

K-Nearest Neighbors (KNN)

K-최근접이웃

분류와 회귀

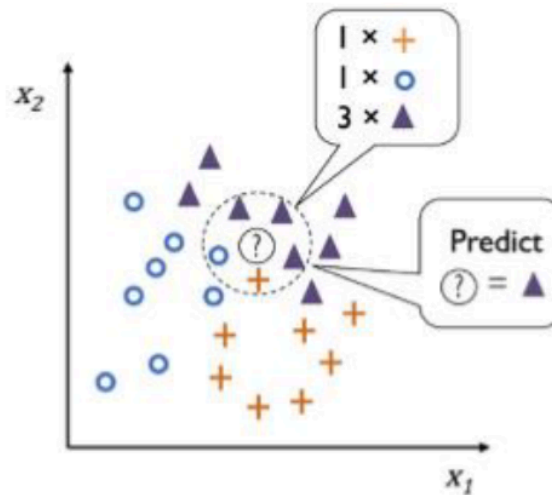
- 지도 학습의 대표적인 머신 러닝 방법
 - 분류 (classification)
 - 회귀 (regression)
- 분류
 - 분류는 미리 정의된, 가능성 있는 여러 클래스 레이블(class label) 중 하나를 예측하는 것
 - 두 개로만 나누는 이진 분류(Binary classification)와 셋 이상의 클래스로 분류하는 다중 분류(multiclass classification)로 나뉨
 - 분류 예시: 얼굴 인식, 숫자 판별 (MNIST) 등
- 회귀
 - 연속적인 숫자 또는 부동소수점수 (실수)를 예측하는 것
 - 회귀 예시: 주식 가격을 예측하여 수익을 내는 알고리즘 등

KNN의 개념

KNN의 개념

- KNN이란?

- 주변 k 개의 자료의 클래스(class) 중 가장 많은 클래스로 특정 자료를 분류하는 방식
- 새로운 자료 ? 를 가장 가까운 자료 5개의 자료 ($k=5$) 를 이용하여 투표하여 가장 많은 클래스로 할당



- Training-data 자체가 모형일 뿐 어떠한 추정 방법도 모형도 없음
 - 즉, 데이터의 분포를 표현하기 위한 파라미터를 추정하지 않음
- 매우 간단한 방법이지만 performance는 떨어지지 않음

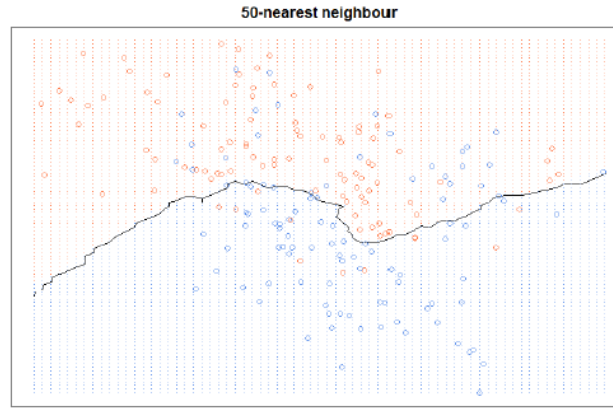
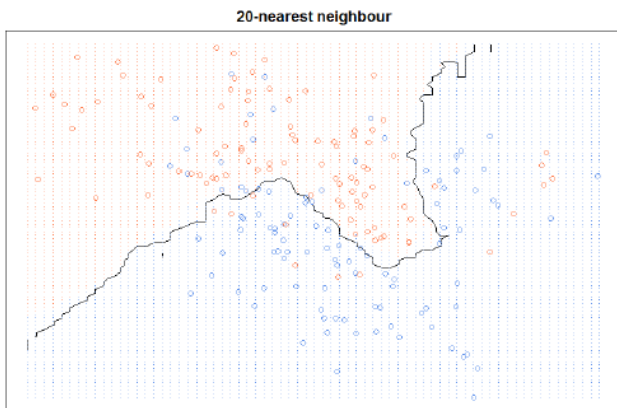
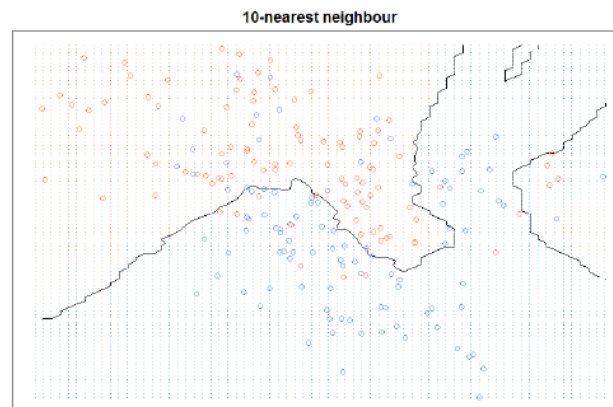
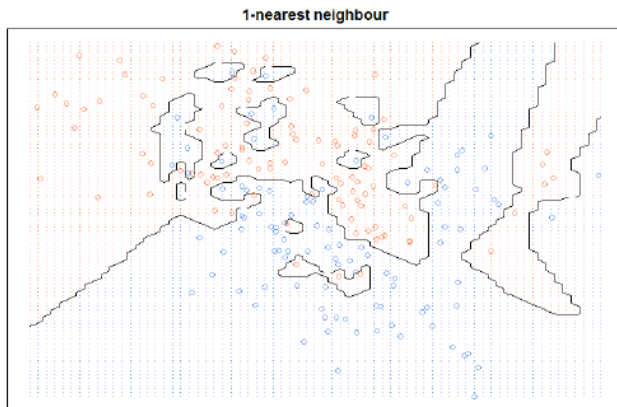
KNN의 개념

