머신러닝
(Machine Learning)

LECTURE VI: 머신러닝의 개요 2
(Brief Overview of Machine Learning)

Dai-Gyoung Kim

Department of Applied Mathematics

Hanyang University ERICA

머신러닝의 개요 (Brief Overview of Machine Learning)

Contents

- 머신러닝을 위한 파이썬 라이브러리
- 왜 머신러닝을 사용하는가?
- 머신러닝 시스템의 종류
- 머신러닝의 주요 도전과제
- 테스트와 검증
- 첫 번째 어플리케이션

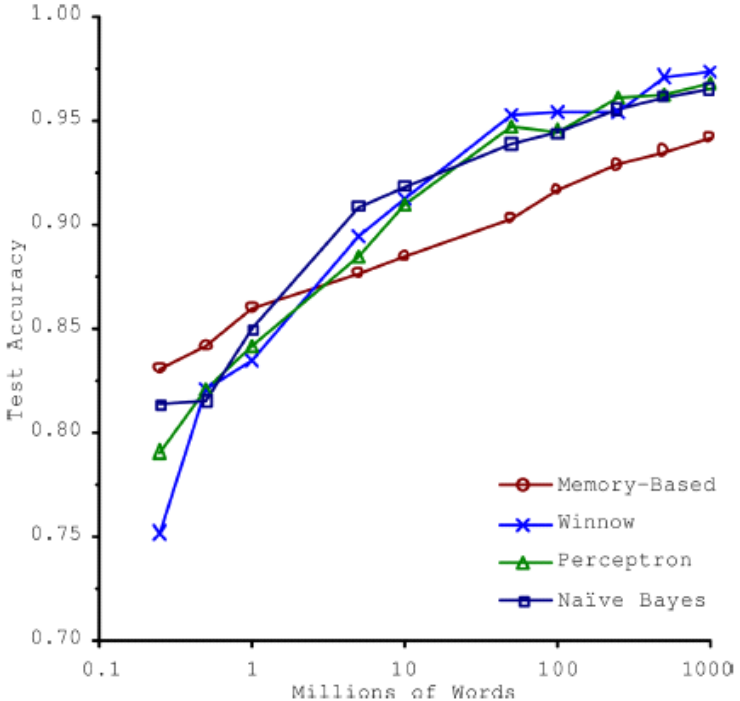
■ 머신러닝의 주요 도전 과제

머신러닝에서 주요 작업은 학습 알고리즘을 잘 선택하여 양질의 데이터로 잘 훈련시키는 것임. 따라서 문제가 될 수 있는 것은 '**나쁜 알고리즘**'과 '**나쁜 데이터**'를 사용하는 것임.

- 충분하지 않은 양의 훈련 데이터
- 대표성 없는 훈련 데이터
- 낮은 품질의 데이터
- 관련 없는 특성을 갖는 데이터
- 훈련 데이터의 과대적합
- 훈련 데이터의 과소적합

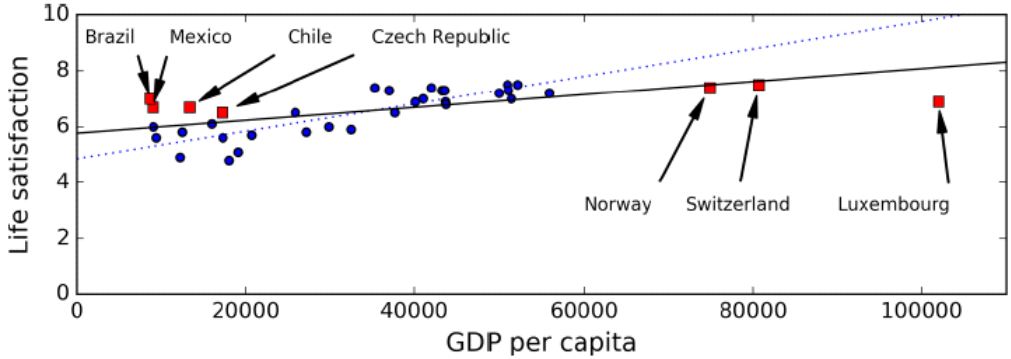
❖ 충분하지 않은 양의 훈련 데이터

- 대부분의 머신러닝 알고리즘이 잘 작동하려면 데이터가 충분히 많아야 함.



