

# **Machine Learning & Scikit-Learn**

실습

```
import seaborn as sns
import pandas as pd
import numpy as np
%matplotlib inline
```

```
sns.get_dataset_names() # seaborn이 지원하는 데이터 세트
```

```
data = sns.load_dataset(name = "iris")
```

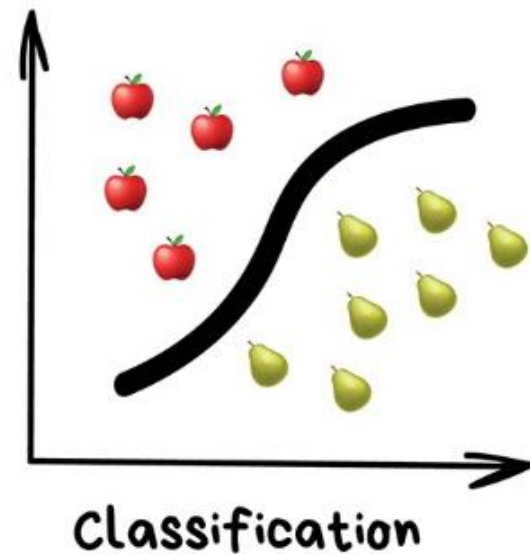
```
data.info()
```

```
data.head()
```

	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

## Day3. 분류

- 분류(Classification)
  - 둘 이상의 이산적인 범주로 레이블을 예측하는 분야
  - 오늘날의 활용 분야
    - ✓ 스팸 필터링
    - ✓ 언어 분류
    - ✓ 문서 유사도 분석
    - ✓ 필기체 문자 인식
    - ✓ 사기 판단(fraud detection)
  - 방법론
    - ✓ Naive Bayes, Decision Tree, Logistic Regression, K-Nearest Neighbors, SVM(Support Vector Machine)



# #1

## 로지스틱회귀분석

## 회귀 모델

## 목적

- 레이블된 학습 데이터를 가지고 특성(feature)과 레이블의 관계를 함수식으로 표현하는 것

## 결과값

- 데이터 세트 범위 내의 값( 어떤 값이 나올지 예상하지 못하기 때문에 예측모델이라함)

## 종류

- 선형 회귀
- 로지스틱 회귀: 범주형 결과값

## 분류 모델

- 그룹명(레이블)이 적힌 학습 데이터로 학습→새로 입력된 데이터가 속한 그룹을 찾아내는 것

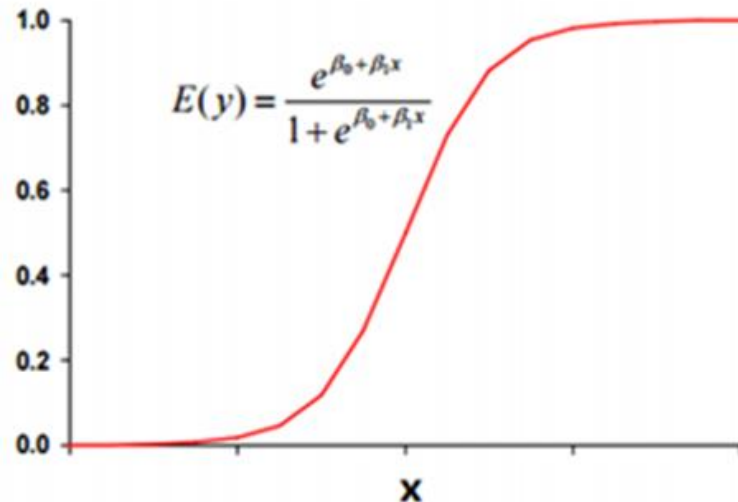
- 학습 데이터 레이블 중 하나

- kNN
- SVM

## 2) 로지스틱 회귀 분석(logistic regression)

- 적용 분야:
  - ✓ 종속변수가 예/아니오, 1/0, 합격/불합격, 구매/비구매 같은 범주형(categorical)으로 표현되는 경우
- 분류 모델임
- 회귀식 형태 : 로지스틱(logistic,=시그모이드 ,sigmoid) 함수

$$p(x)_{\beta_0, \beta_1} = \frac{e^{\beta_0 + \beta_1 x}}{1 + e^{\beta_0 + \beta_1 x}}$$

