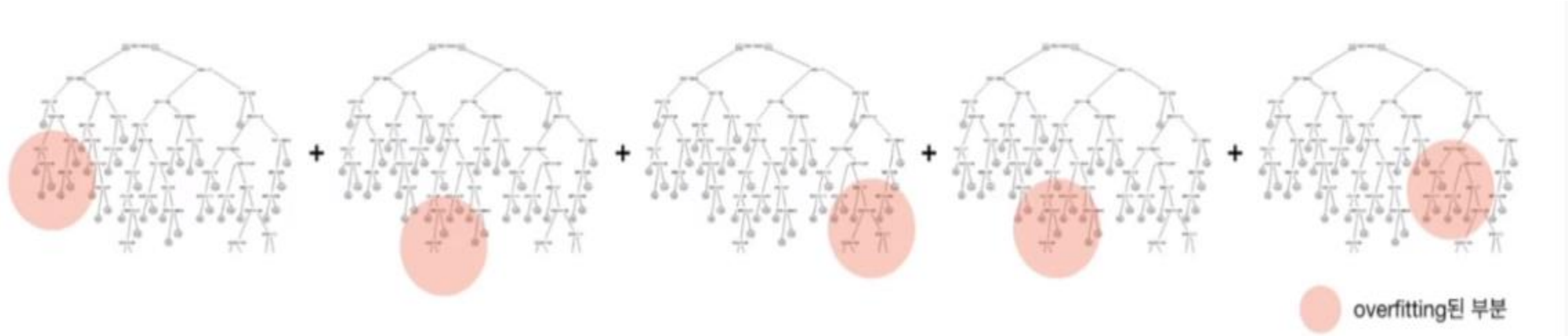


※ Decision Tree는 알고리즘을 이해하기 쉽고, 비교적 성능이 높다는 장점이 있지만, 과적합되기 쉽다는 단점이 있음
※ 과적합 : Train data에서는 높은 정확도를 보이지만 Test data에서는 높은 성능을 보이지 못함

https://ko.wikipedia.org/wiki/%EA%B2%B0%EC%A0%95_%ED%8A%B8%EB%A6%AC_%ED%95%99%EC%8A%B5%EB%B2%95
https://levelup.gitconnected.com/random-forest-regression-209c0f354c84