❖ 대표성 없는 훈련 데이터

- 시스템이 일반화가 잘되려면 우리가 일반화하기 원하는 새로운 사례를 훈련 데이터가 잘 대표하는 것이 중요함.



✓ 샘플이 작으면 **샘플링 잡음** (sampling noise)이 생길 수 있음.
(우연에 의한 대표성이 없는 데이터를 포함할 수 있음)

✓ 샘플이 매우 큰 경우에도 표본 추출 방법이 잘못되면 **샘플링 편향** (sampling bias)이 생길 수 있음. (대표성을 띠지 못한 데이터 추출)

❖ 낮은 품질의 데이터

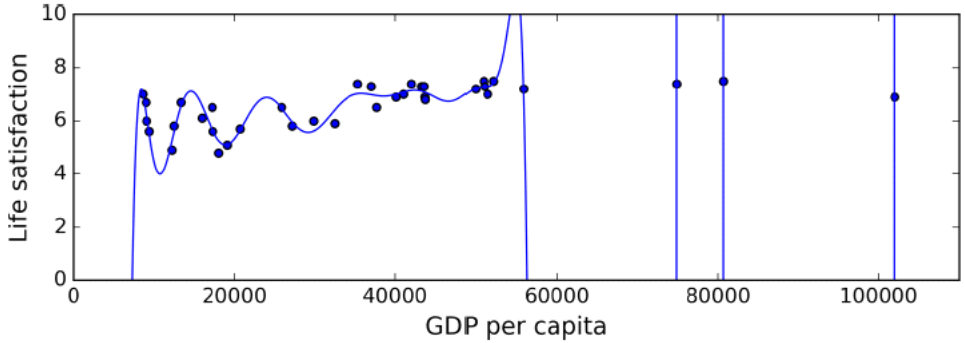
- 훈련 데이터가 에러, 이상치, 잡음으로 가득하면, 머신러닝 시스템이 내재된 패턴을 찾기 어려워 잘 작동하지 못할 수 있음.
  - ✓ 데이터를 분석하여 데이터 정제 (데이터 과학자)

❖ 관련 없는 특성을 갖는 데이터

- 훈련 데이터에 관련 없는 특성이 적고 관련 있는 특성이 충분해야 시스템이 학습할 수 있음.
  - ✓ 성공적인 머신러닝의 프로젝트의 핵심 요소는 훈련에 사용할 좋은 특성들을 찾는 것
  - ✓ 특성 공학 (feature engineering)
    - **특성 선택**: 가지고 있는 특성 중에서 훈련에 가장 유용한 특성을 선택
    - **특성 추출**: 특성을 결합하여 더 유용한 특성을 만듦
    - **새로운 데이터**를 수집하여 **새 특성**을 만듦.

