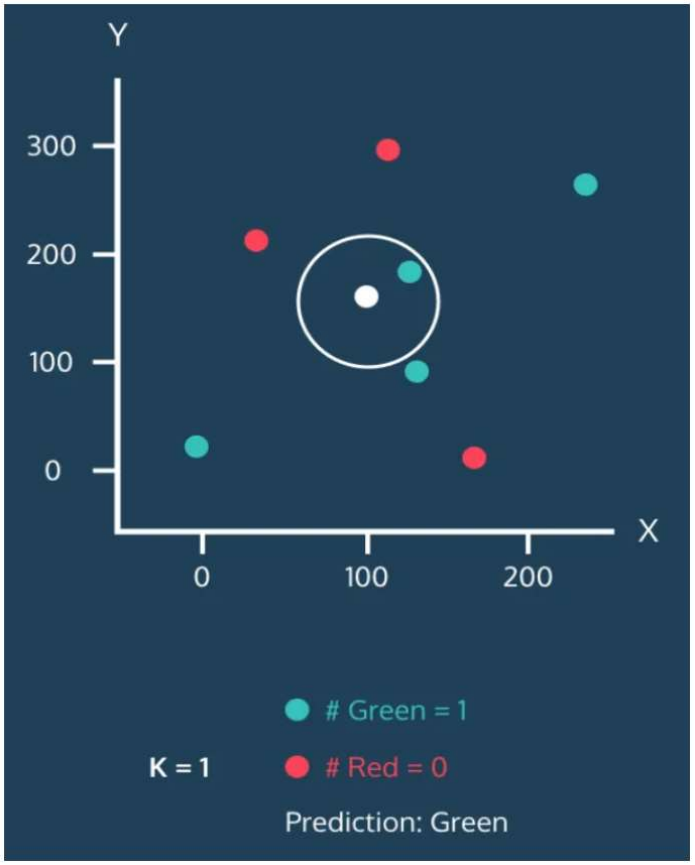
k-최근접 이웃(KNN) 분류

게으른 학습기
기술적으로 예측을 만들기 위해 모델을 훈련하지 않고
가장 가까운 k 개의 샘플에서 다수의 클래스를 그 샘플의 클래스로 예측합니다.
예:클래스가 알려지지 않은 한 샘플이 클래스 1인 샘플로 둘러싸여 있다면 그 샘플을 클래스 1로 분류합니다.

➢ K-최근접 이웃(K-Nearest Neighbor)

- 분류(Classification) 알고리즘
- 유사한 특성을 가진 데이터는 유사한 범주에 속하는 경향이 있다는 가정하에 사용한다.



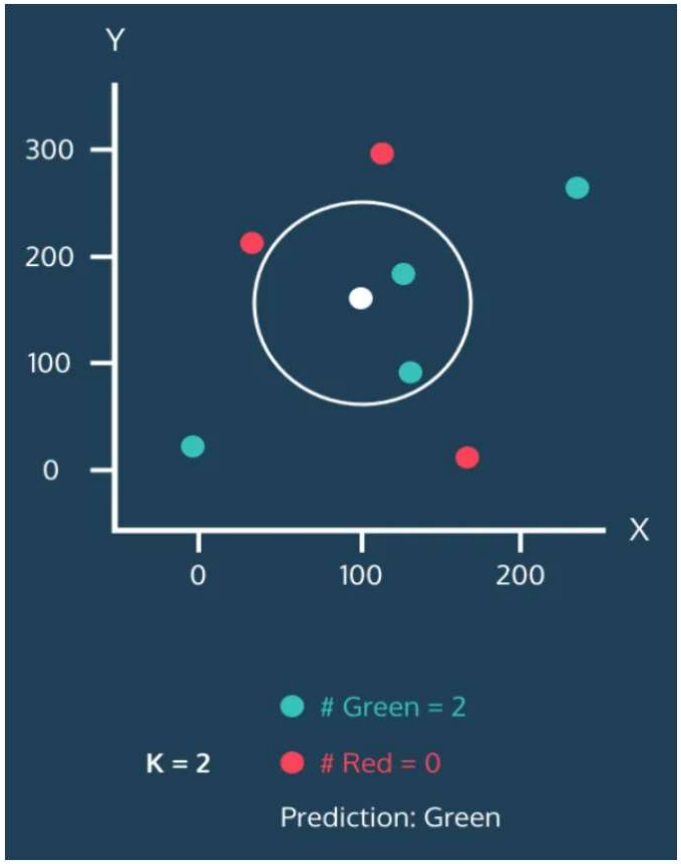
모든 데이터(점)에는 각각 x값과 y값이 있다.
점의 색상으로 초록/빨강으로 표시하여 분류를 나타냈다.
하얀 점은 아직 분류가 안 된 새로운 데이터다.