랜덤포레스트 Random Forest

랜덤포레스트 : **여러개의 Decision Tree**를 만들고 연결하여 결과를 취합한 후 평균을 내어 성능을 높인 모델

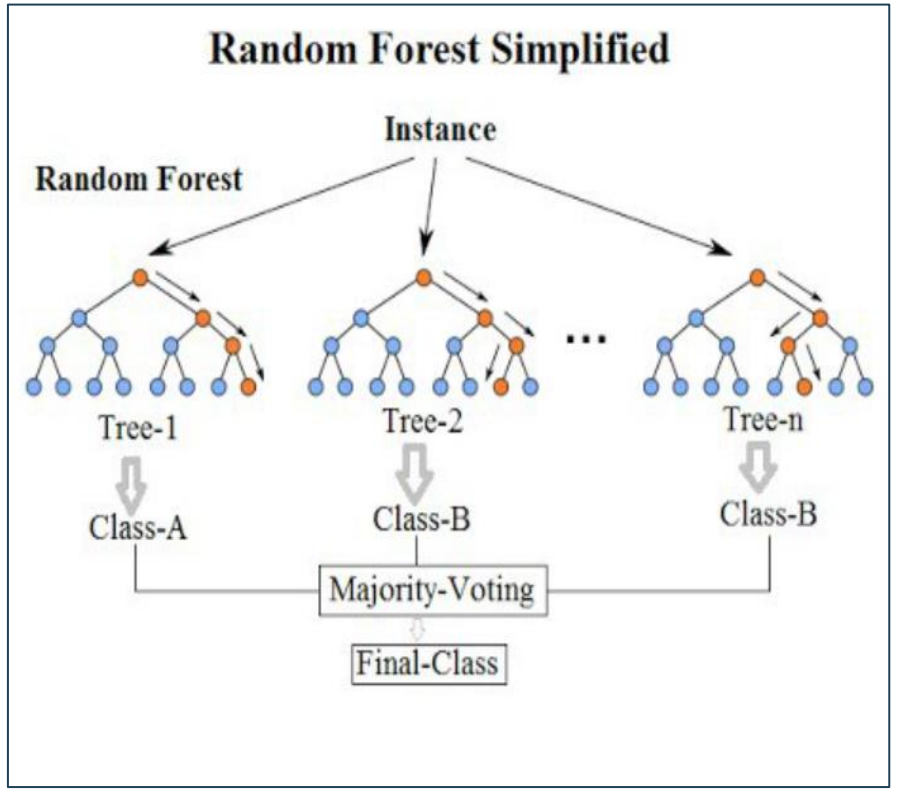


- Decision Tree는 과적합이 되기 쉬운 알고리즘
- 각각의 Tree에서 과적합이 되는 부분이 모두 다른데 이 Tree들을 합쳐서

서로의 과적합 부분을 보완할 수 있다면, 최종적으로 나온 1개의 모델인 랜덤포레스트의 성능은 높아지게 됨

https://ko.wikipedia.org/wiki/%EA%B2%B0%EC%A0%95_%ED%8A%B8%EB%A6%AC_%ED%95%99%EC%8A%B5%EB%B2%95
https://levelup.gitconnected.com/random-forest-regression-209c0f354c84

랜덤포레스트 Random Forest



- 배깅(bagging)의 대표적인 예로 같은 알고리즘으로 여러 개의 분류기를 만들어서 보팅으로 최종 결정하는 알고리즘

- 앙상블 알고리즘 중 비교적 빠른 수행 속도를 가지고 있으며, 다양한 영역에서 높은 예측 성능을 보임

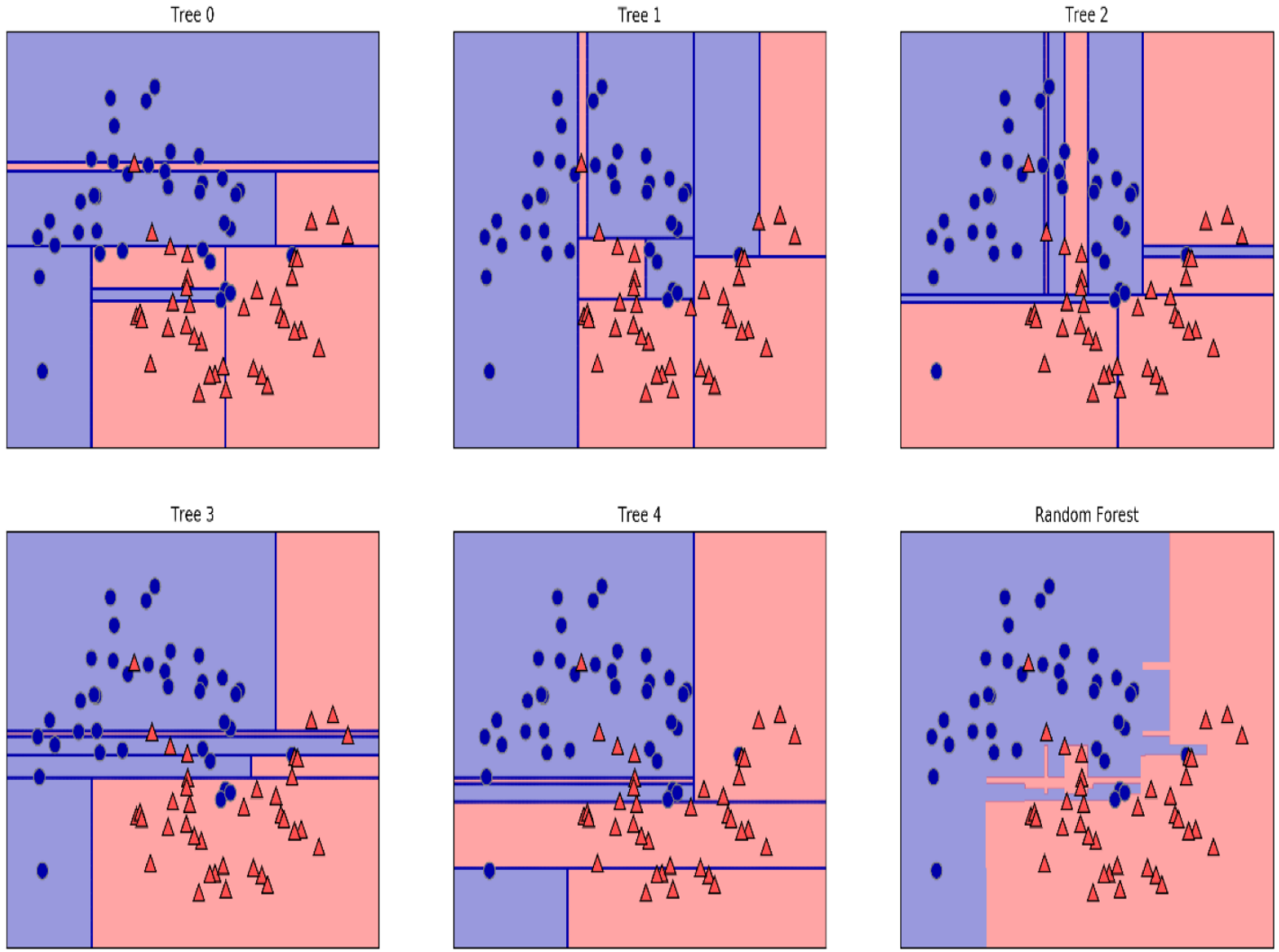
- 결정 트리(Decision Tree)를 기반으로 하고, 결정 트리의 쉽고 직관적인 장점을 그대로 지님