❖ 훈련 데이터 과대적합

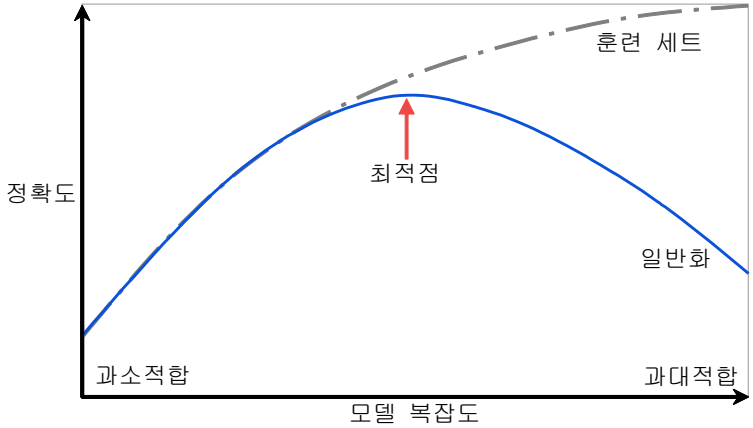
- 가진 정보를 모두 사용하여 너무 복잡한 모델을 만드는 것을 **과대적합** (overfitting)이라고 함.
  - ✓ 모델의 복잡도는 모델 파라미터의 개수와 관련됨
  - ✓ 과대적합 모델은 훈련 데이터셋에 잘 맞춰져 있지만 새로운 데이터에 일반화되기 어려움.
  - ✓ 과대적합 모델은 훈련 데이터셋에 잡음이 있을 때, 잡음이 섞인 패턴을 도출 할 수 있음.



- 과대적합을 줄이는 방법
  - ✓ 파라미터의 수가 적은 모델을 선택
  - ✓ 훈련 데이터에 있는 특성 수를 줄임
  - ✓ 모델에 제약을 가하여 단순화시킴 (모델 규제, regularization)
  - ✓ 훈련 데이터를 더 많이 수집
  - ✓ 훈련 데이터의 잡음을 줄임
  - ✓ 데이터를 맞추는 것과 모델을 단순화하는 것 사이의 균형을 최적화함 (규제 하이퍼파라미터 (hyperparameter)로 튜닝)

❖ 훈련 데이터 과소적합

- **과소적합** (underfitting)은 과대적합의 반대 개념으로, 모델이 너무 단순하여 데이터의 내재된 구조를 학습하지 못할 경우에 발생함.
- 과소적합을 줄이는 방법
  - ✓ 파라미터의 수가 더 많은 모델을 선택
  - ✓ 훈련 데이터에 더 좋은 특성을 제공 (특성공학)
  - ✓ 모델의 제약을 줄임



요약

- 머신러닝은 명시적인 규칙을 프로그램하지 않고 기계가 데이터부터 학습하여 어떤 작업을 더 잘 하도록 만드는 것.
- 여러 종류의 머신러닝 시스템이 존재함.
  - ✓ 지도학습, 비지도학습, 배치학습, 온라인 학습, 사례 기반 학습, 모델 기반 학습 등.
- 머신러닝 프로젝트에서는 훈련 세트에 데이터를 모아 학습 알고리즘을 주입함.
  - ✓ 모델 기반 학습 알고리즘은 훈련 세트에 모델을 맞추기 위해 파라미터를 조정하고, 새로운 데이터에서도 좋은 예측을 만들 것을 기대.
  - ✓ 사례 기반 학습 알고리즘에서는 샘플을 기억하는 것이 학습이고 새로운 샘플을 일반화하기 위해 유사도를 측정함.
- 훈련 세트가 너무 작거나, 대표성이 없는 데이터이거나, 잡음이 많고 관련 없는 특성으로 오염되어 있다면, 시스템이 잘 작동하지 않음.
- 모델이 너무 단순 (과소 적합)하거나, 너무 복잡 (과대 적합)하지 않아야 함.

■ 테스트와 검증

모델이 새로운 샘플에 얼마나 잘 일반화될지 아는 유일한 방법은 새로운 데이터에 실제로 적용해보는 것임. 그러나 실제로 훈련 데이터를 적절히 나누어 모델에 적용하여 평가하고 이를 바탕으로 성능을 향상시켜야 함. 중요한 것은 모델이 훈련 세트에 잘 맞는 것보다, 학습 과정에 없는 테스트 세트에 대해 예측을 얼마나 잘 하느냐가 중요함.

