ch2 지도학습 - knn

- Machine Learning with sklearn @ DJ,Lim
- date : 21/10

- 지도학습은 대표적인 머신러닝 방법론중의 하나이다.
- 지도학습은 입력과 출력 샘플 데이터가 있고, 주어진 입력으로부터 출력을 예측하고자 할 때 사용.
- 지도학습에는 두가지 종류 **분류(classification), 회귀(regression)**이 있다.
- knn은 사용자가 쉽게 이해할 수 있는 대표적인 지도학습 방법중에 하나로, 분류와 회귀에 다 사용된다.

학습 내용

- 01 지도학습의 종류
- 02 knn 알고리즘 시각화
- 03 knn을 이용한 유방암 데이터 실습

In [1]:

```
### 한글 폰트 설정
import matplotlib
from matplotlib import font_manager, rc
import matplotlib.pyplot as plt
import platform

path = "C:/Windows/Fonts/malgun.ttf"
if platform.system() == "Windows":
    font_name = font_manager.FontProperties(fname=path).get_name()
    rc('font', family=font_name)
elif platform.system()=="Darwin":
    rc('font', family='AppleGothic')
else:
    print("Unknown System")

matplotlib.rcParams['axes.unicode_minus'] = False

%matplotlib inline
```

01 지도학습의 종류

분류(Classification)

- 분류는 가능성 있는 여러 클래스 레이블(class label)중 **하나를 예측하는 것**이다.
- 분류는 두개의 클래스로 분류하는 **이진 분류(binary classification)**과 셋 이상의 클래스로 분류하는 **다중 분류(multiclass classification)**로 나누어진다.
- 이진 분류는 질문의 답이 예/아니오 등의 예. 이진분류의 양성(positive) 클래스, 음성(negativ) 클래스라고 한다.

회귀(Regression)

- 회귀는 연속적인 숫자, 또는 프로그래밍 용어로 말하면 **부동소수점수(수학 용어로는 실수)를 예측**하는 것. 수치형 데이터를 예측.
- 예로는 어떤 사람의 나이, 키, 몸무게 정해진 수의 값이 아닌 해당 예측 값은 수치형 데이터.

02 knn 알고리즘 시각화

- 라이브러리 설치 : !pip install mglearn

In [2]:

```
# !pip install mglearn
```

In [4]:

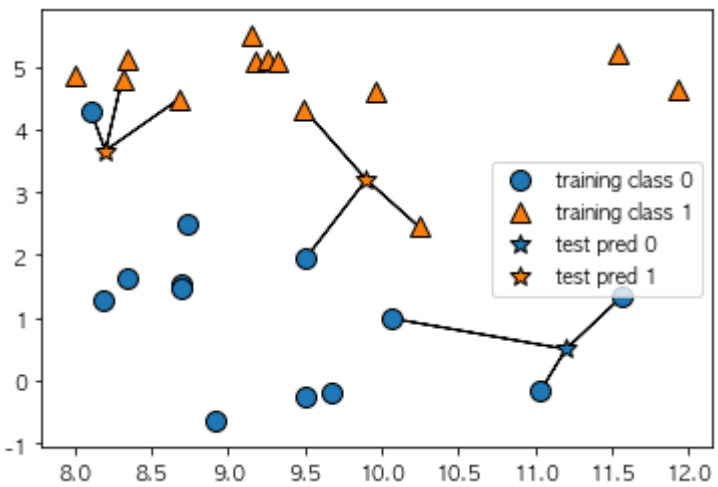
```
import mglearn
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

%matplotlib inline
```

knn의 k가 3인 경우의 알고리즘(분류- 범주형 값의 예측)