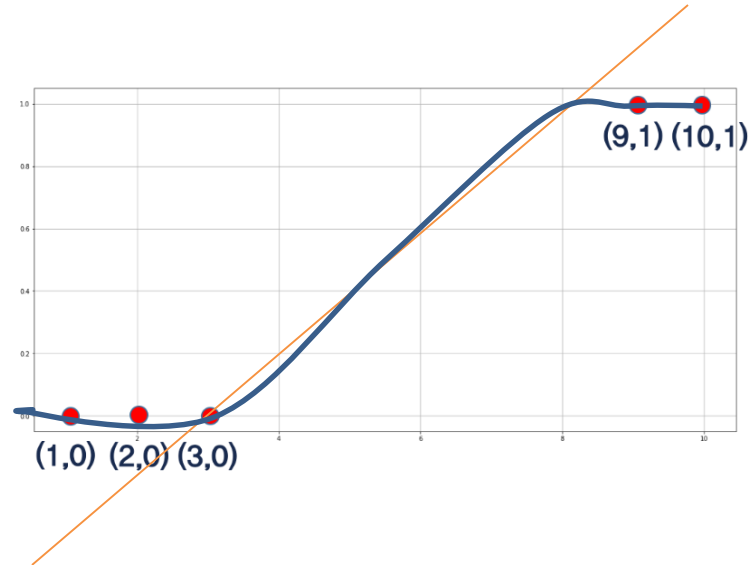




## 2) 로지스틱 회귀 분석(logistic regression)

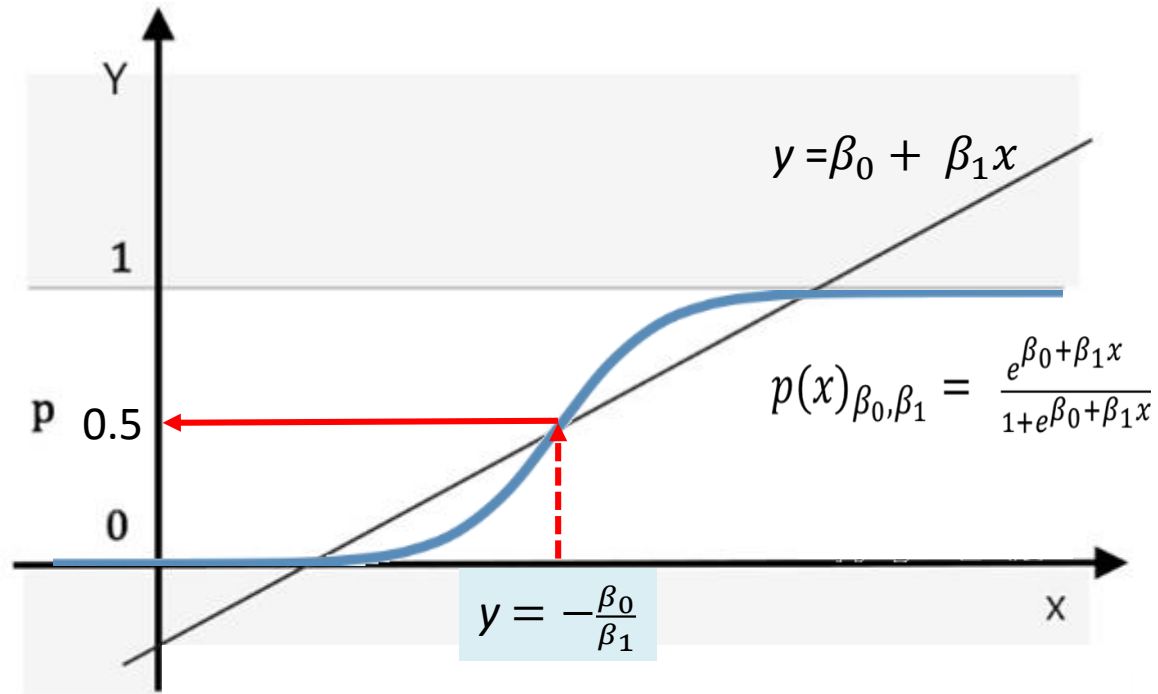
적용 분야 사례

X 데이터	Y 데이터
10	Pass (1)
9	Pass (1)
3	Fail (0)
2	Fail (0)
1	Fail (0)
5	???