- KNN이란?
 - 게으른 학습(lazy learner) 또는 사례중심학습(instance-based learning)
 - 게으른 학습이란: 알고리즘은 훈련 데이터에서 판별 함수(discriminative function)를 학습하는 대신 훈련 데이터 셋을 메모리에 저장하기 방법
 - 데이터의 차원이 증가하면 차원의 저주(curse of dimension) 문제가 발생함
 - 즉, KNN은 차원이 증가할 수록 성능 저하가 심함
 - 데이터의 차원(dimensionality)이 증가할수록 해당 공간의 크기(부피)가 기하급수적으로 증가하여 동일한 개수의 데이터의 밀도는 차원이 증가할수록 급속도로 희박(sparse)해짐
 - 차원이 증가할수록 데이터의 분포 분석에 필요한 샘플 데이터의 개수가 기하급수적으로 증가하게 되는데 이러한 어려움을 표현한 용어가 차원의 저주임
 - i번째 관측치와 j번째 관측치의 거리로 Minkowski 거리를 이용

$$d(\mathbf{x}_i, \mathbf{x}_j) = \sqrt[p]{\sum_{k=1}^d |x_{ik} - x_{jk}|^p}$$

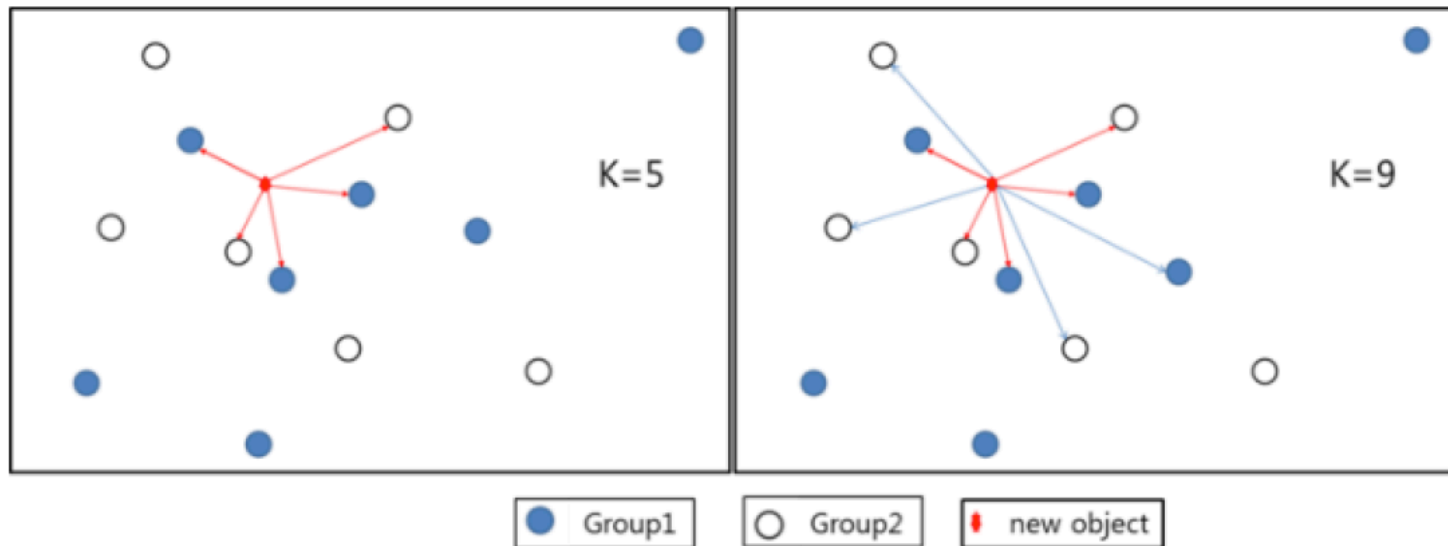
KNN의 하이퍼파라미터

- 탐색할 이웃 수(k)와 거리 측정 방법
 - k가 작을 경우 데이터의 지역적 특성을 지나치게 반영하여 **과적합(overfitting)** 발생
 - 반대로 매우 클 경우 모델이 과하게 **정규화 (underfitting)** 발생



