머신러닝
(Machine Learning)

LECTURE XVII: 비지도 학습과 데이터 전처리 1
(Unsupervised Learning and Preprocessing)

Dai-Gyoung Kim

Department of Applied Mathematics

Hanyang University ERICA

ⓒ dgkim@hanyang.ac.kr

비지도 학습과 데이터 전처리 (Unsupervised Learning and Preprocessing)

Contents

© dgkim@hanyang.ac.kr

■ 비지도 학습의 종류

두 가지 비지도 학습, 데이터의 **비지도 변환**과 **군집**을 살펴보고자 함.

[비지도 변환]

- 비지도 변환(unsupervised transformation)은 데이터를 새롭게 표현하여 머신러닝 알고리즘이 원 데이터보다 쉽게 해석하거나 계산할 수 있도록 하는 방법임.

- 비지도 학습에서 많이 쓰이는 변환은 **차원 축소**(dimension reduction)임.
  - ✓ 특성이 많은 고차원 데이터의 특성 수를 줄이면서 꼭 필요한 특징을 포함한 데이터로 표현하는 기법.
  - ✓ 차원 축소의 대표적인 예는 저차원으로의 데이터 시각화임.

ⓒ dgkim@hanyang.ac.kr

- 비지도 변환으로 데이터를 구성하는 단위나 성분을 찾기도 함. 예를 들어, 텍스트 문서에서 주제를 추출하는 문제임.

[군집 알고리즘]

- 군집(clustering)은 주어진 데이터에서 비슷한 것끼리 묶는 기법임.

ⓒ dgkim@hanyang.ac.kr

■ 비지도 학습의 도전과제

- 비지도 학습에서 가장 어려운 과제는 적용한 비지도 학습이 어떤 유용한 것을 학습하였는지 평가하는 것임.

- 비지도 학습 알고리즘은 데이터 과학자가 데이터를 더 잘 이해하고 싶을 때 탐색적 단계에서 많이 사용됨.

- 비지도 학습은 지도 학습의 전처리 단계에서 많이 사용됨.
  - ✓ 비지도 학습 결과로 새롭게 표현된 데이터를 사용해 학습하면 지도 학습의 정확도가 높아지며 메모리와 시간이 절약될 수 있음.

ⓒ dgkim@hanyang.ac.kr

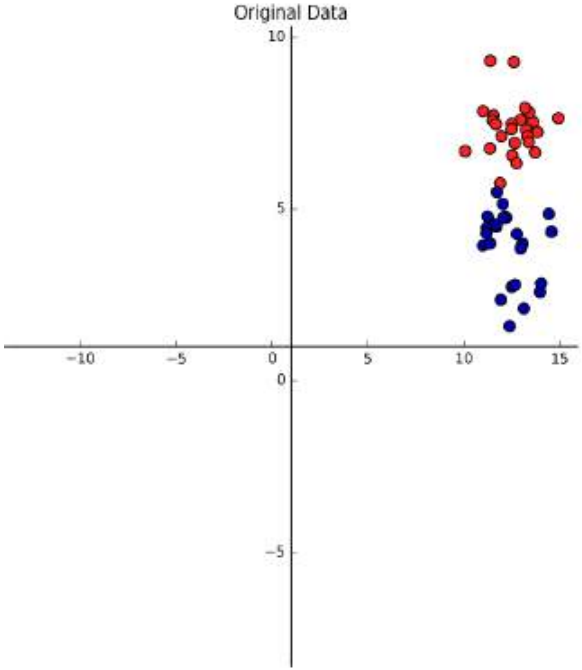
■ 데이터 전처리와 스케일 조정

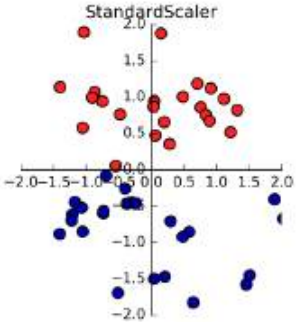
- 데이터에 적용할 가장 중요한 변환 중 하나는 데이터의 특성 스케일링(feature scaling)임.
  - ✓ 타깃값에 대한 스케일링은 일반적으로 수행하지 않음.