In [6]:

```
mglearn.plots.plot_knn_classification(n_neighbors=3)
```

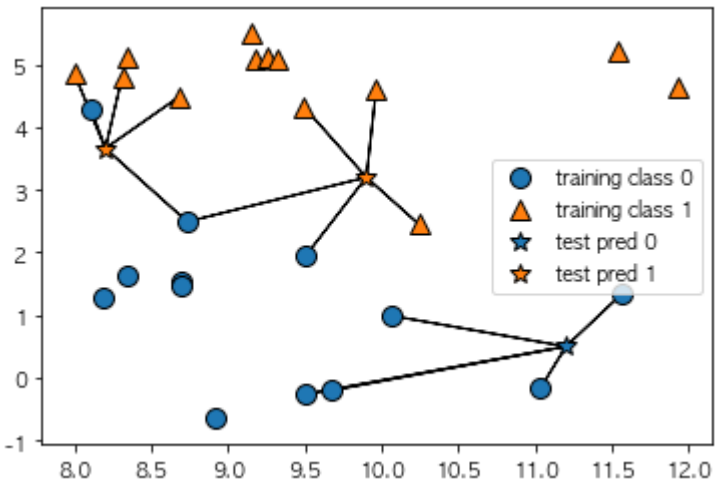
```
/Users/toto/Documents/anaconda3/lib/python3.8/site-packages/sklearn/ut
ils/deprecation.py:86: FutureWarning: Function make_blobs is deprecate
d; Please import make_blobs directly from scikit-learn
  warnings.warn(msg, category=FutureWarning)
```



In [7]:

```
mglearn.plots.plot_knn_classification(n_neighbors=5)
```

```
/Users/toto/Documents/anaconda3/lib/python3.8/site-packages/sklearn/ut
ils/deprecation.py:86: FutureWarning: Function make_blobs is deprecate
d; Please import make_blobs directly from scikit-learn
  warnings.warn(msg, category=FutureWarning)
```

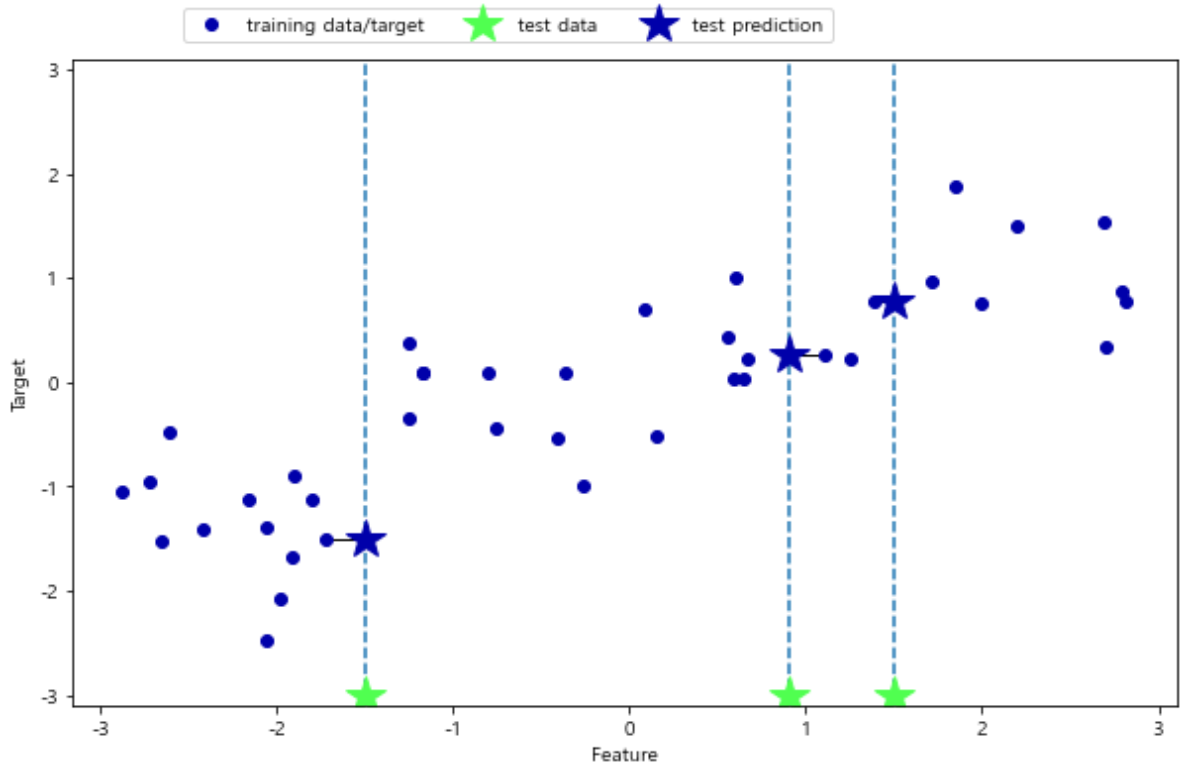


knn의 k가 1인 경우의 알고리즘(회귀-수치형 값의 예측)

- 예측하려고 하는 값이 연속형값이다. 아래 그래프는 하나의 feature를 이용하여 target를 예측하는 모델이다.
- k=1가 일 경우,
  - 새로운 값이 들어갈 경우, 하나의 feature의 값이 가장 가까운 데이터를 찾아, 해당 데이터가 가르키는 Target으로 예측하게 된다.