## 2) 로지스틱 회귀 분석(logistic regression)

## ✓ 로지스틱 함수의 특성



## 2) 로지스틱 회귀 분석(logistic regression)

- 에러 함수
- 에러 함수의 최소값을 구하는 방법
  - ✓ 최대가능도법(maximum likelihood method)
  - ✓ Coordinate distance algorithm
  - ✓ Stochastic average gradient descent algorithm
  - ✓ Newton method
  - ✓ BFGS(Broyden-Fletcher-Goldfarb-Shanno)
  - ✓ LBFGS(Limited memory BFGS)

## 2) 로지스틱 회귀 분석(logistic regression)

- Coordinate distance algorithm
  - ✓ Scikit-learn에서 사용하는 방법
  - ✓ 편미분 사용하지 않음

## 1. 데이터 준비

```
import pandas as pd
from sklearn.datasets import load_iris
import seaborn as sns; sns.set()
%matplotlib inline
```

```
iris = load_iris()
```

```
df = pd.DataFrame(iris.data, columns=iris.feature_names)
```

```
df['species'] = pd.Series(iris.target)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
sepal length (cm)    150 non-null float64
sepal width (cm)     150 non-null float64
petal length (cm)    150 non-null float64
petal width (cm)     150 non-null float64
species              150 non-null int32
dtypes: float64(4), int32(1)
memory usage: 5.4 KB
```

## 2. 학습 시킬 데이터를 가지는 새로운 data frame 만들기

```
sl_df = pd.DataFrame()
```

```
sl_df['sepal_length'] = df['sepal length (cm)']
sl_df['species'] = df['species']
sl_df.info()
```

```
sl_df = sl_df[:100]
sl_df.info()
```

2 클래스만

```
sl_df.describe()
```

ame'&gt;

	sepal_length	species
count	100.000000	100.000000
mean	5.471000	0.500000
std	0.641698	0.502519
min	4.300000	0.000000
25%	5.000000	0.000000
50%	5.400000	0.500000
75%	5.900000	1.000000
max	7.000000	1.000000