- 랜덤 포레스트는 여러 개의 결정 트리 분류기가 전체 데이터에서 배깅 방식으로 각자의 데이터를 샘플링해 최종적으로 모든 분류기가 보팅을 통해 예측을 결정하게 됨

- 랜덤 포레스트는 데이터의 수가 많아지면 결정 트리에 비해 속도가 떨어지며 결과 해석이 어렵다는 단점이 있음

https://nonmeyet.tistory.com/entry/%EB%B2%A0%EA%B9%85Bagging-%EA%B3%BC-%EB%9E%9C%EB%8D%A4%ED%8F%AC%EB%A0%88%EC%8A%A4%ED%8A%B8%EC%9D%98-%EC%A0%95%EC%9D%98-%EB%B0%8F-%EC%84%A4%EB%AA%85

Forest-앙상블효과에 따른 결 정경계



결정트리를 각 5번 돌려 앙상블한 랜

덤포레스트의 모습

- 각각의 Tree들의 모습만 봤을 때는 결정경계가 난잡한 것을 볼 수 있음
- 앙상블이 되었을 때는 결정경계가 훨씬 일반적이고 매끄러워져서 과적합이 해결되는 모습을 볼 수 있음

https://nonmeyet.tistory.com/entry/%EB%B2%A0%EA%B9%85Bagging-%EA%B3%BC-%EB%9E%9C%EB%8D%A4%ED%8F%AC%EB%A0%88%EC%8A%A4%ED%8A%B8%EC%9D%98-%EC%A0%95%EC%9D%98-%EB%B0%8F-%EC%84%A4%EB%AA%85