KNN의 K가 가지는 의미

- 새로운 자료에 대해 근접치 K의 개수에 따라 Group이 달리 분류됨
 - 다수결 방식 (Majority voting): 이웃 범주 가운데 빈도 기준 제일 많은 범주로 새 데이터의 범주를 예측하는 것



- 가중 합 방식 (Weighted voting): 가까운 이웃의 정보에 좀 더 가중치를 부여

KNN의 장단점 요약

// 장점

- 학습데이터 내에 끼어있는 노이즈의 영향을 크게 받지 않음
- 학습데이터 수가 많다면 꽤 효과적인 알고리즘
- 마할라노비스 거리와 같이 데이터의 분산을 고려할 경우 매우 강건(robust)한 방법론
 - 마할라노비스 거리(Mahalanobis distance)는 평균과의 거리가 표준편차의 몇 배인지를 나타내는 값
 - 즉, 어떤 값이 얼마나 일어나기 힘든 값인지, 또는 얼마나 이상한 값인지를 수치화하는 한 방법

// 단점

- 최적 이웃의 수(k)와 어떤 거리 척도(distance metric)가 분석에 적합한지 불분명해
데이터 각각의 특성에 맞게 연구자가 임의로 선정해야 함
 - best K는 데이터 마다 다르기 때문에 탐욕적인 방식(Grid Search)으로 탐색
- 새로운 관측치와 각각의 학습 데이터 사이의 거리를 전부 측정해야 하므로 계산 시간이 오래 걸리는 한계
- KNN의 계산복잡성을 줄이려는 시도들
 - Locality Sensitive Hashing, Network based Indexer, Optimized product quantization

KNN의 적용

Classification

기계 학습의 일반적인 실습 순서

- 데이터셋 불러오기
 - seaborn 라이브러리 사용, 유명한 데이터 셋 대부분 지원 (예. Iris)
- 데이터셋 카테고리의 실수화
 - setosa, versicolor, virginica → "0", "1", "2"
- 데이터 분할
 - 학습데이터와 테스트 데이터로 나누기
- (옵션) 입력데이터의 표준화
- 모형 추정 혹은 사례중심학습
- 결과 분석
 - Confusion matrix 로 확인

Iris 데이터셋 불러오기

- Iris 데이터셋이란?

- 데이터명 : IRIS (아이리스, 붓꽃 데이터)
- 레코드수 : 150개
- 필드개수 : 5개
- 데이터설명 : 아이리스(붓꽃) 데이터에 대한 데이터. 꽃잎의 각 부분의 너비와 길이 등을 측정한 데이터이며 150개의 레코드로 구성되어 있음.

- 필드의 이해 :

총 6개의 필드로 구성되어있음. caseno는 단지 순서를 표시하므로 분석에서 제외. 2번째부터 5번째의 4개의 필드는 **입력 변수(X)**로 사용되고, 맨 아래의 Species 속성이 **목표(종속) 변수(Y)**로 사용된다.



	caseno	SepalLength	SepalWidth	PetalLength	PetalWidth	Species
1	1	5.1	3.5	1.4	.2	setosa
2	2	4.9	3.0	1.4	.2	setosa
3	3	4.7	3.2	1.3	.2	setosa

Iris 데이터셋 불러오기



```
1 # 3장 KNN
2 import seaborn as sns # seaborn을 불러오고 sns로 축약함.
3 iris=sns.load_dataset('iris') # iris라는 변수명으로 Iris data를 download함.
4 print(iris.head()) # 최초의 5개의 관측치를 print
```



	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



```
1 print(iris.shape) # iris data의 행과 열의 수
2
3 X = iris.drop('species', axis=1) # 'species'열을 drop하고 input x를 정의함.
4 print(X.shape)
5
6 y=iris['species'] # 'species'열을 label y를 정의함.
```