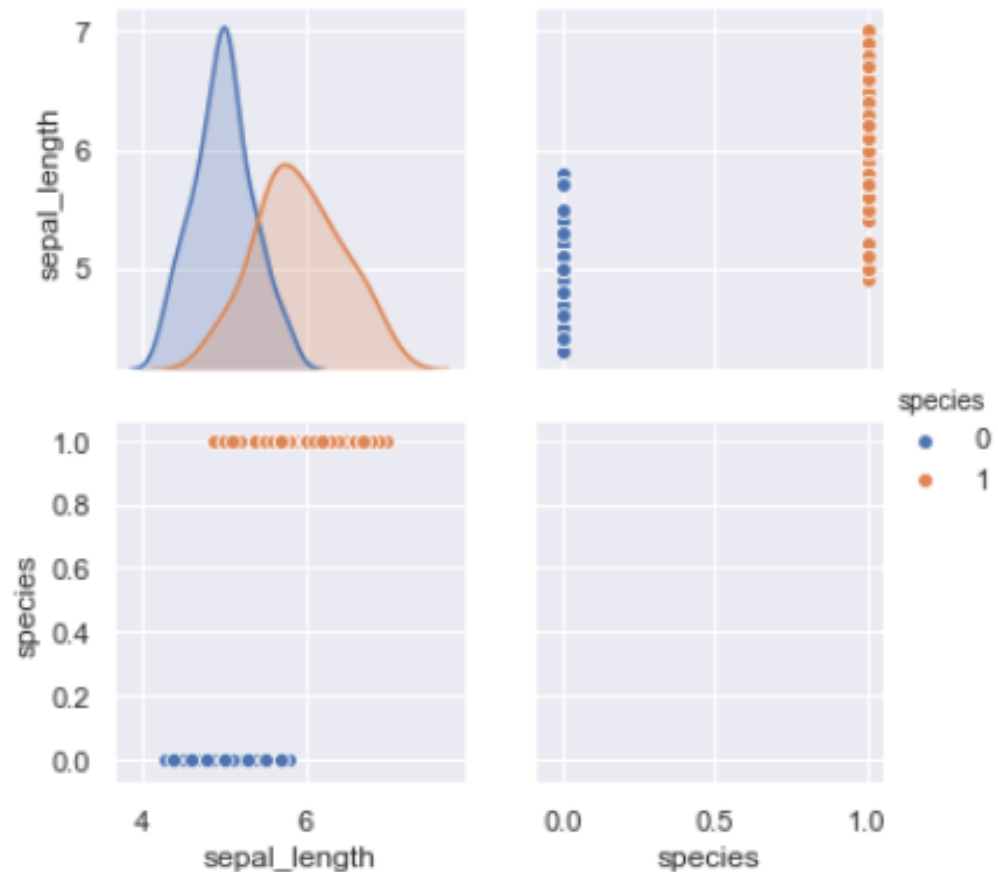
at 64  
32

## 2. 학습 시킬 데이터를 가지는 새로운 data frame 만들기

\* 데이터 확인

```
sns.pairplot(sl_df, hue='species')
```

<seaborn.axisgrid.PairGrid at 0x9ea84b11d0>



### 3. 학습 데이터와 테스트 데이터 나누기

```
# 학습 데이터, 테스트 데이터 나누기
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split( sl_df.iloc[:, :1],  
                                                    sl_df.iloc[:, 1:], test_size=0.33)
```



## 4. 모듈 import

```
from sklearn.linear_model import LogisticRegression
```

## 5. 모델을 인스턴스화

```
lr = LogisticRegression()
```

## 6. 데이터를 특징과 대상 벡터로 배치

```
X = x[:, np.newaxis]
```

```
x.shape
```

```
(50,)
```

```
X.shape
```

```
(50, 1)
```

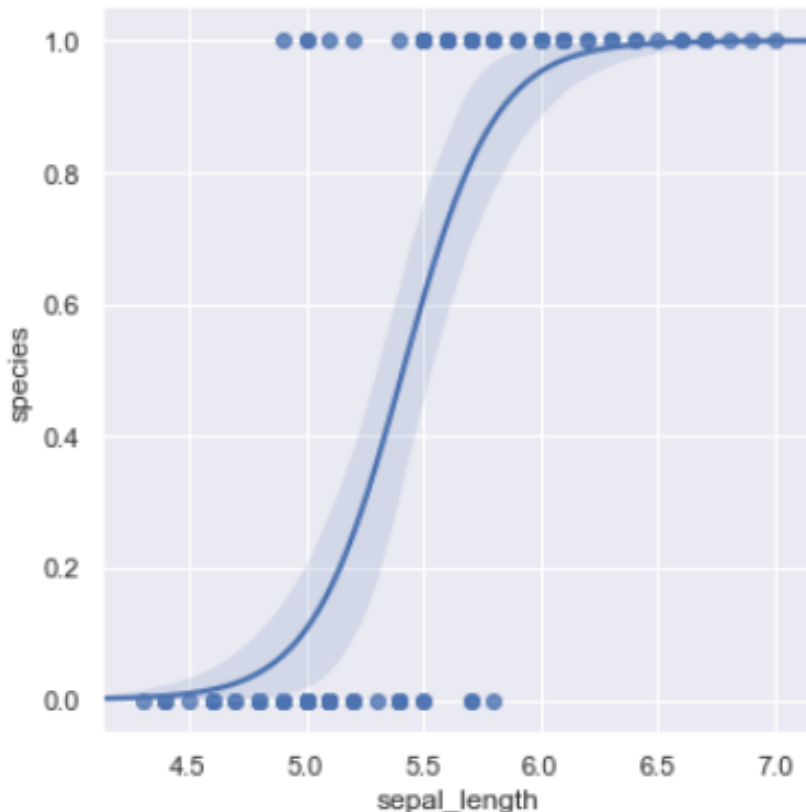
## 6. 주어진 데이터로 모델을 학습시키기

```
lr.fit(X_train, y_train)
```

## 7. 결과 확인: 데이터로 regression 모델의 그래프 그려보기

```
import seaborn as sns; sns.set()  
sns.lmplot(x='sepal_length', y='species', data=s1_df, logistic=True)
```

<seaborn.axisgrid.FacetGrid at 0x9ea97429e8>



## 8. 학습된 데이터에 대한 성능 평가 하기

```
from sklearn.metrics import classification_report, confusion_matrix
```

```
print(confusion_matrix(y_train, lr.predict(X_train)))
```

```
[[27  8]
 [ 4 28]]
```

```
print(classification_report(y_train, lr.predict(X_train)))
```

	precision	recall	f1-score	support
0	0.87	0.77	0.82	35
1	0.78	0.88	0.82	32
accuracy			0.82	67
macro avg	0.82	0.82	0.82	67
weighted avg	0.83	0.82	0.82	67