In [13]:

```
mglearn.plots.plot_knn_regression(n_neighbors=1)
```



mglearn.plots.plot_knn_regression(n_neighbors=3)

- k=3일 경우,
  - 새로운 값이 들어갈 경우, 하나의 feature의 값(입력으로 선택)이 가장 가까운 3개의 데이터를 찾아, 해당 3개의 데이터가 가르키는 Target 값의 평균값으로 예측하게 된다.

하이퍼 파라미터 k에 따른 결정경계

- k의 값에 따른 어디로 분류가 되는지에 대한 결정 경계를 보여준다.

In [9]:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
```

데이터 준비 및 나누기

In [10]:

```
X, y = mglearn.datasets.make_forge()

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

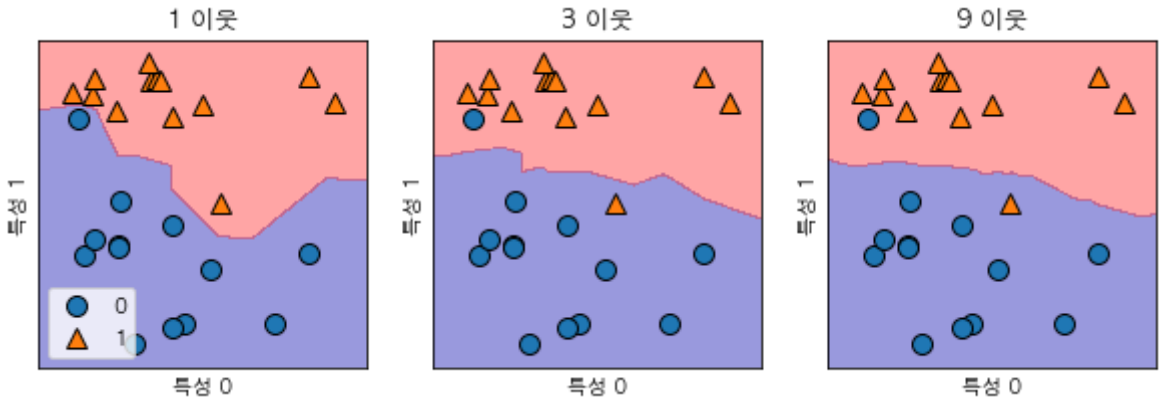
```
/Users/toto/Documents/anaconda3/lib/python3.8/site-packages/sklearn/ut
ils/deprecation.py:86: FutureWarning: Function make_blobs is deprecate
d; Please import make_blobs directly from scikit-learn
  warnings.warn(msg, category=FutureWarning)
```

In [11]:

```
fig, axes = plt.subplots(1, 3, figsize=(10, 3))

for n_neighbors, ax in zip([1, 3, 9], axes):
    # fit 메소드는 self 오브젝트를 리턴합니다
    # 그래서 객체 생성과 fit 메소드를 한 줄에 쓸 수 있습니다
    model = KNeighborsClassifier(n_neighbors=n_neighbors).fit(X, y)
    mglearn.plots.plot_2d_separator(model, X,
                                    fill=True, eps=0.5, ax=ax, alpha=.4)

    mglearn.discrete_scatter(X[:, 0], X[:, 1], y, ax=ax)
    ax.set_title("{} 이웃".format(n_neighbors))
    ax.set_xlabel("특성 0")
    ax.set_ylabel("특성 1")
axes[0].legend(loc=3)
```

Out[11]:

```
<matplotlib.legend.Legend at 0x7fed24ca9130>
```



일반화, 과대적합, 과소적합

- 모델이 처음보는 데이터에 대해 예측이 가능하다면 이를 훈련세트에서 테스트 세트로 **일반화(generalization)**되었다고 한다.
- 복잡한 모델(학습용 데이터에만 충실한)을 만든다면 훈련세트에만 정확한 모델이 된다.(**과대적합**)
  - 과대적합(overfitting)는 모델이 훈련 세트의 각 샘플에 너무 가깝게 맞춰져서 새로운 데이터가 일반화되기 어려울 때 발생.
- 반대로 모델이 너무 간단해서 잘 예측을 못함.(과소적합-**underfitting**)