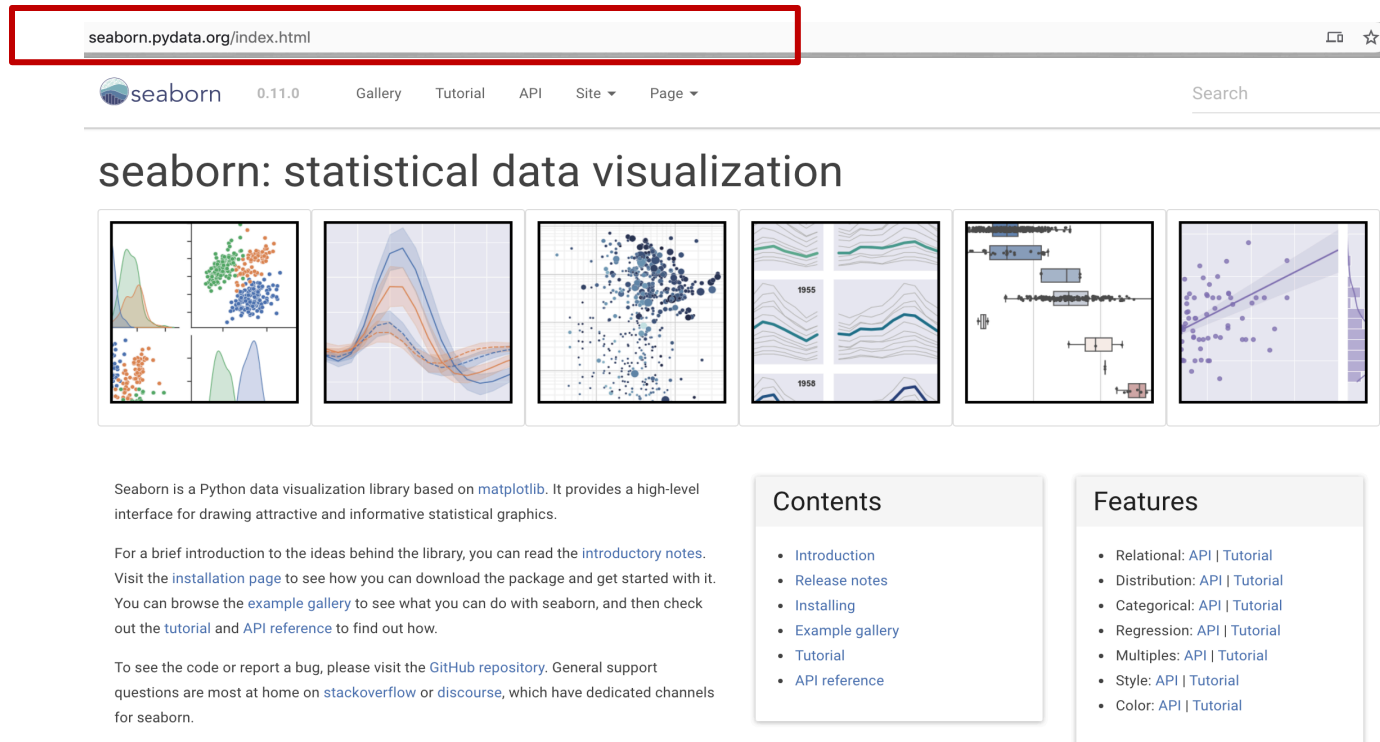


```
(150, 5)
(150, 4)
```

Iris 데이터셋 불러오기

Seaborn 라이브러리란?

- 파이썬에서 데이터 시각화를 담당하는 모듈
- 유익한 통계 그래픽을 그리기 위한 고급 인터페이스를 제공
- 파이썬 사용자들이 약간의 변수와 파라미터 조정으로 쉽게 그래프를 표현해 볼 수 있게 해주는 도구

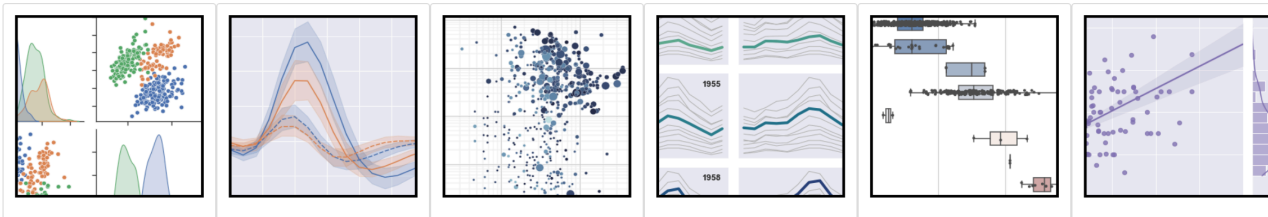


The screenshot shows the Seaborn website interface. At the top, the URL `seaborn.pydata.org/index.html` is highlighted in a red box. Below the URL bar, the Seaborn logo and version `0.11.0` are displayed, along with navigation links for `Gallery`, `Tutorial`, `API`, `Site`, and `Page`. A search bar is also present. The main heading reads `seaborn: statistical data visualization`. Below this, a grid of six sample plots is shown, including a joint plot, a density plot, a scatter plot, a faceted line plot, a box plot, and a regression plot. The page content includes a description of Seaborn as a Python data visualization library based on `matplotlib`, providing a high-level interface for drawing attractive and informative statistical graphics. It also includes links to `introductory notes`, `installation page`, `example gallery`, `tutorial`, and `API reference`. Two sidebars are visible: `Contents` with links to `Introduction`, `Release notes`, `Installing`, `Example gallery`, `Tutorial`, and `API reference`; and `Features` with links to `Relational`, `Distribution`, `Categorical`, `Regression`, `Multiples`, `Style`, and `Color`, each followed by `API | Tutorial`.

seaborn.pydata.org/index.html

seaborn 0.11.0 Gallery Tutorial API Site Page Search

seaborn: statistical data visualization



Seaborn is a Python data visualization library based on `matplotlib`. It provides a high-level interface for drawing attractive and informative statistical graphics.

