In [ ]:

```
from preamble import *
```

지도 학습 알고리즘

k-최근접 이웃

k-최근접 이웃 분류

In [ ]:

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

In [ ]:

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

In [ ]:

```
from sklearn.model_selection import train_test_split
X, y = mglearn.datasets.make_forge()

print(X.shape)
print(y.shape)

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

print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

In [ ]:

```
from sklearn.neighbors import KNeighborsClassifier
clf = KNeighborsClassifier(n_neighbors=3)
```

In [ ]:

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

In [ ]:

```
y_pred = clf.predict(X_test)
print("테스트 세트 예측:", y_pred)
print("테스트 세트 참값:", y_test)
```

In [ ]:

```
print("테스트 세트 정확도: {:.2f}".format(clf.score(X_test, y_test)))
```

KNeighborsClassifier 분석

In [ ]:

```
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
    clf = KNeighborsClassifier(n_neighbors=n_neighbors).fit(X, y)

    mglearn.plots.plot_2d_separator(clf, X, fill=True, eps=0.5, ax=ax, alpha=.4)
    mglearn.discrete_scatter(X[:, 0], X[:, 1], y, ax=ax)

    ax.set_title("{} neighbor".format(n_neighbors))
    ax.set_xlabel("feature 0")
    ax.set_ylabel("feature 1")
```

```
axes[0].legend(loc='best')
```

In [ ]:

```
from sklearn.datasets import load_breast_cancer

cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(
    cancer.data, cancer.target, stratify=cancer.target, random_state=66)

training_accuracy = []
test_accuracy = []

# 1 에서 10 까지 n_neighbors 를 적용
neighbors_settings = range(1, 11)

for n_neighbors in neighbors_settings:
    # 모델 생성
    clf = KNeighborsClassifier(n_neighbors=n_neighbors)
    clf.fit(X_train, y_train)

    # 훈련 세트 정확도 저장
    training_accuracy.append(clf.score(X_train, y_train))

    # 일반화 정확도 저장
    test_accuracy.append(clf.score(X_test, y_test))

plt.plot(neighbors_settings, training_accuracy, label="train accuracy")
plt.plot(neighbors_settings, test_accuracy, label="test accuracy")
plt.ylabel("accuracy")
plt.xlabel("n_neighbors")
plt.legend()
```

k-Neighbors Regression

In [ ]:

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

In [ ]:

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

In [ ]:

```
from sklearn.neighbors import KNeighborsRegressor

X, y = mglearn.datasets.make_wave(n_samples=40)

# wave 데이터셋을 훈련 세트와 테스트 세트로 나눕니다
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

print(X_train.shape)
print(X_test.shape)

# 이웃의 수를 3으로 하여 모델의 객체를 만듭니다
reg = KNeighborsRegressor(n_neighbors=3)
# 훈련 데이터와 타깃을 사용하여 모델을 학습시킵니다
reg.fit(X_train, y_train)
```

In [ ]:

```
print("테스트 세트 예측:₩n", reg.predict(X_test))
```

In [ ]:

```
print("테스트 세트 R^2: {:.2f}".format(reg.score(X_test, y_test)))
```

KNeighborsRegressor 분석

In [ ]:

```
fig, axes = plt.subplots(1, 3, figsize=(15, 4))

# -3 과 3 사이에 1,000 개의 데이터 포인트를 만듭니다
line = np.linspace(-3, 3, 1000).reshape(-1, 1)

for n_neighbors, ax in zip([1, 3, 9], axes):

    # 1, 3, 9 이웃을 사용한 예측을 합니다
    reg = KNeighborsRegressor(n_neighbors=n_neighbors)
    reg.fit(X_train, y_train)
    ax.plot(line, reg.predict(line))
    ax.plot(X_train, y_train, '^', c=mglearn.cm2(0), markersize=8)
    ax.plot(X_test, y_test, 'v', c=mglearn.cm2(1), markersize=8)

    ax.set_title(
        "{}-neighbor : {:.2f} test score: {:.2f}".format(
            n_neighbors, reg.score(X_train, y_train), reg.score(X_test, y_test)))
    ax.set_xlabel("feature")
    ax.set_ylabel("target")
axes[0].legend(["model predictions", "train data/target", "test data/target"], loc="be
```

In [ ]:

```
```