하얀 점 주의로 원을 그려놓았고, 이 원 안에는 1개의 이웃이 있다.
원 안에 포함될 이웃의 개수를 k
k=1 이고, 하얀 점은 초록으로 분류된다

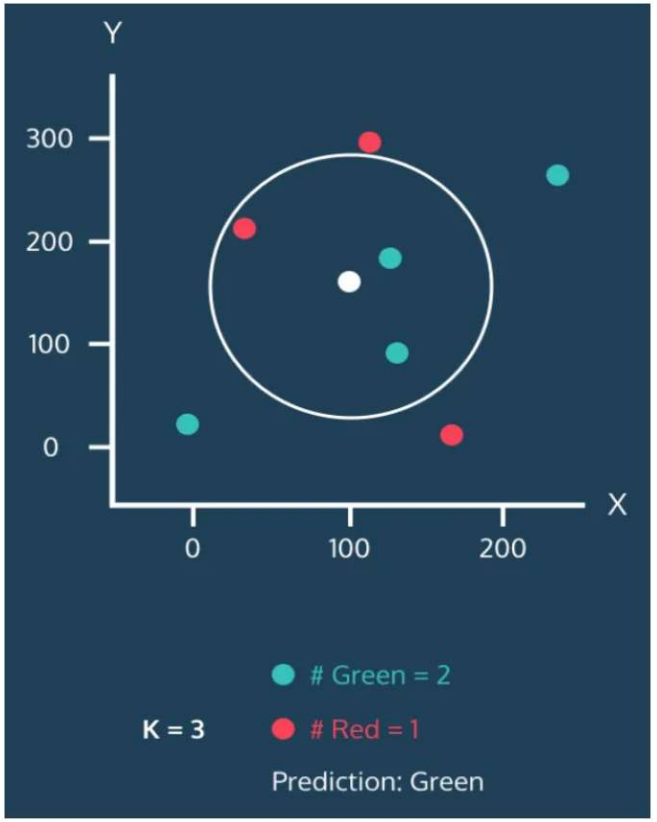
K-최근접 이웃(K-Nearest Neighbor) 알고리즘의 목적은 새로운 점이 등장했을 때 이걸 초록이나 빨강으로 분류한다.

➢ K-최근접 이웃(K-Nearest Neighbor)

•

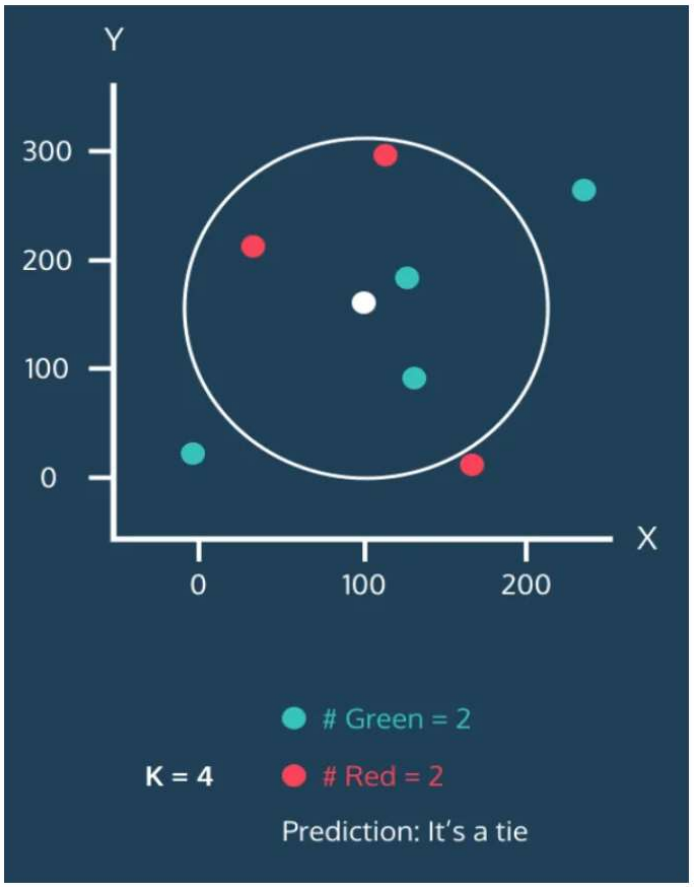


k를 2로 조정해서 가장 가까운 이웃의 수를 2개까지 늘린 경우 이웃 2개 모두 초록이므로 하얀점은 초록으로 분류된다.