03 유방암 데이터 셋 실습

- 데이터 셋 : 위스콘신 유방암(Wisconsin Breast Cancer)데이터 셋
- 각 종양은 악성(malignant-암 종양), 양성(benign-해롭지 않은 종양)

In [12]:

```
from sklearn.datasets import load_breast_cancer
import matplotlib.pyplot as plt
```

In [13]:

```
cancer = load_breast_cancer()
print("cancer.keys() : \n{}".format(cancer.keys()))
print("유방암 데이터의 행열 : {}".format(cancer.data.shape))
```

```
cancer.keys() :
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'featur
e_names', 'filename'])
유방암 데이터의 행열 : (569, 30)
```

feature 이름, class 이름

In [14]:

```
print("특성이름(featuer_names) : {}".format(cancer['feature_names']))
print("유방암 데이터의 형태 : ", cancer.data.shape)
print()
print("클래스 이름(target_names) : {}".format(cancer['target_names']))
print("클래스별 샘플 개수 : \n",np.bincount(cancer.target))
```

```
특성이름(featuer_names) : ['mean radius' 'mean texture' 'mean perimeter'
'mean area'
 'mean smoothness' 'mean compactness' 'mean concavity'
 'mean concave points' 'mean symmetry' 'mean fractal dimension'
 'radius error' 'texture error' 'perimeter error' 'area error'
 'smoothness error' 'compactness error' 'concavity error'
 'concave points error' 'symmetry error' 'fractal dimension error'
 'worst radius' 'worst texture' 'worst perimeter' 'worst area'
 'worst smoothness' 'worst compactness' 'worst concavity'
 'worst concave points' 'worst symmetry' 'worst fractal dimension']
유방암 데이터의 형태 :  (569, 30)

클래스 이름(target_names) : ['malignant' 'benign']
클래스별 샘플 개수 :
 [212 357]
```

- malignant(악성):0 이 212개
- benign(양성):1 이 357개

데이터 셋 나누기

- 비율을 맞추기 위해 stratify를 cancer.target로 지정

In [15]:

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

In [16]:

```
X = cancer.data
y = cancer.target

X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    stratify=cancer.target,
                                                    random_state=77)
```

In [21]:

```
print( len(y_train) )
y_1_all = (y == 1).sum()
y_0_all =  (y == 0).sum()

print(f"데이터 셋의 target y에서 0의개수 : {y_0_all}, 1의개수 : {y_1_all}")
```

```
426
데이터 셋의 target y에서 0의개수 : 212, 1의개수 : 357
```

In [23]:

```
y_train_1 = (y_train == 1).sum()
y_test_1 = (y_test == 1).sum()

y_train_0 = len(y_train) - (y_train == 1).sum()
y_test_0 = len(y_test) - (y_test == 1).sum()

print("데이터 셋의 target(학습:테스트)의 비율 - 1에 대해서")
print(f"train비율 : {y_train_1/y_1_all}, test비율 : {y_test_1/y_1_all}")

print("데이터 셋의 target(학습:테스트)의 비율 - 0에 대해서")
print(f"train비율 : {y_train_0/y_0_all}, test비율 : {y_test_0/y_0_all}")
```

```
데이터 셋의 target(학습:테스트)의 비율 - 1에 대해서
train비율 : 0.7478991596638656, test비율 : 0.25210084033613445
데이터 셋의 target(학습:테스트)의 비율 - 0에 대해서
train비율 : 0.75, test비율 : 0.25
```

- 정확하게 75%, 25%로 나누어져있다.

(실습)

- 모델을 만들어 보자. knn모델을 생성하고, 이를 평가해보자.

04 머신러닝 모델 만들고 예측하기

작업 단계

(1) 모델 만들기
(2) 모델 학습 시키기(fit)
(3) 모델을 이용한 값 예측(predict)
(4) 훈련 데이터를 이용한 정확도 확인
(5) 테스트 데이터를 이용한 정확도 확인

In [24]:

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

In [25]:

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

Out[25]:

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

In [26]:

```
# 정확도 구하기
(pred == y_test).sum()/(len(pred))
```

Out[26]:

```
0.9090909090909091
```

In [27]:

```
acc_tr = model.score(X_train, y_train)    # 정확도(학습용 데이터)
acc_test = model.score(X_test, y_test)    # 정확도(테스트 데이터)
acc_tr, acc_test
```