For a brief introduction to the ideas behind the library, you can read the [introductory notes](#). Visit the [installation page](#) to see how you can download the package and get started with it. You can browse the [example gallery](#) to see what you can do with seaborn, and then check out the [tutorial](#) and [API reference](#) to find out how.

To see the code or report a bug, please visit the [GitHub repository](#). General support questions are most at home on [stackoverflow](#) or [discourse](#), which have dedicated channels for seaborn.

Contents

- [Introduction](#)
- [Release notes](#)
- [Installing](#)
- [Example gallery](#)
- [Tutorial](#)
- [API reference](#)

Features

- Relational: [API](#) | [Tutorial](#)
- Distribution: [API](#) | [Tutorial](#)
- Categorical: [API](#) | [Tutorial](#)
- Regression: [API](#) | [Tutorial](#)
- Multiples: [API](#) | [Tutorial](#)
- Style: [API](#) | [Tutorial](#)
- Color: [API](#) | [Tutorial](#)

카테고리의 실수화

```
[3] 1 from sklearn.preprocessing import LabelEncoder # LabelEncoder() method를 불러옴
    2 import numpy as np # numpy를 불러옴
    3 classle=LabelEncoder()
    4 y=classle.fit_transform(iris['species'].values) # species 열의 문자열은 categorical 값으로 전환
    5 print('species labels:', np.unique(y)) # 중복되는 y 값을 하나로 정리하여 print
```

```
➤ species labels: [0 1 2]
```

```
▶ 1 yo=classle.inverse_transform(y) # 원래의 species 문자열로 전환
   2 print('species:', np.unique(yo))
```

```
➤ species: ['setosa' 'versicolor' 'virginica']
```

- [주의] DictVectorizer 클래스 vs LabelEncoder 클래스
 - One-hot encoding vs 범주형 라벨

데이터 분할

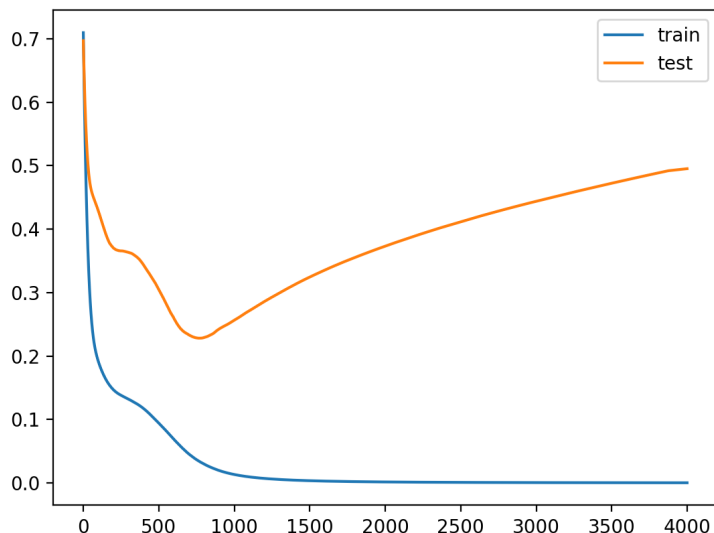


- 데이터 분할이란?

- 학습 데이터(train)와 시험 데이터(test)가 서로 겹치지 않도록 나누는 것

- 데이터 분할의 목적

- 학습데이터로 자료를 학습시키고 학습에 전혀 사용하지 않은 시험데이터에 적용하여 학습 결과의 일반화(generalization)가 가능한지 알아보기 위함



모델이 과적합되었다면, validation 셋으로 검증 시 **예측율이나 오차율이 떨어지는 현상**을 확인할 수 있으며, 이런 현상이 나타나면 **학습을 종료**

데이터 분할

```
1
2 from sklearn.model_selection import train_test_split #Scikit-Learn 의 model_selection library를 train_test_split로 명명
3 X_train,X_test,y_train,y_test=train_test_split(X,y, test_size=0.3, random_state=1, stratify=y) # x와 y의 data를 각각 30%, 70%의 비율
4 print(X_train.shape)
5 print(X_test.shape)
6 print(y_train.shape)
7 print(y_test.shape)
```

```
↳ (105, 4)
   (45, 4)
   (105,)
   (45,)
```