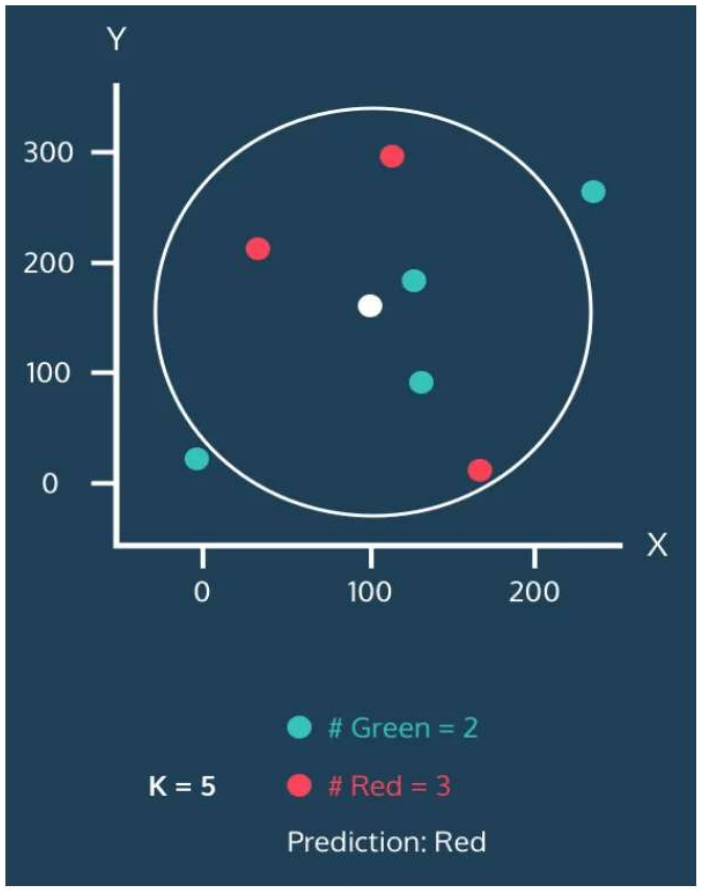


k를 3로 조정해서 가장 가까운 이웃의 수를 3개까지 늘린 경우 초록 2개, 빨강 1개다. 초록이 많기 때문에 하얀 점은 초록으로 분류된다.

➢ K-최근접 이웃(K-Nearest Neighbor)

- 

동률(tie)이 나왔을 때 처리하는 몇가지 방법 :
가장 가까운 이웃을 따르거나, 랜덤으로 분류



k를 5로 늘리면 원이 확장되면서 이웃 중 빨강이 3개, 초록이 2개이므로 하얀 점이 비로소 빨강으로 분류가 된다.

➢ K-최근접 이웃(K-Nearest Neighbor)

- K-Nearest Neighbor 알고리즘을 사용할 때는 모든 특성들을 모두 고르게 반영하기 위해 정규화(Noramlizaion)을 해주곤 합니다
- 최소값을 0, 최대값을 1로 고정한 뒤 모든 값들을 0과 1사이 값으로 변환하는 방법
- 평균과 표준편차를 활용해서 평균으로부터 얼마나 떨어져 있는지 z-점수로 변환하는 방법
- K 개수 선택 - 모든 값을 실제로 테스트하면서 분류 정확도(Accuracy)를 계산하는 과정에서 찾을 수 있습니다.
- K-Nearest Neighbors 알고리즘에서는 주변 다른 이웃들까지 충분히 고려하지 않았을 때 오버피팅이 발생한다.
- k가 너무 큰 경우에는 underfitting(과소적합)이 발생한다