훈련 세트와 테스트 세트

- 주어진 데이터를 **훈련 세트**와 **테스트 세트**로 나눔.
  - ✓ 대용량 데이터를 훈련 세트 75~80%, 테스트 세트 20~25% 정도로 나눔.
  - ✓ 테스트 세트는 대표성이 있도록 무작위로 단 한번만 추출함.
- 훈련 세트로 모델을 학습하고, 테스트 세트로 모델의 성능을 평가함.

- 새로운 샘플에 대한 오류 비율을 **일반화 오차**(generalization error) 또는 **외부 샘플 오차**(out-of-sample error)라고 함.
- 테스트 세트에서 모델을 평가함으로써 이 오차에 대한 추정 값을 얻음.
  - ✓ 이 추정 값은 새로운 샘플에 모델이 얼마나 잘 작동하는 지 알려줌.
  - ✓ 훈련 오차와 일반화 오차를 비교 평가함.

검증 세트의 도입

- 일반화 오차를 테스트 세트에서 여러 번 측정함으로써, 모델과 하이퍼파라미터가 테스트 세트에 최적화되어 향상된 모델을 얻을 수 있으나, 이 최적화된 모델이 새로운 데이터에 잘 작동하지 않을 수 있음.
- 이러한 문제를 해결하기 위해 **검증 세트**(validation set)를 만들어 모델을 검증함.
  1) 훈련 세트를 사용해 다양한 하이퍼파라미터로 여러 모델을 훈련시키고 검증 세트에서 최상의 성능을 내는 모델과 하이퍼파라미터를 선택함.
  2) 만족스러운 모델을 찾으면 일반화 오차의 추정값을 얻기 위해 테스트 세트로 단 한 번의 최종 테스트를 수행함.

교차 검증

- 훈련 데이터에서 검증 세트로 너무 많은 양의 데이터가 할당되지 않기 위해 일반적으로 **교차 검증** (cross-validation) 기법을 사용함.
- 훈련 세트를 여러 서브셋으로 나누어 각 모델을 이 서브셋의 조합으로 훈련시키고 나머지 부분으로 검증을 함.
- 모델과 하이퍼파라미터가 선택되면 전체 훈련 데이터를 사용하여 선택한 하이퍼파라미터로 최종 모델을 훈련시키고, 테스트 세트에서 일반화 오차를 측정함.
- **교차 검증은 새로운 데이터에 적용할 모델을 만드는 것이 아니라, 단지 주어진 데이터 세트에 학습된 모델이 얼마나 잘 일반화될지 평가하는 것임.**