## 9. 테스트 데이터에 대한 성능 평가 하기

```
print(confusion_matrix(y_test, lr.predict(X_test)))
```

```
[[13  2]
 [ 1 17]]
```

```
print(classification_report(y_test, lr.predict(X_test)))
```

	precision	recall	f1-score	support
0	0.93	0.87	0.90	15
1	0.89	0.94	0.92	18
accuracy			0.91	33
macro avg	0.91	0.91	0.91	33
weighted avg	0.91	0.91	0.91	33

**#2**

## K-Nearest Neighbor

## ■ kNN(k-Nearrest Neighbor) 모델

- 머신러닝 모델 중 가장 직관적이고 간단한 지도학습 모델
- 유사성 척도(즉, 거리함수)기반 분류 방법

✓ Euclidean distance 
$$\sqrt{\sum_{i=1}^k (x_i - y_i)^2}$$

### 알고리즘

- 1) 학습데이터가 주어짐 : 데이터를 클래스별로 저장해 놓음
- 2) 분류할 새로운 데이터가 들어옴

- ① 입력 데이터와 가장 가까운 k개의 (학습)데이터를 찾음
- ② 찾은 k 개의 점이 속한 그룹 중에서

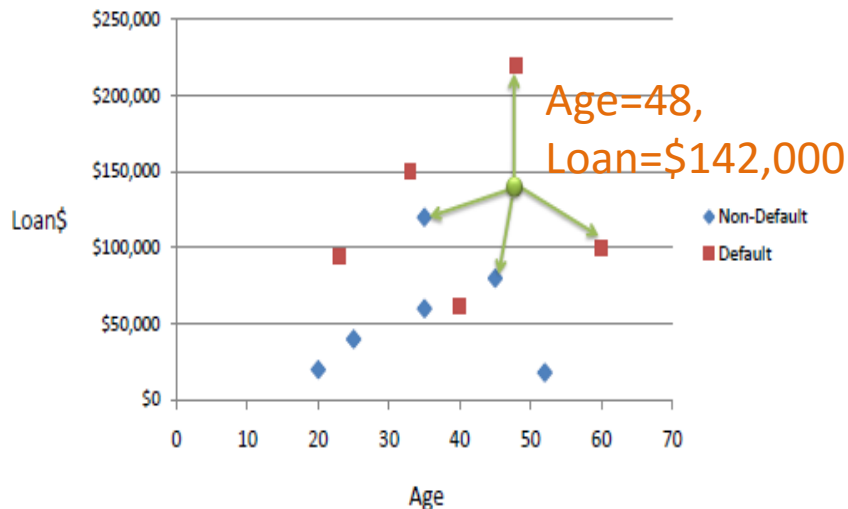
데이터가 가장 많은 그룹을 입력데이터의 그룹으로 정함  
(k는 항상 홀수, 3~10 사이의 값이 최적)

### 데이터 정규화 변환

- ✓ Minmax scaling



- (예) 나이(age)와 대출금(loan)으로 이 사람의 채무불이행(default)여부 판단



### (1) Euclidean distance 계산

$$D = \text{Sqrt}[(48-33)^2 + (142000-150000)^2] = 8000.01 \gg \text{Default}=Y$$

Age	Loan	Default	Distance	
25	\$40,000	N	102000	
35	\$60,000	N	82000	
45	\$80,000	N	62000	
20	\$20,000	N	122000	
35	\$120,000	N	22000	2
52	\$18,000	N	124000	
23	\$95,000	Y	47000	
40	\$62,000	Y	80000	
60	\$100,000	Y	42000	3
48	\$220,000	Y	78000	
33	\$150,000	Y	8000	1
48	\$142,000	?		

Euclidean Distance

$$D = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$$

### (2) K 결정

k=1 이면 Default = Y

k=3 이면 Default = Y (2 Y & 1 N)

- (예) 나이(age)와 대출금(loan)으로 이 사람의 채무불이행(default)여부 판단

✓ 데이터 정규화 변환(scaling)의 필요성

(1) Euclidean distance 계산

- Distance는 Loan의 영향을 받음 (Age < 100 이고 40000 < Loan 임)

Age	Loan	Default	Distance
0.125	0.11	N	0.7652
0.375	0.21	N	0.5200
0.625	0.31	N	0.3160
0	0.01	N	0.9245
0.375	0.50	N	0.3428
0.8	0.00	N	0.6220
0.075	0.38	Y	0.6669
0.5	0.22	Y	0.4437
1	0.41	Y	0.3650
0.7	1.00	Y	0.3861
0.325	0.65	Y	0.3771
0.7	0.61	?	