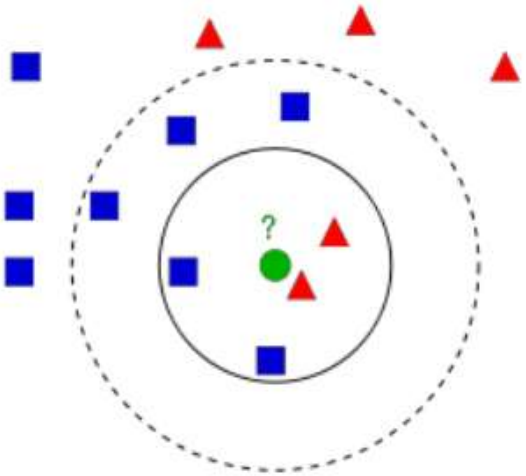
➢ **K-최근접 이웃(K-Nearest Neighbor)**

- n개의 특성(feature)을 가진 데이터는 n차원의 공간에 점으로 개념화 할 수 있다.
- 유사한 특성을 가진 데이터들끼리는 거리가 가깝다. 그리고 거리 공식을 사용하여 데이터 사이의 거리를 구할 수 있다.
- 분류를 알 수 없는 데이터에 대해 가장 가까운 이웃 k개의 분류를 확인하여 다수결을 할 수 있다.
- 분류기의 효과를 높이기 위해 파라미터를 조정할 수 있다.
- K-Nearest Neighbors의 경우 k 값을 변경할 수 있다.
- 분류기가 부적절하게 학습되면 overfitting 또는 underfitting이 나타날 수 있다.
- K-Nearest Neighbors의 경우 너무 작은 k는 overfitting, 너무 큰 k는 underfitting을 야기한다.

kNN(k-Nearest Neighbors)

➢ kNN(k-Nearest Neighbors)

- 지도 학습에 활용되는 가장 단순한 종류의 알고리즘
- 파란색 사각형, 빨간색 삼각형은 각각 동일한 특성을 가진 특정 그룹이라고 가정합니다.
- 파란색 사각형들과 빨간색 삼각형들은 2차원 평면에 불규칙적으로 분포되어 있습니다.
- 만약 3가지 특성으로 구분되는 것이라면 3차원 공간에 표현하며 이와 같은 공간을 특성 공간(feature space)이라 부릅니다.
- 새로운 멤버 초록색 원은 파란색 사각형 또는 빨간색 삼각형 중 하나가 되야 합니다.
- 새로운 멤버와 가장 가깝게 위치하는 멤버가 속해 있는 그룹으로 분류하는 방법을 Nearest Neighbour라고 합니다.



- 예] 빨간색 삼각형보다 파란색 사각형이 더 많이 분포하고 있고, 특히 새로운 멤버 근처에 제일 가깝게 있는 녀석은 빨간색 삼각형이지만 조금만 범위를 넓혀 보면 파란색 사각형이 더 많습니다. 새로운 멤버인 초록색 원과 4번째까지 가까운 멤버들의 분포를 보고 판단하면 파란색 사각형으로 분류할지 빨간색 삼각형으로 분류할지 판단할 수 없습니다. 조금 더 범위를 확대해서 7번째까지 가까운 멤버들의 분포를 보고 판단하면 초록색 원은 파란색 사각형으로 분류해야 합니다.
- 주어진 개수만큼 가까운 멤버들과 비교하여 판단하는 방법을 k-Nearest Neighbours 알고리즘이라 부릅니다.
- 주어지는 멤버 개수를 k로 표현한 것입니다.
- 가까운 멤버에는 가중치를 높게, 멀리 떨어져 있는 멤버에는 가중치를 낮게 하는 것이 바람직합니다.

kNN(k-Nearest Neighbors)

- 샘플의 최근접 이웃 찾기
  - 샘플에서 가장 가까운 k개의 샘플(이웃) 찾기

```
from sklearn import datasets
from sklearn.neighbors import NearestNeighbors
from sklearn.preprocessing import StandardScaler

iris = datasets.load_iris()  # 데이터 로드
features = iris.data
standardizer = StandardScaler()  # 표준화 객체 생성
features_standardized = standardizer.fit_transform(features)  # 특성을 표준화

# k=2인 최근접 이웃 모델 생성
nearest_neighbors = NearestNeighbors(n_neighbors=2).fit(features_standardized)
new_observation = [ 1,  1,  1,  1]  #New Sample Data
# New 샘플과 가장 가까운 이웃의 인덱스와 거리를 찾습니다.
distances, indices = nearest_neighbors.kneighbors([new_observation])
features_standardized[indices]  # 최근접 이웃을 확인

nearestneighbors_euclidean = NearestNeighbors( n_neighbors=2, metric='euclidean').fit(features_standardized)
distances  # 거리  확인
# 유클리디안 거리를 기반으로 각 샘플에 대해 (자기 자신을 포함한) 세 개의 최근접 이웃을 찾습니다.
nearestneighbors_euclidean = NearestNeighbors(n_neighbors=3, metric="euclidean").fit(features_standardized)
```