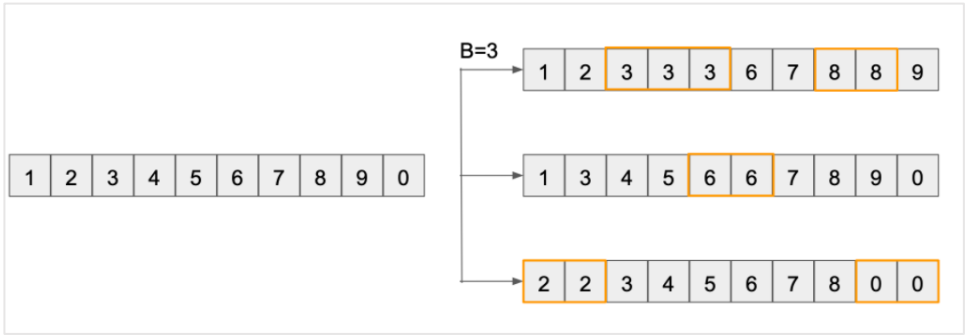
사용자 인식 데이터를 RandomForestClassifier을 통해 예측

개별 트리가 학습하는 데이터 세트는 전체 데이터에서 일부가 중첩되게 샘플링된 데이터 세트

여러 개의 데이터 세트를 중첩되게 분리하는 것을 **부트스트래핑(bootstrapping) 분할 방식**
(그래서 배깅(Bagging)이 bootstrap aggregating의 줄임말)

부트스트랩 bootstrap : 통계학에서 여러 개의 작은 데이터 세트를 임의로 만들어 개별 평균의 분포도를 측정하는 등의 목적을 위한 샘플링 방식

- 모집단으로부터 새로운 표본을 추출할 수 없을 때 모집단에서 독립적인 데이터셋을 반복하여 얻는 대신에 원래의 데이터셋(독립표본)으로부터 관측치를 반복적으로 추출하여 데이터셋을 얻는 기법. 즉, 복원추출법(중복허용)
- 부트스트랩을 사용함으로써 잡음이나 outlier로부터의 영향을 받지 않게 된다.



랜덤포레스트의 서브세트(subset) 데이터는 이러한 부트스트래핑으로 데이터가 임의로 만들어짐.
서브세트의 데이터 건수는 전체 데이터 건수와 동일하지만, 개별 데이터가 중첩되게 만들어짐

이렇게 데이터가 중첩된 개별 데이터 세트에 결정 트리 분류기를 각각 적용하는 것이 랜덤포레스트

랜덤포레스트 Random Forest

9주차 (p.56~p.60)의 코드를 실행 시킨 후,
(다음 페이지에 코드가 있으니 복사-붙여넣기 해주세요!)

```
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier

X_train, X_test, y_train, y_test = get_human_dataset()

# 예제 반복시마다 동일한 예측 결과 도출을 위해 난수값(random_state) 설정
dt_clf = DecisionTreeClassifier(random_state=0)
dt_clf.fit(X_train, y_train)
pred = dt_clf.predict(X_test)
accuracy = accuracy_score(y_test, pred)
print("Decision Tree 예측 정확도 : {0:.4f}".format(accuracy))
```

```
Decision Tree 예측 정확도 : 0.8595
```

: **Decision Tree**의 모델 학습/예측/평가
예측 정확도 = 0.8595

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
import pandas as pd
import warnings
warnings.filterwarnings('ignore')

# 결정 트리에서 사용한 get_human_dataset()를 이용해 학습/테스트용 DataFrame 반환
X_train, X_test, y_train, y_test = get_human_dataset()

# 랜덤 포레스트 학습 및 별도의 테스트 세트로 예측 성능 평가
rf_clf = RandomForestClassifier(random_state = 0)
rf_clf.fit(X_train, y_train)
pred = rf_clf.predict(X_test)
accuracy = accuracy_score(y_test, pred)
print("랜덤 포레스트 정확도 : {0:.4f}".format(accuracy))
```

```
랜덤 포레스트 정확도 : 0.9253
```

: **Random Forest**의 모델 학습/예측/평가
예측 정확도 = 0.9253

사용자 인식 데이터를 RandomForestClassifier을 통해 예측

9주차 (p.56~p.60)의 코드 :

```
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

# features.txt 파일에는 피처 이름 index와 피처명이 공백으로 분리되어 있음.
# 이를 DataFrame으로 로드
feature_name_df = pd.read_csv('./human_activity/features.txt', sep = '₩s+', header = None, names = ['column_index', 'column_name'])