▪ train_test_split() 함수의 인자 설명

옵션 값 설명

- **test_size**: 테스트 셋 구성의 비율을 나타냅니다. train_size의 옵션과 반대 관계에 있는 옵션 값이며, 주로 test_size를 지정해 줍니다. 0.2는 전체 데이터 셋의 20%를 test (validation) 셋으로 지정하겠다는 의미입니다. **default 값은 0.25** 입니다.
- **shuffle**: **default=True** 입니다. split을 해주기 이전에 섞을건지 여부입니다. 보통은 default 값으로 놔둡니다.
- **stratify**: **default=None** 입니다. classification을 다룰 때 매우 중요한 옵션값입니다. stratify 값을 target으로 지정해주면 각각의 **class 비율(ratio)**을 **train / validation에 유지**해 줍니다. (한 쪽에 **쏠려서 분배되는 것을 방지**합니다) 만약 이 옵션을 지정해 주지 않고 classification 문제를 다룬다면, 성능의 차이가 많이 날 수 있습니다.
- **random_state**: 세트를 섞을 때 해당 int 값을 보고 섞으며, 하이퍼 파라미터를 튜닝시 이 값을 고정해두고 튜닝해야 매번 데이터셋이 변경되는 것을 방지할 수 있습니다.

모형 추정 및 사례중심 학습



```
1 # KNN 의 적용
2 from sklearn.neighbors import KNeighborsClassifier #KNN 불러오기
3 knn=KNeighborsClassifier(n_neighbors=5,p=2) #5개의 인접한이웃, 거리측정기준:유클리드
4 #knn.fit(X_train_std,y_train) #모델 fitting과정
5 knn.fit(X_train,y_train) #모델 fitting과정
```

```
[> KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                        metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                        weights='uniform')
```

```
[17] 1 #y_train_pred=knn.predict(X_train_std) #train data의 y값 예측치
      2 y_train_pred=knn.predict(X_train) #train data의 y값 예측치
      3 y_test_pred=knn.predict(X_test_std) #모델을 적용한 test data의 y값 예측치
      4 y_test_pred=knn.predict(X_test) #모델을 적용한 test data의 y값 예측치
      5 print('Misclassified training samples: %d' %(y_train!=y_train_pred).sum()) #오분류 데이터 갯수 확인
      6 print('Misclassified test samples: %d' %(y_test!=y_test_pred).sum()) #오분류 데이터 갯수 확인
```

```
[> Misclassified training samples: 2
     Misclassified test samples: 1
```

```
[18] 1 from sklearn.metrics import accuracy_score #정확도 계산을 위한 모듈 import
      2 print(accuracy_score(y_test,y_test_pred)) # 45개 test sample중 42개가 정확하게 분류됨.
```

```
[> 0.9777777777777777
```

결과 분석

- 성능 평가
 - 분류 문제는 회귀 분석과 달리 다양한 성능 평가 기준(metric)이 필요함
 - 평가 방법마다 장단점이 존재함
- 싸이킷런에서 제공하는 분류 성능 평가 방법
 - `confusion_matrix(y_true, y_pred)`
 - `accuracy_score(y_true, y_pred)`
 - `precision_score(y_true, y_pred)`
 - `recall_score(y_true, y_pred)`
 - `fbeta_score(y_true, y_pred, beta)`
 - `f1_score(y_true, y_pred)`
 - `roc_curve`
 - `auc`

결과 분석

	예측 클래스 0	예측 클래스 1	예측 클래스 2
정답 클래스 0	정답 클래스가 0, 예측 클래스가 0인 표본의 수	정답 클래스가 0, 예측 클래스가 1인 표본의 수	정답 클래스가 0, 예측 클래스가 2인 표본의 수
정답 클래스 1	정답 클래스가 1, 예측 클래스가 0인 표본의 수	정답 클래스가 1, 예측 클래스가 1인 표본의 수	정답 클래스가 1, 예측 클래스가 2인 표본의 수
정답 클래스 2	정답 클래스가 2, 예측 클래스가 0인 표본의 수	정답 클래스가 2, 예측 클래스가 1인 표본의 수	정답 클래스가 2, 예측 클래스가 2인 표본의 수

- **혼합 행렬 (confusion matrix):** 타겟의 원래 클래스와 모형이 예측한 클래스가 일치하는지는 갯수로 센 결과를 표나 나타낸 것

```
1 from sklearn.metrics import confusion_matrix# 오분류표 작성을 위한 모듈 import
2 conf=confusion_matrix(y_true=y_test,y_pred=y_test_pred) # 대각원소가 각각 setosa, versicolor, virginica를 정확하게 분류한 갯수.
3 print(conf)
4 # setosa는 모두 정확하게 분류되었고 versicolor는 15개 중 2개가 virginica로 오분류 되었으며 virginica는 15개 중 1개가 versicolor로 오분류됨.
```

```
[[15  0  0]
 [ 0 13  2]
 [ 0  1 14]]
```

결과 분석

암(cancer, 악성종양)을 검진할 때도 암에 걸린 것을 양성(陽性, positive)이라하고 걸리지 않은 것을 음성이라고 한다. 종양(tumor)의 양성(良性, benign), 악성(惡性, malignant) 용어와 다르다는 점에 주의하라.