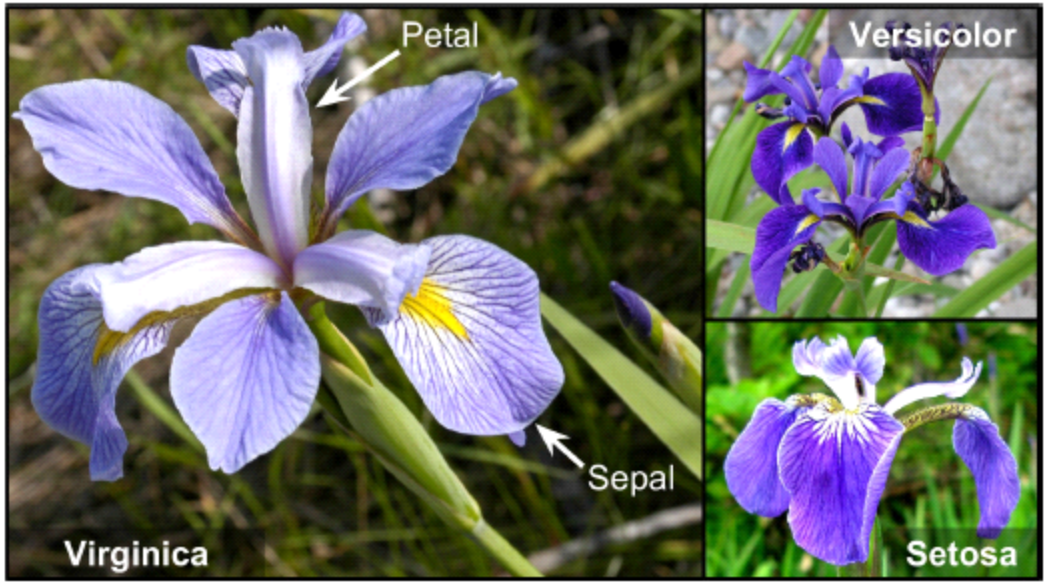
■ 첫 번째 애플리케이션: 붓꽃의 품종 분류

간단한 머신러닝 애플리케이션을 소개하면서 첫 번째 모델을 만들고, 이때 사용되는 머신러닝의 핵심 개념과 용어를 소개함.

[붓꽃의 품종 분류 문제]

- 붓꽃(Iris) 데이터셋은 세 개의 품종 setosa, versicolor, virginica에 속하는 붓꽃 150개의 꽃잎(petal)과 꽃받침(sepal)의 너비와 길이를 담고 있음.
- 이 데이터셋을 이용하여 새로 채집한 붓꽃의 품종을 예측하는 머신러닝 모델을 만드는 것이 목표임.

- 위 문제는 붓꽃 품종을 정확하게 분류한 데이터를 가지고 있으므로 **지도 학습**에 속함.
- 붓꽃의 품종(세 개의 클래스) 중에 하나를 선택하는 **분류** 문제임.



- 데이터 포인트 하나(붓꽃 하나)에 대한 기대 출력은 꽃의 품종이 되며, 이러한 특정 데이터 포인트에 대한 출력(품종)을 **레이블**(label)이라고 함.

❖ 데이터 적재

- 먼저 'scikit-learn'의 dataset 모듈에 포함되어 있는 'load_iris' 함수를 사용해 데이터를 적재함.

```
from sklearn.datasets import load_iris

iris_dataset = load_iris()
print("Keys of iris_dataset:\n{}".format(iris_dataset.keys()))
```

```
Keys of iris_dataset:
dict_keys(['data', 'target', 'target_names', 'DESCR', 'feature_names'])
```

✓ iris 객체는 딕셔너리와 유사한 Bunch라는 클래스 형식의 객체며, 이 클래스 객체는 다음과 같은 속성을 가짐.

1) **data** : 독립변수 ndarray 배열
2) **target** : 종속변수 ndarray 배열
3) **feature_names** : 독립변수 이름 리스트
4) **target_names** : 종속변수 이름 리스트
5) **DESCR** : 데이터에 대한 설명

- 다음은 DESCR 키에 있는 데이터셋에 대한 출력을 보여줌.

```
print(iris_dataset['DESCR'][:193] + "\n...")
```

```
Iris Plants Database
====================

Notes
----
Data Set Characteristics:
    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, predictive att

...
```

- target_names

```
print("Target names: {}".format(iris_dataset['target_names']))
```

```
Target names: ['setosa' 'versicolor' 'virginica']
```

- feature_names의 값은 각 특성을 설명하는 문자열 리스트임.

```
print("Feature names:\n{}".format(iris_dataset['feature_names']))
```

```
Feature names:
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)',
 'petal width (cm)']
```

- 실제 데이터는 'target'과 'data' 필드에 있음.

```
print("Type of data:{}".format(type(iris_dataset['data'])))
```

```
Type of data: <class 'numpy.ndarray'>
```

```
print("Shape of data:{}".format(iris_dataset['data'].shape))
```

```
Shape of data: (150, 4)
```

- 머신러닝에서 각 아이템 또는 데이터 포인트를 **샘플**이라고 하고, 속성은 **특성**이라고 함.
- 다음은 붓꽃 데이터셋의 맨 처음 다섯 샘플의 특성값을 보여줌.

```
print("First five rows of data:\n{}".format(iris_dataset['data'][:5]))
```

```
First five rows of data:
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]]
```