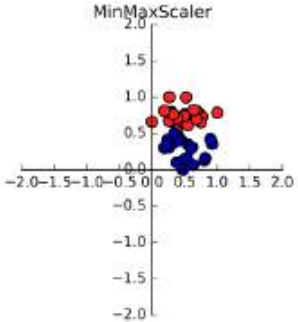
- 대부분 머신러닝 알고리즘은 입력 특성의 스케일이 많이 다르면 잘 작동하지 않음.
  - ✓ 특히, 신경망과 SVM 같은 알고리즘은 데이터 스케일에 매우 민감함.

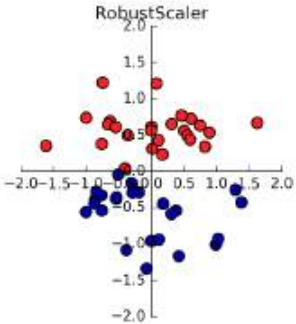
- 다음은 데이터셋의 스케일을 조정하거나 전처리하는 방법의 예를 보여줌.

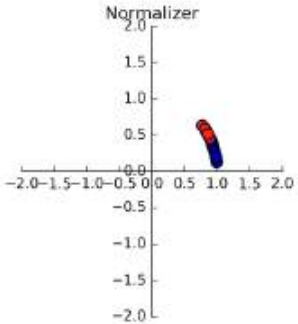
ⓒ dgkim@hanyang.ac.kr











© dgkim@hanyang.ac.kr

❖ 여러 가지 전처리 방법

1) StandardScaler (범용적)

- ✓ 모든 특성의 성분 값의 평균을 0, 분산을 1로 변환함.

2) RobustScaler

- ✓ 평균과 분산 대신 각 특성의 중앙값, 사분위 값을 사용함.
- ✓ $\frac{x - q_2}{q_3 - q_1}$

3) MinMaxScaler (범용적)

- ✓ 모든 특성의 성분 값을 정확하게 0과 1사이에 위치하도록 변환함.

4) Normalizer

- ✓ 특성 벡터의 유클리디안 길이가 1이 되도록 정규화 함.

ⓒ dgkim@hanyang.ac.kr

❖ 데이터 변환 적용하기

- 스케일을 조정하는 전처리 메서드들은 지도 학습 알고리즘을 적용하기 전에 적용함.
- 다음은 cancer 데이터셋에 MinMaxScaler를 사용하여 데이터 전처리를 수행하고 커널 SVM(SVC)를 적용하는 예제임.

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(cancer.data, cancer.target, random_state=1)

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

```
(426, 30)
(143, 30)
```

✓ 데이터셋은 569개의 샘플과 각 샘플은 30개의 측정(특성)값으로 이루어져 있음.

© dgkim@hanyang.ac.kr

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()

scaler.fit(X_train)
```

```
MinMaxScaler(copy=True, feature_range=(0, 1))
```

✓ 스케일 객체는 'fit' 메서드를 호출할 때 훈련 데이터만 입력하며 타깃 데이터는 입력하지 않음.

- 'fit' 메서드로 학습한 변환을 사용하려면(실제로 훈련 데이터의 스케일을 조정) 스케일 객체의 'transform' 메서드를 사용함.
- scikit-learn의 'transform'은 새로운 데이터 표현을 만들 때 사용하는 메서드임.

© dgkim@hanyang.ac.kr

```
# transform data
X_train_scaled = scaler.transform(X_train)
# print dataset properties before and after scaling
print("transformed shape: {}".format(X_train_scaled.shape))
print("per-feature minimum before scaling:\n {}".format(X_train.min(axis=0)))
print("per-feature maximum before scaling:\n {}".format(X_train.max(axis=0)))
print("per-feature minimum after scaling:\n {}".format(X_train_scaled.min(axis=0)))
print("per-feature maximum after scaling:\n {}".format( X_train_scaled.max(axis=0)))
```

```
transformed shape: (426, 30)
per-feature minimum before scaling:

[ 6.981   9.71    43.79    143.5     0.053    0.019    0.       0.       0.106
  0.05    0.115   0.36      0.757   6.802    0.002    0.002    0.       0.
  0.01    0.001   7.93     12.02   50.41   185.2      0.071    0.027    0.
  0.      0.157   0.055 ]

per-feature maximum before scaling:

[ 28.11   39.28   188.5    2501.00     0.163    0.287    0.427    0.201    0.304
    0.096  2.873    4.885   21.98   542.20     0.031    0.135    0.396    0.053
    0.061  0.03    36.04    49.54   251.20  4254.00     0.223    0.938    1.170
    0.291  0.577    0.149]

per-feature minimum after scaling:
[ 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.]
per-feature maximum after scaling:
[ 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.]
```

- scikit-learn의 'transform'은 새로운 데이터 표현을 만들 때 사용하는 메서드임.
- 위의 변환된 데이터에 SVM를 적용하여 머신을 구축하려면 테스트 세트도 변환해야함.

```
# transform test data
X_test_scaled = scaler.transform(X_test)
# print test data properties after scaling
print("per-feature minimum after scaling:\n{}".format(X_test_scaled.min(axis=0)))
print("per-feature maximum after scaling:\n{}".format(X_test_scaled.max(axis=0)))
```

```
per-feature minimum after scaling:
[  0.034   0.023   0.031   0.011   0.141   0.044   0.      0.      0.154
  -0.006  -0.001   0.006   0.004   0.001   0.039   0.011   0.      0.
  -0.032   0.007   0.027   0.058   0.02    0.009   0.109   0.026   0.
   0.     -0.     -0.002 ]
per-feature maximum after scaling:
[  0.958   0.815   0.956   0.894   0.811   1.22    0.88    0.933   0.932
   1.037   0.427   0.498   0.441   0.284   0.487   0.739   0.767   0.629
   1.337   0.391   0.896   0.793   0.849   0.745   0.915   1.132   1.07
   0.924   1.205   1.631 ]
```


© dgkim@hanyang.ac.kr

✓ 'transform' 메서드는 테스트 세트의 **최솟값과 범위를 사용하지 않고**, 항상 훈련 세트의 최솟값을 빼고 훈련 세트의 범위로 나눔.

❖ 훈련 데이터와 테스트 데이터의 스케일을 같은 방법으로 조정하기

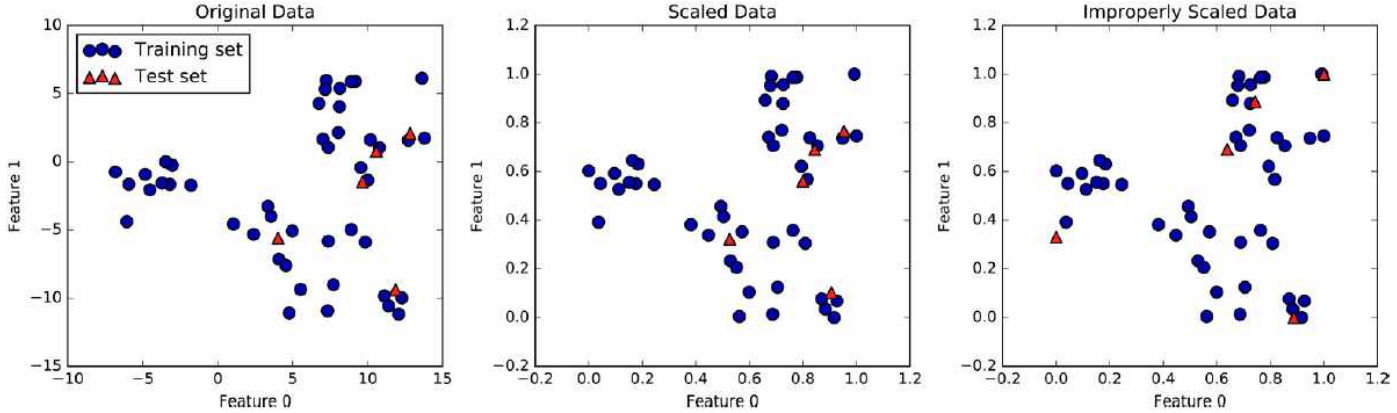
- 지도 학습 모델에서 테스트 세트를 사용하려면 훈련 세트와 테스트 세트에 같은 변환을 적용해야 함.
- 테스트 세트를 따로 변환할 경우 훈련 세트와 다른 비율로 변환되어 데이터 배열의 일관성이 떨어짐.
- 다음은 'MinMaxScaler'를 사용하여 위 경우의 예시를 보여준 것임.

ⓒ dgkim@hanyang.ac.kr