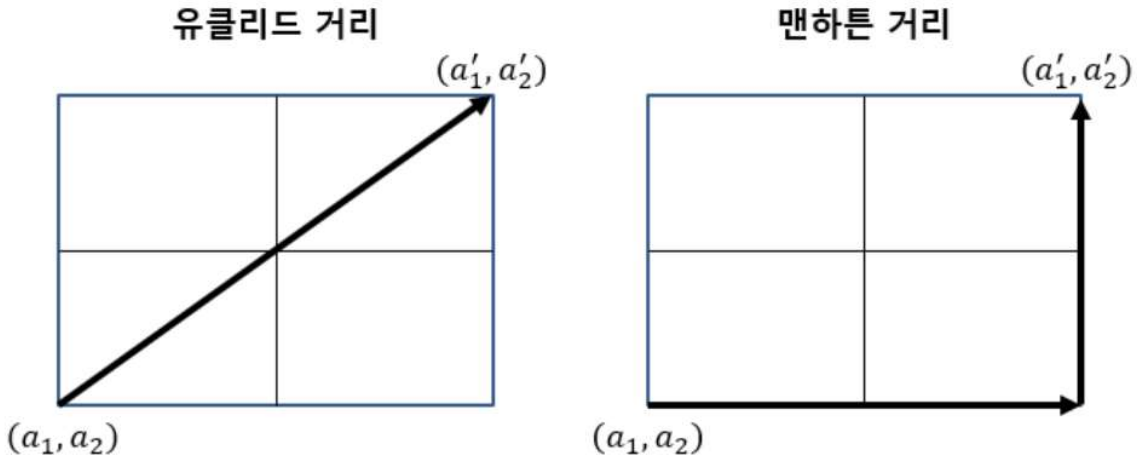
kNN(k-Nearest Neighbors)

- 샘플의 최근접 이웃 찾기

```
# 각 샘플의 (자기 자신을 포함한) 3개의 최근접 이웃을 나타내는 리스트의 리스트
nearest_neighbors_with_self = nearestneighbors_euclidean.kneighbors_graph( features_standardized).toarray()
# 최근접 이웃 중에서 1로 표시된 자기 자신을 제외시킵니다.
for i, x in enumerate(nearest_neighbors_with_self):
    x[i] = 0

# 첫 번째 샘플에 대한 두 개의 최근접 이웃을 확인합니다.
nearest_neighbors_with_self[0]

# 이 샘플과 가장 가까운 이웃의 다섯개의 인덱스를 찾습니다.
indices = nearest_neighbors.kneighbors([new_observation], n_neighbors=5, return_distance=False)
features_standardized[indices]  # 최근접 이웃을 확인

# 반경 0.5 안에 있는 모든 샘플의 인덱스를 찾습니다.
indices = nearest_neighbors.radius_neighbors( [new_observation], radius=0.5, return_distance=False)
features_standardized[indices[0]]  # 반경 내의 이웃을 확인

# 반경 내의 이웃을 나타내는 리스트의 리스트
nearest_neighbors_with_self = nearest_neighbors.radius_neighbors_graph(  [new_observation], radius=0.5).toarray()
nearest_neighbors_with_self[0]   # 첫 번째 샘플에 대한 반경 내의 이웃을 확인
```

kNN(k-Nearest Neighbors)

- kNN(k-Nearest Neighbors)
  - KNeighborsClassifier(n_neighbors, p, metric)의 매개변수 metric='minkowski'는 유클리드 거리(Euclidean distance)와 맨하튼 거리(Manhattan distance)를 일반화 한 것입니다.



매개변수 p=1이면 맨하튼 거리를 나타내며 , p=2 이면 유클리드 거리를 나타냅니다.

$$d = \sqrt{(a_1' - a_1)^2 + (a_2' - a_2)^2} \qquad d = (a_1' - a_1) + (a_2' - a_2)$$

- 유클리드 거리는 좌표계에 두 점이 있을 때 두 지점의 최단거리이며, 맨하튼 거리는 격자를 이루는 선이 길이라고 생각하고 그 길이를 따라 잰 거리를 말합니다.
- minkowski 거리는 이 두 거리를 하나의 식으로 나타낸 것입니다

$$d = \sqrt[p]{\sum_k (a_k' - a_k)^p}$$

kNN(k-Nearest Neighbors)

- K-최근접 이웃 분류기(KNeighborsClassifier)
  - 클래스를 모르는 샘플이 주어졌을 때 이웃한 샘플의 클래스를 기반으로 이 샘플의 클래스를 예측해야 합니다.