- 'target' 배열도 샘플 붓꽃 품종을 담은 NumPy 배열임.

```
print("Type of target: {}".format(type(iris_dataset['target'])))
```

```
Type of target: <class 'numpy.ndarray'>
```

```
print("Shape of target:{}".format(iris_dataset['target'].shape))
```

```
Shape of target: (150,)
```

```
print("Target:\n{}".format(iris_dataset['target']))
```

```
Target:
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

✓ 여기서 0은 setosa, 1은 versicolor, 2는 virginica임.

❖ 성능 측정: 훈련 데이터와 테스트 데이터

- 모델의 성능을 측정하려면 레이블을 알고 있는 새로운 데이터를 모델에 적용해봐야 함.
- 이를 위해 주어진 레이블된 데이터(150 개의 붓꽃 데이터셋)를 두 그룹으로 나눔.
  - ✓ 한 그룹은 머신러닝 모델을 만들 때 사용하며, **훈련 데이터** 또는 **훈련 세트**라고 함.
  - ✓ 또 하나의 그룹은 모델이 얼마나 잘 작동하는 지 측정하는 데 사용하며, 이를 **테스트 데이터**, **테스트 세트** 또는 **홀드아웃 세트**(hold-out set)라고 부름.

- scikit-learn은 데이터셋을 섞어서 나누어 주는 'train_test_split' 함수를 제공함.
  - ✓ 이 함수는 전체 데이터 샘플 중 75%를 레이블 데이터와 함께 훈련 세트로 뽑고, 나머지 25%는 레이블 데이터와 함께 테스트 데이터로 뽑음.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
        iris_dataset['data'], iris_dataset['target'], random_state=0)
```

✓ 데이터를 나누기 전에 'random_state=0'으로 하여 유사 난수 생성기(pseudorandom random number generator)를 사용해 데이터를 무작위로 섞어야 함.

```
print("X_train shape:", X_train.shape)
print("y_train shape:", y_train.shape)
```

```
X_train shape: (112, 4)
y_train shape: (112,)
```

```
print("X_test shape:", X_test.shape)
print("y_test shape:", y_test.shape)
```

```
X_test shape: (38, 4)
y_test shape: (38,)
```