# 피처명 index를 제거하고 피처명만 리스트 객체로 생성한 뒤 샘플로 10개만 추출
feature_name = feature_name_df.iloc[:,1].values.tolist()

feature_dup_df = feature_name_df.groupby('column_name').count()
print(feature_dup_df[feature_dup_df['column_index']>1].count())
feature_dup_df[feature_dup_df['column_index']>1].head()

def get_new_feature_name_df(old_feature_name_df):
    feature_dup_df = pd.DataFrame(data = old_feature_name_df.groupby('column_name').cumcount(), columns = ['dup_cnt'])

    feature_dup_df = feature_dup_df.reset_index()
    new_feature_name_df = pd.merge(old_feature_name_df.reset_index(), feature_dup_df, how='outer')
    new_feature_name_df['column_name'] = new_feature_name_df[['column_name', 'dup_cnt']].apply(lambda x : x[0]+'_'+str(x[1]) if x[1]>0 else x[0], axis = 1)
    new_feature_name_df = new_feature_name_df.drop(['index'], axis = 1)
    return new_feature_name_df

def get_human_dataset():
    feature_name_df = pd.read_csv('./human_activity/features.txt', sep = '₩s+', header=None, names=['column_index', 'column_name'])

    new_feature_name_df = get_new_feature_name_df(feature_name_df)

    feature_name = new_feature_name_df.iloc[:,1].values.tolist()

    X_train = pd.read_csv('./human_activity/train/X_train.txt', sep='₩s+', names = feature_name)
    X_test = pd.read_csv('./human_activity/test/X_test.txt', sep = '₩s+', names = feature_name)

    y_train = pd.read_csv('./human_activity/train/y_train.txt', sep='₩s+', header=None, names = ['action'])
    y_test = pd.read_csv('./human_activity/test/y_test.txt', sep = '₩s+', header=None, names = ['action'])

    return X_train, X_test, y_train, y_test
```

랜덤 포레스트 하이퍼 파라미터 및 튜닝

트리 기반의 앙상블 알고리즘의 단점은 **하이퍼 파라미터가 너무 많고**, 이로 인해 튜닝을 **위한 시간이 많이 소모**
더불어, 많은 시간을 소요했음에도 튜닝 후 예측 성능이 크게 향상 되는 경우가 많지 않음

하이퍼 파라미터	
n_estimators	랜덤 포레스트에서 결정 트리의 개수 지정. 디폴트는 10. 많이 설정할수록 좋은 성능을 기대할 수 있지만 계속 증가시킨다고 성능이 무조건 향상되는 것은 아님. 또한 늘릴수록 학습 수행 시간이 오래 걸림
max_features	결정 트리의 max_features 파라미터와 동일. 하지만, RandomForestClassifier의 기본 max_features는 'None'이 아닌 'auto'. 즉, 'sqrt'와 같음. 따라서 랜덤 포레스트의 트리를 분할하는 피처를 참조할 때 전체 피처가 아니라 sqrt 만큼 참조
max_depth	결정 트리의 max_depth와 동일 (과적합을 개선)
min_samples_leaf	결정 트리의 min_samples_leaf와 동일 (과적합을 개선)

나머지 하이퍼 파라미터는 결정 트리와 동일

랜덤 포레스트 하이퍼 파라미터 및 튜닝

GridSearchCV를 이용해 랜덤 포레스트의 하이퍼 파라미터 튜닝 (튜닝 시간 절약을 위해 n_estimators = 100)

```
from sklearn.model_selection import GridSearchCV

params = {
    'n_estimators' : [100],
    'max_depth' : [6,8,10,12],
    'min_samples_leaf' : [8,12,18],
    'min_samples_split' : [8,16,20]
}