- True Positive: 암을 암이라고 정확하게 예측
- True Negative: 암이 아닌것을 암이 아니라고 정확하게 예측
- False Positive: 암을 암이 아니라고 잘못 예측
- False Negative: 암이 아닌것을 암이라고 잘못 예측

	암이라고 예측	암이 아니라고 예측
실제로 암	True Positive	False Negative
실제로 암이 아님	False Positive	True Negative

- 정확도 (accuracy): 전체 샘플 중 맞게 예측한 샘플 수의 비율

$$\text{accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

결과 분석

암(cancer, 악성종양)을 검진할 때도 암에 걸린 것을 양성(陽性, positive)이라하고 걸리지 않은 것을 음성이라고 한다. 종양(tumor)의 양성(良性, benign), 악성(惡性, malignant) 용어와 다르다는 점에 주의하라.

- True Positive: 암을 암이라고 정확하게 예측
- True Negative: 암이 아닌것을 암이 아니라고 정확하게 예측
- False Positive: 암을 암이 아니라고 잘못 예측
- False Negative: 암이 아닌것을 암이라고 잘못 예측

	암이라고 예측	암이 아니라고 예측
실제로 암	True Positive	False Negative
실제로 암이 아님	False Positive	True Negative

- **정밀도 (precision):** 양성 클래스에 속한다고 예측한 샘플 중 실제로 양성 클래스에 속하는 샘플 수의 비율

$$\text{precision} = \frac{TP}{TP + FP}$$

결과 분석

암(cancer, 악성종양)을 검진할 때도 암에 걸린 것을 양성(陽性, positive)이라하고 걸리지 않은 것을 음성이라고 한다. 종양(tumor)의 양성(良性, benign), 악성(惡性, malignant) 용어와 다르다는 점에 주의하라.

- True Positive: 암을 암이라고 정확하게 예측
- True Negative: 암이 아닌것을 암이 아니라고 정확하게 예측
- False Positive: 암을 암이 아니라고 잘못 예측
- False Negative: 암이 아닌것을 암이라고 잘못 예측

	암이라고 예측	암이 아니라고 예측
실제로 암	True Positive	False Negative
실제로 암이 아님	False Positive	True Negative

- **재현율 (recall):** 실제 양성 클래스에 속한 표본 중에 양성 클래스에 속한다고 예측한 표본의 수의 비율

$$\text{recall} = \frac{TP}{TP + FN}$$

(옵션) 입력데이터의 표준화

■ 표준화

- 특성 자료의 측정 단위(Scaling)에 의해 영향 받지 않도록 하는 과정
- 사이킷런의 **StandardScaler 클래스**를 호출하여 사용
- 시험 데이터(test data)의 표준화는 학습 데이터(train data)에서 구한 특성 변수의 평균과 표준편차를 이용함
- 표준화로 인해 데이터의 분포인 통계적 특성이 깨지면 머신러닝의 학습 저하를 가져옴

```
[7] 1 from sklearn.preprocessing import StandardScaler #Scikit-Learn 의 model_selection library를 train_test_split로 명명
2 sc=StandardScaler()
3 sc.fit(X_train)
4 X_train_std=sc.transform(X_train) # training data의 표준화
5 X_test_std=sc.transform(X_test) # test data의 표준화
6
7 #표준화된 data의 확인
8 print(X_train.head()) # X_train data 최초 5개의 관측치
9 X_train_std[1:5,] # X_train_std data 최초 5개의 관측치
```

```
sepal_length sepal_width petal_length petal_width
33          5.5         4.2          1.4          0.2
20          5.4         3.4          1.7          0.2
115         6.4         3.2          5.3          2.3
124         6.7         3.3          5.7          2.1
35          5.0         3.2          1.2          0.2
array([[ -0.55053619,  0.76918392, -1.16537974, -1.30728421],
       [ 0.65376173,  0.30368356,  0.84243039,  1.44587881],
       [ 1.0150511 ,  0.53643374,  1.0655204 ,  1.18367281],
       [-1.03225536,  0.30368356, -1.44424226, -1.30728421]])
```

(옵션) 입력데이터의 표준화

```
1 # KNN 의 적용
2 from sklearn.neighbors import KNeighborsClassifier #KNN 불러오기
3 knn=KNeighborsClassifier(n_neighbors=5,p=2) #5개의 인접한이웃, 거리측정기준:유클리드
4 knn.fit(X_train_std,y_train) #모델 fitting과정
5 #knn.fit(X_train,y_train) #모델 fitting과정
```