❖ 가장 먼저 할 일: 데이터 살펴보기

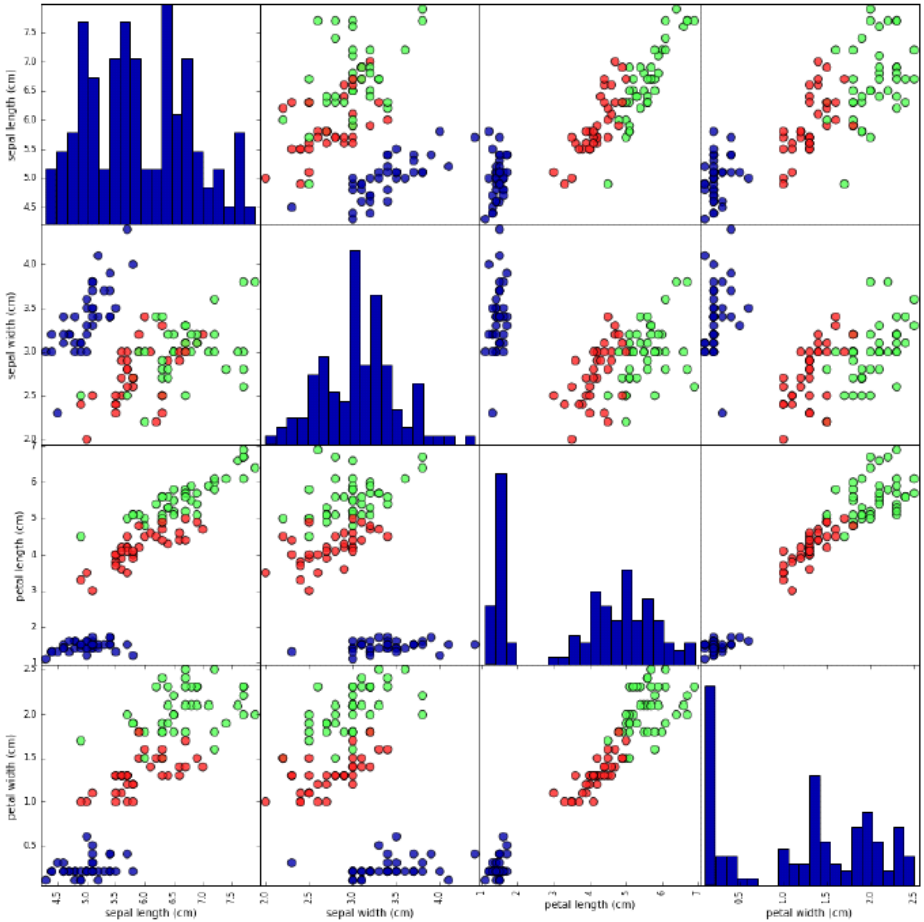
- 머신러닝 모델을 만들기 전에 데이터의 일관성이 없는지, 비정상적인 값이나 특이한 값들이 있는지, 또는 필요한 정보가 누락되어 있는지 사전에 데이터를 조사해 보는 것이 좋음.

- 데이터 시각화는 데이터를 조사하는데 아주 좋은 방법임.
  - ✓ 산점도(scatter plot)
  - ✓ 산점도 행렬(scatter matrix)

- 다음은 훈련 세트를 사용해 만든 4개의 특성에 대한 산점도 행렬을 생성하는 코드임.
  - ✓ 데이터 포인트의 색은 붓꽃의 품종에 따라 구분됨.
  - ✓ 산점도 그래프를 그리려면 먼저 NumPy 배열을 pandas의 DataFrame으로 변경해야함. pandas는 산점도 행렬을 그려주는 'scatter_matrix' 함수를 제공함.

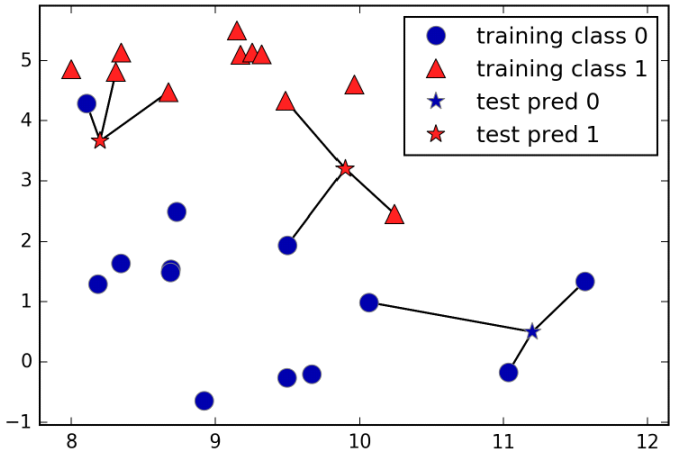
```
# create dataframe from data in X_train
# label the columns using the strings in iris_dataset.feature_names
iris_dataframe = pd.DataFrame(X_train, columns=iris_dataset.feature_names)

# create a scatter matrix from the dataframe, color by y_train
pd.plotting.scatter_matrix(iris_dataframe, c=y_train, figsize=(15, 15), marker='o',
                           hist_kwds={'bins': 20}, s=60, alpha=.8, cmap=mglearn.cm3)
```

✓ setosa(blue), versicolor(red), virginica(green)