# RandomForestClassifier 객체 생성 후 GridSearchCV 수행
rf_clf = RandomForestClassifier(random_state=0, n_jobs = -1)
grid_cv = GridSearchCV(rf_clf, param_grid = params, cv = 2, n_jobs = -1)
grid_cv.fit(X_train, y_train)

print("최적의 하이퍼 파라미터 : ₩n", grid_cv.best_params_)
print("최고 예측 정확도 : {0:.4f}".format(grid_cv.best_score_))
```

```
최적의 하이퍼 파라미터 :
 {'max_depth': 10, 'min_samples_leaf': 8, 'min_samples_split': 8, 'n_estimators': 100}
최고 예측 정확도 : 0.9180
```

최적의 하이퍼 파라미터 :
n_estimators 가 100, max_depth가 10,
min_samples_leaf가 8, min_samples_split가 8

```
rf_clf1 = RandomForestClassifier(n_estimators = 300, max_depth = 10, min_samples_leaf = 8,
                                 min_samples_split = 8, random_state = 0)
rf_clf1.fit(X_train, y_train)
pred = rf_clf1.predict(X_test)
print("예측 정확도 : {0:.4f}".format(accuracy_score(y_test, pred)))
```

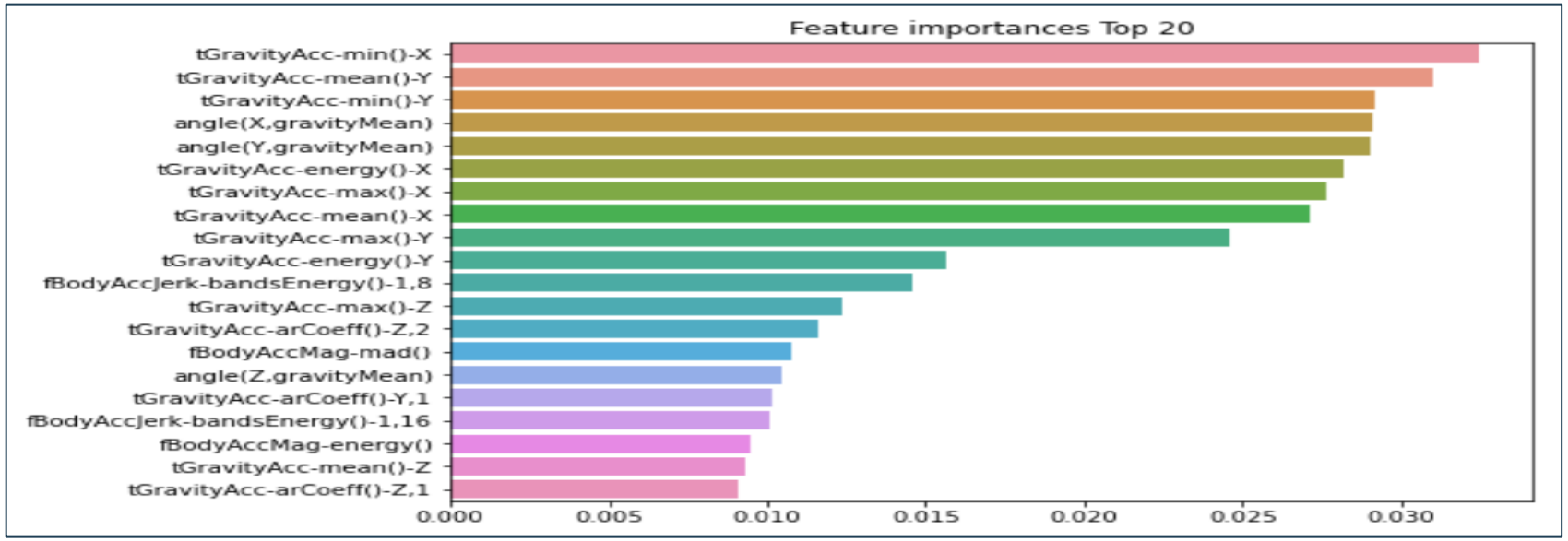
```
예측 정확도 : 0.9165
```

n_estimators = 300으로 증가시키고,
최적화 하이퍼 파라미터로
RandomForestClassifier를 학습시킨 후 예측 성능 확인

랜덤 포레스트 피쳐 중요도 확인