Out[27]:

```
(0.9553990610328639, 0.9090909090909091)
```

score를 이용한 결과 확인

In [28]:

```
print("k : {}".format(3))
print("훈련 데이터셋 정확도 : {:.2f}".format(acc_tr))
print("테스트 데이터 셋 정확도 : {:.2f}".format(acc_test))
```

```
k : 3
훈련 데이터셋 정확도 : 0.96
테스트 데이터 셋 정확도 : 0.91
```

05 k의 값에 따른 정확도 확인해 보기

In [29]:

```
tr_acc = []
test_acc = []
k_nums = range(1, 22, 2)# 1,3,5~21

for n in k_nums:
    # 모델 선택 및 학습
    model = KNeighborsClassifier(n_neighbors=n)
    model.fit(X_train, y_train)

    # 정확도 구하기
    acc_tr = model.score(X_train, y_train)
    acc_test = model.score(X_test, y_test)

    # 정확도 값 저장.
    tr_acc.append(acc_tr)
    test_acc.append(acc_test)

    print("k : ", n)
    print("학습용셋 정확도 {:.3f}".format(acc_tr) )
    print("테스트용셋 정확도 {:.3f}".format(acc_test) )
```

```
k :  1
학습용셋 정확도 1.000
테스트용셋 정확도 0.888
k :  3
학습용셋 정확도 0.955
테스트용셋 정확도 0.909
k :  5
학습용셋 정확도 0.953
테스트용셋 정확도 0.916
k :  7
학습용셋 정확도 0.953
테스트용셋 정확도 0.909
k :  9
학습용셋 정확도 0.946
테스트용셋 정확도 0.909
k :  11
학습용셋 정확도 0.939
테스트용셋 정확도 0.909
k :  13
학습용셋 정확도 0.937
테스트용셋 정확도 0.916
k :  15
학습용셋 정확도 0.939
테스트용셋 정확도 0.916
k :  17
학습용셋 정확도 0.934
테스트용셋 정확도 0.923
k :  19
학습용셋 정확도 0.937
테스트용셋 정확도 0.923
k :  21
학습용셋 정확도 0.934
테스트용셋 정확도 0.923
```

In [30]:

```
import seaborn as sns
print(sns.__version__)
```

```
0.11.0
```

In [31]:

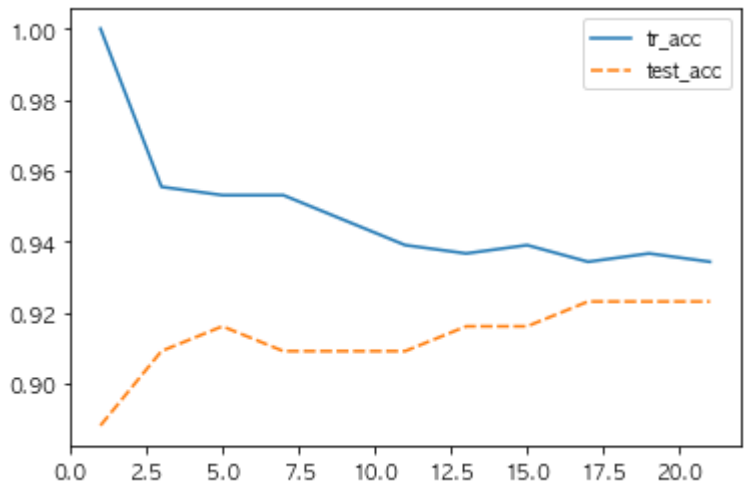
```
# tr_acc = []
# test_acc = []
dat = { "tr_acc":tr_acc, "test_acc":test_acc }
data_df = pd.DataFrame(dat, index=range(1, 22, 2))
data_df
```

Out[31]:

	tr_acc	test_acc
1	1.000000	0.888112
3	0.955399	0.909091
5	0.953052	0.916084
7	0.953052	0.909091
9	0.946009	0.909091
11	0.938967	0.909091
13	0.936620	0.916084
15	0.938967	0.916084
17	0.934272	0.923077
19	0.936620	0.923077
21	0.934272	0.923077

In [32]:

```
sns.lineplot(data=data_df, palette="tab10")
plt.show()
```



직접 해보기

- k을 1부터 100까지 돌려보고 가장 높은 값을 갖는 k의 값을 구해보자.

실습해 보기

- titanic 데이터 셋을 활용하여 knn 모델을 구현한다.
- 가장 높은 일반화 성능을 갖는 k의 값은 무엇인지 찾아보자.

더 해보기

- 이를 그래프로 표현해 보기
- Bike 데이터 셋을 knn모델을 활용하여 예측해 보기