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn import datasets

iris = datasets.load_iris()   # 데이터 로드
X = iris.data
y = iris.target

standardizer = StandardScaler()  # 표준화 객체
X_std = standardizer.fit_transform(X)  # 특성을 표준화
# 5개의 이웃을 사용한 KNN 분류기를 훈련합니다.
knn = KNeighborsClassifier(n_neighbors=5, n_jobs=-1).fit(X_std, y)
new_observations = [[ 0.75,  0.75,  0.75,  0.75],
                    [ 1,  1,  1,  1]]    # 두 개의 샘플을 만듭니다.

knn.predict(new_observations)   # 두 샘플의 클래스를 예측
knn.predict_proba(new_observations) # 각 샘플이 세 클래스에 속할 확률을 확인
```

kNN(k-Nearest Neighbors)

- K-최근접 이웃 분류기(KNeighborsClassifier)
  - metric 매개변수는 사용할 거리 측정 방법을 지정합니다.
  - n_jobs 매개변수는 컴퓨터 코어를 사용할지 결정합니다.
  - algorithm 매개변수는 가장 가까운 이웃을 계산하기 위한 방법을 지정
  - weights 매개변수를 distance로 지정하면 멀리 떨어진 샘플보다 가까운 이웃의 투표에 가중치가 더 부여됩니다

```
from sklearn.neighbors import KNeighborsRegressor
from sklearn import datasets

boston = datasets.load_boston()  # 데이터 로드
features = boston.data[:,0:2]   #두 개의 특성만 선택
target = boston.target

knn_regressor = KNeighborsRegressor(n_neighbors=10)  # 최근접 회귀 모델 객체 생성
model = knn_regressor.fit(features, target)  # 모델 훈련
# 첫 번째 샘플의 타깃 값을 예측하고 1000을 곱합니다.
model.predict(features[0:1])[0]*1000

import numpy as np

indices = model.kneighbors(features[0:1], return_distance=False)
np.mean(target[indices]) * 1000
```

kNN(k-Nearest Neighbors)

- 최선의 이웃 개수 결정
  - KNN 분류기에 각기 다른 k 값으로 5-폴드 교차검증을 수행하는 GridSearchCV를 사용

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline, FeatureUnion
from sklearn.model_selection import GridSearchCV

iris = datasets.load_iris()   # 데이터 로드
features = iris.data
target = iris.target

standardizer = StandardScaler()                    # 표준화 객체 생성
knn = KNeighborsClassifier(n_neighbors=5, n_jobs=-1)      # KNN 분류기 객체 생성
pipe = Pipeline([("standardizer", standardizer), ("knn", knn)])          # 파이프라인 생성
# 탐색 영역의 후보를 만듭니다.
search_space = [{"knn__n_neighbors": [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]}]
# 그리드 서치 객체 생성
classifier = GridSearchCV( pipe, search_space, cv=5, verbose=0).fit(features, target)
# 최선의 이웃 개수 (k)
classifier.best_estimator_.get_params()["knn__n_neighbors"]
```

kNN(k-Nearest Neighbors)

- 반지름 기반의 최근접 이웃 분류기(RadiusNeighborsClassifier)
  - 샘플의 클래스가 주어진 반지름 r 이내에 있는 모든 샘플의 클래스로부터 예측됩니다.
  - radius 매개변수로 고정 영역의 반지름을 지정하여 이웃 샘플을 결정합니다.
  - outlier_label 매개변수는 반지름 내에 다른 샘플이 하나도 없는 샘플에 부여할 레이블을 지정합니다.

```
from sklearn.neighbors import RadiusNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn import datasets

iris = datasets.load_iris()  # 데이터 로드
features = iris.data
target = iris.target
standardizer = StandardScaler()  # 표준화 객체 생성
features_standardized = standardizer.fit_transform(features)  # 특성을 표준화

# 반지름 이웃 분류기를 훈련합니다.
rnn = RadiusNeighborsClassifier( radius=.5, n_jobs=-1).fit(features_standardized, target)
new_observations = [[ 1,  1,  1,  1]]  # 두 개의 샘플을 만듭니다.
rnn.predict(new_observations)  # 두 샘플의 클래스를 예측
# 반지름 이웃 분류기를 훈련합니다.
rnn = RadiusNeighborsClassifier(  radius=.5, outlier_label=-1, n_jobs=-1).fit(features_standardized, target)
rnn.predict([[100, 100, 100, 100]])
```