K=1일 때, 결과 Default= N

K=3일 때, 결과 Default=N

Standardized Variable

$$X_s = \frac{X - \text{Min}}{\text{Max} - \text{Min}}$$

Minmax scaling

## ■ IRIS 데이터에 적용하기

### (1) 자료 준비하기

```
import pandas as pd
import numpy as np
from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
%matplotlib inline
```

```
iris = load_iris()
```

```
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['species'] = pd.Series(iris.target)
```

## ■ IRIS

```
def setcolor(value):  
    color = []  
    colors = ['r', 'g', 'b']  
    for i in value.values:  
        color.append(colors[i])  
    return color
```

```
plt.scatter(x=df['petal length (cm)'], y=df['petal width (cm)'],  
           color = setcolor(df['species']))
```

<matplotlib.collections.PathCollection at 0x9eaaa63dd8>



- IRIS 데이터에 적용하기

(2) 모델 import 하기

```
from sklearn.neighbors import KNeighborsClassifier
```

- IRIS 데이터에 적용하기

(3) kNN = 3으로 학습하기

```
from sklearn.neighbors import KNeighborsClassifier
```

```
column_train = ['petal length (cm)', 'petal width (cm)']  
neigh_3 = KNeighborsClassifier(n_neighbors = 3, weights = 'distance')
```

```
neigh_3_train = neigh_3.fit(df[column_train], df['species'])
```

## ■ IRIS 데이터에 적용하기

(4) kNN = 3으로 학습한 결과를 새로운 점에 적용해 예측하기

```
new_data = np.array([2.5, 0.8]).reshape(1, -1)
```

```
# 학습 자료 출력
```

```
plt.scatter(x=df['petal length (cm)'], y=df['petal width (cm)'],  
            color=setcolor(df['species']))
```

```
# 새 자료의 위치
```

```
plt.scatter(x=new_data[0,0], y=new_data[0,1], color='orange')
```

<matplotlib.collections.PathCollection at 0x9eaaae8358>



- IRIS 데이터에 적용하기

(4) kNN = 3으로 학습한 결과를 새로운 점에 적용해 예측하기

```
neigh_3_class = neigh_3_train.predict(new_data)
```

```
print(neigh_3_class)
```

```
[1]
```

1번 클래스,  
'g'color



## ■ IRIS 데이터 (4) kNN

```
# 학습 자료 출력
plt.scatter(x=df['petal length (cm)'], y=df['petal width (cm)'],
            color=setcolor(df['species']))

# 새 자료의 위치
c3 = pd.DataFrame(np.array(neigh_3_class), columns=['c'])
col3 = c3['c']
plt.scatter(x=new_data[0,0], y=new_data[0,1],
            color=setcolor(col3))
```

<matplotlib.collections.PathCollection at 0x9eaafc55f8>



- IRIS 데이터에 적용하기

(4) kNN = 7으로 kNN 알고리즘 학습하기

```
column_train = ['petal length (cm)', 'petal width (cm)']  
neigh_7 = KNeighborsClassifier(n_neighbors = 7, weights = 'distance')
```

```
neigh_7_train = neigh_7.fit(df[column_train], df['species'])
```

- IRIS 데이터에 적용하기

(5) kNN = 7으로 학습한 결과를 새로운 점에 적용해 예측하기

```
neigh_7_class = neigh_7_train.predict(new_data)
```

```
print(neigh_7_class)
```

```
[0]
```

## ■ IRIS 데

## (5) kNN

# 학습 자료 출력

```
plt.scatter(x=df['petal length (cm)'], y=df['petal width (cm)'],
            color=setcolor(df['species']))
```

# 새 자료의 위치

```
c7 = pd.DataFrame(np.array(neigh_7_class), columns=['c'])
col7 = c7['c']
plt.scatter(x=new_data[0,0], y=new_data[0,1],
            color=setcolor(col7))
```

&lt;matplotlib.collections.PathCollection at 0x9eaab1c748&gt;