❖ 첫 번째 머신러닝 모델: k-최근접 이웃 알고리즘

- k-**최근접 이웃**(k-Nearest Neighbors, k-NN) 분류기는 비교적 이해기 쉬운 직감적인 모델로서 scikit-learn에서 지원하고 있음.
- 이 모델은 단순히 훈련 데이터를 저장하고, 새로운 데이터 포인트에서 가장 가까운 훈련 데이터 포인트를 찾아 그 훈련 데이터의 레이블을 새 데이터 포인트의 레이블로 지정함.
- k-최근접 이웃 알고리즘에서 k는 가장 가까운 k개의 이웃을 찾아 이 이웃들의 클래스 중 빈도가 가장 높은 클래스를 예측값으로 사용함.



- k-최근접 이웃 분류 알고리즘은 'neighbors' 모듈 아래 'KNeighborsClassifier' 클래스에 구현되어 있음. 이 모델을 사용하려면 클래스로부터 객체를 만들어야 함.

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=1)
```

  - ✓ 'knn' 객체는 훈련 데이터로 모델을 만들고 새로운 데이터 포인트에 대해 예측을 수행하는 알고리즘을 캡슐화한 것이고, 그 알고리즘이 훈련 데이터로부터 추출한 정보를 담음.
  - ✓ 'KNeighborsClassfier'는 훈련 데이터 자체를 저장함.

- 훈련 데이터셋으로부터 모델을 만들려면 'knn' 객체의 'fit' 메서드를 사용함.
- 이 메서드는 훈련 데이터인 NumPy 배열 'X_train'과 훈련 데이터의 레이블을 담고 있는 NumPy 배열 'y_train'을 변수로 받음.

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

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=1, p=2, weights='uniform')
```

  - ✓ 'fit' 메서드는 'knn' 객체 자체를 반환함 ('knn' 객체 자체를 변경시킴).

❖ 예측하기

- 이제 위에서 만든 모델을 사용하여 새로운 데이터에 대해 예측을 만들고자 함.
- 예를 들어 야생에서 꽃받침의 길이가 5cm, 폭이 2.9cm이고 꽃잎의 길이가 1cm, 폭이 0.2cm인 붓꽃이 있다고 할 때, 이 붓꽃의 품종을 예측하고자 함.

```
X_new = np.array([[5, 2.9, 1, 0.2]])
print("X_new.shape: {}".format(X_new.shape))
```

```
X_new.shape: (1, 4)
```

✓ 이 측정값은 샘플수(1)에 특성의 수(4)인 NumPy 배열임.

- 예측은 'knn' 객체의 'predict' 메서드를 사용함.

```
prediction = knn.predict(X_new)
print("Prediction: {}".format(prediction))
print("Predicted target name: {}".format(iris_dataset['target_names'][prediction]))
```

```
Prediction: [0]
Predicted target name: ['setosa']
```

❖ 모델 평가하기

- 다음은 앞서 만든 테스트 세트를 사용하여 k-NN 모델의 성능을 평가함.

```
y_pred = knn.predict(X_test)
print("Test set predictions:\n {}".format(y_pred))
```

```
Test set predictions:
[2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0 0 1 1 0 2 1 0 2 2 1 0 2]
```

```
print("Test set score: {:.2f} ".format(np.mean(y_pred == y_test)))
```

```
Test set score: 0.97
```

```
print("Test set score: {:.2f}".format(knn.score(X_test, y_test)))
```

```
Test set score: 0.97
```

✓ 'knn' 객체의 'score' 메서드로도 테스트 세트의 정확도를 계산할 수 있음.

- 다음은 훈련과 평가의 과정을 담은 전체 코드임.

```
from sklearn.neighbors import KNeighborsClassifier

X_train, X_test, y_train, y_test = train_test_split(
        iris_dataset['data'], iris_dataset['target'], random_state=0)

knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train, y_train)

print("Test set score: {:.2f} ".format(knn.score(X_test, y_test)))
```

```
Test set score: 0.97
```