```
from sklearn.datasets import make_blobs
# make synthetic data
X, _ = make_blobs(n_samples=50, centers=5, random_state=4, cluster_std=2)
# split it into training and test sets
X_train, X_test = train_test_split(X, random_state=5, test_size=.1)

# plot the training and test sets
fig, axes = plt.subplots(1, 3, figsize=(13, 4))
axes[0].scatter(X_train[:, 0], X_train[:, 1], c=mglearn.cm2(0), label="Training set", s=60)
axes[0].scatter(X_test[:, 0], X_test[:, 1], marker='^', c=mglearn.cm2(1), label="Test set", s=60)
axes[0].legend(loc='upper left')
axes[0].set_title("Original Data")

# scale the data using MinMaxScaler
scaler = MinMaxScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
# visualize the properly scaled data
axes[1].scatter(X_train_scaled[:, 0], X_train_scaled[:, 1], c=mglearn.cm2(0),
                label="Training set", s=60)
axes[1].scatter(X_test_scaled[:, 0], X_test_scaled[:, 1], marker='^', c=mglearn.cm2(1),
                label="Test set", s=60)
axes[1].set_title("Scaled Data")


# rescale the test set separately, so test set min is 0 and test set max is 1
# DO NOT DO THIS! For illustration purposes only.
test_scaler = MinMaxScaler()
test_scaler.fit(X_test)
X_test_scaled_badly = test_scaler.transform(X_test)
# visualize wrongly scaled data
axes[2].scatter(X_train_scaled[:, 0], X_train_scaled[:, 1], c=mglearn.cm2(0),
                label="training set", s=60)
axes[2].scatter(X_test_scaled_badly[:, 0], X_test_scaled_badly[:, 1], marker='^',
                c=mglearn.cm2(1), label="test set", s=60)
axes[2].set_title("Improperly Scaled Data")
```

```
for ax in axes:
    ax.set_xlabel("Feature 0")
    ax.set_ylabel("Feature 1")
fig.tight_layout()
```



✓ 훈련 데이터와 테스트 데이터의 스케일 조정을 함께 했을 때와 따로 했을 때의 결과.

© dgkim@hanyang.ac.kr

❖ 지도 학습에서 데이터 전처리 효과

- cancer 데이터셋에 SVM(SVC)를 학습시킬 때, MinMaxScaler의 효과를 확인해 보기.

```
from sklearn.svm import SVC
X_train, X_test, y_train, y_test = train_test_split(cancer.data, cancer.target, random_state=0)

svm = SVC(C=100)
svm.fit(X_train, y_train)
print("Test set accuracy: {:.2f}".format(svm.score(X_test, y_test)))
```

```
Test set accuracy: 0.63
```

✓ 위의 예시는 데이터 스케일 조정을 하지 않고 원본 데이터로 SVC 모델을 만든 경우임.

© dgkim@hanyang.ac.kr

```
# preprocessing using 0-1 scaling
scaler = MinMaxScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)

# learning an SVM on the scaled training data
svm.fit(X_train_scaled, y_train)

# scoring on the scaled test set
print("Scaled test set accuracy: {:.2f}".format( svm.score(X_test_scaled, y_test)))
```

```
Scaled test set accuracy: 0.97
```

- ✓ 위의 예시는 SVC 모델을 학습시키기 전에 MinMaxScaler를 사용해 데이터의 스케일을 조정하여 머신을 만든 경우임.
- ✓ 단지 스케일 조정만으로 정확도가 상당히 개선되었음을 볼 수 있음.

- 모든 전처리 모델이 동일한 'fit', 'transform' 메서드를 제공함. 따라서 사용하는 파이썬 클래스만 바꾸면 다른 전처리 알고리즘으로 손쉽게 교체할 수 있음.

```
# preprocessing using zero mean and unit variance scaling
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)

# learning an SVM on the scaled training data
svm.fit(X_train_scaled, y_train)

# scoring on the scaled test set
print("SVM test accuracy: {:.2f}".format(svm.score(X_test_scaled, y_test)))
```

```
SVM test accuracy: 0.96
```


© dgkim@hanyang.ac.kr