```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

ftr_importances_values = rf_clf1.feature_importances_
ftr_importances = pd.Series(ftr_importances_values, index = X_train.columns)
ftr_top20 = ftr_importances.sort_values(ascending = False)[:20]

plt.figure(figsize = (8,6))
plt.title('Feature importances Top 20')
sns.barplot(x = ftr_top20, y = ftr_top20.index)
plt.show()
```

feature_importances_ 속성을 이용



5. 실습

아이리스 데이터 분석 : KNN

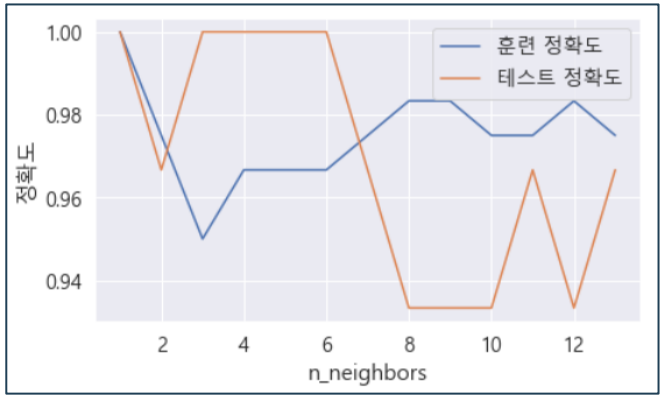
```
import pandas as pd
import numpy as np
from sklearn.datasets import load_iris

df = pd.read_csv('iris.csv')
X=df.drop('species',axis=1)
y=df['species']
```

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix

X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.2,
                                                    random_state=9, stratify=y)

knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
pred = knn.predict(X_test)

print('KNN training set 정확도:',knn.score(X_train, y_train))
print('KNN test set 정확도:',accuracy_score(y_test, pred))
confusion_matrix(y_test,pred)
```

```
KNN training set 정확도: 0.95
KNN test set 정확도: 1.0

array([[10,  0,  0],
       [ 0, 10,  0],
       [ 0,  0, 10]], dtype=int64)
```

아이리스 데이터 분석 : KNN

```
from matplotlib import font_manager, rc
font_path = "malgun.ttf"
font_name = font_manager.FontProperties(fname=font_path).get_name()
rc("font", family=font_name)

training_accuracy = []
test_accuracy = []
# 1에서 10까지 n_neighbors를 적용
neighbors_settings = range(1, 14)

for n_neighbors in neighbors_settings:
    knn = KNeighborsClassifier(n_neighbors=n_neighbors)
    knn.fit(X_train, y_train)
    # 훈련 세트 정확도 저장
    training_accuracy.append(knn.score(X_train, y_train))
    # 일반화 정확도 저장
    test_accuracy.append(knn.score(X_test, y_test))

plt.figure(figsize=(7,4))
plt.plot(neighbors_settings, training_accuracy, label="훈련 정확도")
plt.plot(neighbors_settings, test_accuracy, label="테스트 정확도")
plt.ylabel("정확도")
plt.xlabel("n_neighbors")
plt.legend(loc='bottom right')
```




```
from sklearn.model_selection import GridSearchCV

grid_params = {'n_neighbors':[3,5,7,9],
               'weights':['uniform','distance'],
               'metric':['euclidean','manhattan']}
gs = GridSearchCV(knn,grid_params, cv=3)
gs_results = gs.fit(X_train, y_train)

gs_results.best_estimator_
scores_df = pd.DataFrame(gs_results.cv_results_)
print(scores_df[['params','mean_test_score','rank_test_score',
          'split0_test_score','split1_test_score',
           'split2_test_score']].sort_values(by=['rank_test_score']).head())


params = scores_df['params'][scores_df['rank_test_score']==1]
print('₩n')
for i in range(len(params)):
    print(params.iloc[i])
```

```
                                              params  mean_test_score  ₩
4  {'metric': 'euclidean', 'n_neighbors': 7, 'wei...         0.975000
5  {'metric': 'euclidean', 'n_neighbors': 7, 'wei...         0.975000
7  {'metric': 'euclidean', 'n_neighbors': 9, 'wei...         0.975000
2  {'metric': 'euclidean', 'n_neighbors': 5, 'wei...         0.966667
3  {'metric': 'euclidean', 'n_neighbors': 5, 'wei...         0.966667

   rank_test_score  split0_test_score  split1_test_score  split2_test_score
4                1              1.000              0.975              0.950
5                1              1.000              0.950              0.975
7                1              0.975              0.975              0.975
2                4              0.975              0.950              0.975
3                4              0.975              0.950              0.975


{'metric': 'euclidean', 'n_neighbors': 7, 'weights': 'uniform'}
{'metric': 'euclidean', 'n_neighbors': 7, 'weights': 'distance'}
{'metric': 'euclidean', 'n_neighbors': 9, 'weights': 'distance'}
```