- IRIS 데이터에 적용: n\_neighbor 모델의 accuracy 평가하기
  - Data를 train과 test로 나누어서 정확도 확인하기(1)

```
# n_neighbor 모델의 accuracy
```

```
from sklearn.model_selection import train_test_split
```

```
from sklearn.metrics import classification_report, confusion_matrix
```

```
X_train, X_test, y_train, y_test = train_test_split(df[column_train],  
                                                    df['species'], test_size=0.33)
```

- IRIS 데이터에 적용: n\_neighbor 모델의 accuracy 평가하기
  - Data를 train과 test로 나누어서 정확도 확인하기(2)

```
# n_neighbor=3로 학습시키기
```

```
neigh3 = KNeighborsClassifier(n_neighbors=3, weights='distance')  
neigh3.fit(X_train, y_train)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',  
                     metric_params=None, n_jobs=None, n_neighbors=3, p=2,  
                     weights='distance')
```

- IRIS 데이터에 적용: n\_neighbor 모델의 accuracy 평가하기
  - Data를 train과 test로 나누어서 정확도 확인하기(3)

```
# 학습시킨 모델의 정확도 테스트
print('----- 3개의 이웃 데이터 -----')
print(classification_report(y_test, neigh3.predict(X_test)))
```

```
----- 3개의 이웃 데이터 -----
              precision    recall  f1-score   support

    0           1.00        1.00        1.00         18
    1           0.92        0.92        0.92         12
    2           0.95        0.95        0.95         20

 accuracy                   0.96         50
 macro avg              0.96        0.96        0.96         50
 weighted avg           0.96        0.96        0.96         50
```

**#3**

# **SVM(Support Vector Machine)**



- 서포트 벡터 머신(SVM: support vector machine)
  - 특징
    - ✓ 보편적으로 사용된 분류 머신 러닝 모델
    - ✓ 데이터가 2개의 그룹으로 분류될 때 사용
  - SVM의 목표
    - ✓ 벡터 공간에서 학습 데이터가 속한 2개의 그룹을 분류하는 선형 분리자를 찾음.  
(2개의 그룹을 가장 멀리 구분할 수 있는 선형 분리자)
    - ✓ 필요시, 선형 분류가 불가능한 현재 공간을 (한 차원 높은 공간으로 변환하여)  
선형 분류가 가능하게 분포되는 공간으로 변환하여 분리함

- SVM 분류 예시

- SVM 구성 요소

- Margin 구하기

## ■ 선형 SVM

### (1) 선형 SVM 선언 및 학습

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

```
X = np.array([ [0,0], [1,1] ])
y = [0, 1]
```

```
LinearSVM = svm.LinearSVC()
LinearSVM.fit(X, y)
```

```
LinearSVC(C=1.0, class_weight=None, dual=True, fit_intercept=True,
          intercept_scaling=1, loss='squared_hinge', max_iter=1000,
          multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
          verbose=0)
```

- 선형 SVM

- (2) 선형 SVM 학습 결과 확인

```
LinearSVM.predict([[2,2]])
```

```
array([1])
```

```
print(LinearSVM.coef_[0])
```

```
print(LinearSVM.intercept_[0])
```

```
[0.58822994 0.58822994]  
-0.4705748577631975
```

$$y = 0.59 x_1 + 0.59 x_2 + 0.47$$

- 비선형 SVM : XOR 문제

- (1) XOR 데이터 준비

```
import numpy as np
from sklearn import svm
import matplotlib.pyplot as plt
```

```
X = np.array([[0,0],[0,1],[0,1],[1,1]])
y = [0, 1, 1, 0]
```

- 비선형 SVM : XOR 문제

- (2) 비선형 SVM 선언 및 학습

```
SVM_XOR = svm.SVC()  
SVM_XOR.fit(X, y)
```

```
C:\Users\cosmos\Anaconda3\lib\site-packages\sklearn\svm\base.py:193: FutureWarning: 'scale' in version 0.22 to account better for unscaled features. Set gamma: "avoid this warning.", FutureWarning)
```

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



- 비선형 SVM : XOR 문제

(3) 새로운 데이터 대한 예측

```
test_data = np.array([[0.8, 0.8], [0.2, 0.9]])  
SVM_XOR.predict(test_data)
```

```
array([0, 1])
```

# Q & A