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

```
[11] 1 y_train_pred=knn.predict(X_train_std) #train data의 y값 예측치
      2 #y_train_pred=knn.predict(X_train) #train data의 y값 예측치
      3 y_test_pred=knn.predict(X_test_std) #모델을 적용한 test data의 y값 예측치
      4 #y_test_pred=knn.predict(X_test) #모델을 적용한 test data의 y값 예측치
      5 print('Misclassified training samples: %d' %(y_train!=y_train_pred).sum()) #오분류 데이터 갯수 확인
      6 print('Misclassified test samples: %d' %(y_test!=y_test_pred).sum()) #오분류 데이터 갯수 확인
```

```
➤ Misclassified training samples: 4
   Misclassified test samples: 3
```

```
[12] 1 from sklearn.metrics import accuracy_score #정확도 계산을 위한 모듈 import
      2 print(accuracy_score(y_test,y_test_pred)) # 45개 test sample중 42개가 정확하게 분류됨.
```

```
➤ 0.9333333333333333
```

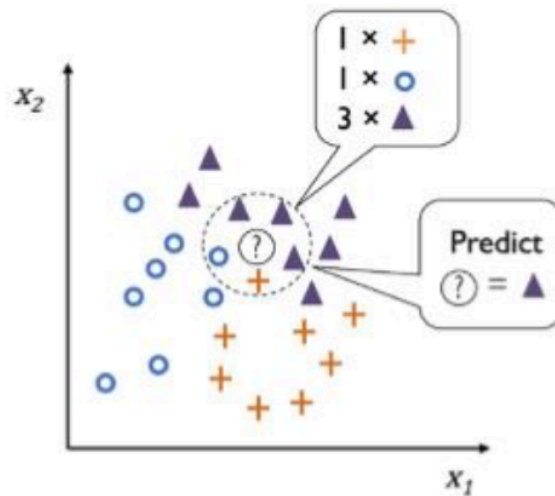
- 표준화로 인해 정확도가 97.8 ➔ 93.3 으로 떨어진 사례
- 표준화 여부는 시험 데이터(test data)의 정밀도(accuracy)를 점검 하여 결정함

KNN의 적용

Regression

KNN 회귀 정의

- KNN 회귀(regression)도 KNN 분류(classification)과 동일
 - y 의 예측치 계산만 다름
- K개 관측치 (x_i, y_i) 에서 \bar{y} 를 계산하여 적합치로 사용
- 주어진 특성 변수 x 에 대응하는 y 의 예측치
 - $\frac{1}{k} \sum_{i=1}^k y_i$ 단, y_i 는 x 에 가장 가까운 K개의 학습 데이터 y



KNN 회귀

- 단순 회귀 란?
 - 가까운 이웃들의 단순한 평균을 구하는 방식
- 가중 회귀(Weighted regression) 란?
 - 각 이웃이 얼마나 가까이 있는지에 따라 가중 평균(weighted average)을 구해 거리가 가까울수록 데이터가 더 유사할 것이라고 보고 가중치를 부여하는 방식

- 예시)

영화 X의 등급을 찾기 위해 3-NN 검색 결과		
영화 A	등급: 5.0	X까지의 거리: 3.2
영화 B	등급: 6.8	X까지의 거리: 11.5
영화 C	등급: 9.0	X까지의 거리: 1.1

- 단순 평균: 6.93, 가중 평균: 7.9 →
$$\frac{\frac{5.0}{3.2} + \frac{6.8}{11.5} + \frac{9.0}{1.1}}{\frac{1}{3.2} + \frac{1}{11.5} + \frac{1}{1.1}} = 7.9$$

KNN 회귀 실습

- KNN 회귀를 이용한 영화 평점 예측
 - “평이 좋다” vs “평이 나쁘다” 레이블로 분류하는 게 아니라 실제 IMDb* 등급(별점)을 예측하는 것.

*IMDb 란? 인터넷영화데이터베이스

```
1
2 from sklearn.neighbors import KNeighborsRegressor
3
4 regressor = KNeighborsRegressor(n_neighbors = 3, weights = "distance")
5
6 training_points = [
7     [0.5, 0.2, 0.1],
8     [0.9, 0.7, 0.3],
9     [0.4, 0.5, 0.7]
10 ]
11
12 training_labels = [5.0, 6.8, 9.0]
13 regressor.fit(training_points, training_labels)
14
15
16 unknown_points = [
17     [0.2, 0.1, 0.7],
18     [0.4, 0.7, 0.6],
19     [0.5, 0.8, 0.1]
20 ]
21
22 guesses = regressor.predict(unknown_points)
23 guesses
```

“distance” → 가중평균,
default = “uniform”

```
↳ array([7.28143288, 7.76451922, 6.8457845 ])
```