아이리스 데이터 분석 : SVM

Kernel = “linear”
C=1

```
from sklearn.svm import SVC

svm = SVC(kernel="linear", C=1)
svm.fit(X_train, y_train)
pred = svm.predict(X_test)

print('SVM training set 정확도:',svm.score(X_train, y_train))
print('SVM test set 정확도:',accuracy_score(y_test, pred))
confusion_matrix(y_test,pred)
```

```
SVM training set 정확도: 0.9916666666666667
SVM test set 정확도: 1.0

array([[10,  0,  0],
       [ 0, 10,  0],
       [ 0,  0, 10]], dtype=int64)
```

아이리스 데이터 분석 : 랜덤포레스트

Iris 데이터를 불러와 확인

```
import pandas as pd
import numpy as np

df = pd.read_csv("iris.csv")
df.head(10)
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
5	5.4	3.9	1.7	0.4	setosa
6	4.6	3.4	1.4	0.3	setosa
7	5.0	3.4	1.5	0.2	setosa
8	4.4	2.9	1.4	0.2	setosa
9	4.9	3.1	1.5	0.1	setosa

Feature를 X에, label을 y에 저장하고
75%:25%의 비율로 train, test 분할

```
from sklearn.model_selection import train_test_split

X = df[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']]
y = df['species']

# 75% : 25%
X_train, X_test, y_train, y_test = train_test_split(X, y)
```

Label의 개수 파악

```
y.value_counts()
```

```
versicolor    50
setosa        50
virginica     50
Name: species, dtype: int64
```

아이리스 데이터 분석 : 랜덤포레스트

Decision Tree로 학습/예측/평가

```
# iris 구분 Decision Tree 모델링
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

dt_clf = DecisionTreeClassifier()
dt_clf.fit(X_train, y_train)
dt_pred = dt_clf.predict(X_test)
dt_accuracy = accuracy_score(y_test, dt_pred)

print("의사결정나무 training set 정확도 : ", dt_clf.score(X_train, y_train))
print("의사결정나무 test set 정확도 : ", dt_accuracy)
# training set 정확도가 1.0
```

```
의사결정나무 training set 정확도 :  1.0
의사결정나무 test set 정확도 :  0.8947368421052632
```

Random Forest로 학습/예측/평가

```
# iris 구분 RandomForest Tree 모델링
from sklearn.ensemble import RandomForestClassifier

rf_clf = RandomForestClassifier(n_estimators=100)
rf_clf.fit(X_train, y_train)
rf_pred = rf_clf.predict(X_test)
rf_accuracy = accuracy_score(y_test, rf_pred)

print("랜덤포레스트 training set 정확도 : ", rf_clf.score(X_train, y_train))
print("랜덤포레스트 test set 정확도 : ", rf_accuracy) # "Decision Tree보다 좋은 성능"
```

```
랜덤포레스트 training set 정확도 :  1.0
랜덤포레스트 test set 정확도 :  0.9210526315789473
```

Q&A