

특성 선택을 사용한 차원 축소

특성 선택을 사용한 차원 축소

➤ 특성 선택

- 특성 선택 (feature selection) : 고품질의 정보가 많은 특성은 선택하고 덜 유용한 특성은 버리는 방식
- 필터(filter) - 통계적인 속성을 조사하여 가장 뛰어난 특성을 선택
- 래퍼(wrapper) - 시행착오를 통해 가장 높은 품질의 예측을 만드는 특성의 부분 조합을 찾아 선택
- 임베디드(embedded) - 학습 알고리즘의 훈련 단계를 확장하거나 일부로 구성하여 가장 좋은 특성의 부분 조합을 선택

특성 선택을 사용한 차원 축소

➤ 분산을 기준으로 수치 특성 선택

- 분산 기준 설정(variance thresholding)은 가장 기본적인 특성 선택 방법 중 하나입니다.
- 분산이 높은 특성보다 분산이 낮은 특성이 효과적이거나 유용하지 않다는 아이디어에 기반합니다.
- 분산 기준 설정 VT(VarianceThreshold)는 특성의 제곱 단위이므로 특성의 단위가 서로 다르면 VT가 동작하지 않습니다.
- 분산의 기준값을 수동으로 선택하기 때문에 어떤 값이 좋은지 판단할 수 있어야 합니다.

```
from sklearn import datasets
from sklearn.feature_selection import VarianceThreshold
```

```
iris = datasets.load_iris() #데이터를 로드
```

```
features = iris.data # 특성과 타깃을 만듭니다
target = iris.target
```

```
threshold = VarianceThreshold(threshold=.5) # 기준값을 만듭니다.
```

```
features_high_variance = threshold.fit_transform(features) # 기준값보다 높은 특성을 선택합니다.
```

```
features_high_variance[0:3] # 선택한 특성을 확인
threshold.variances_ # 분산 확인
```

$$Var(x) = \frac{1}{n} \sum_{i=1}^n (x_i - \mu)^2$$

특성 선택을 사용한 차원 축소

➤ 분산을 기준으로 수치 특성 선택

- 특성이 평균이 0이고 단위 분산으로 표준화 되어 있으면 분산 기준 선택 방식은 올바르게 동작하지 않습니다.

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler() # 특성 행렬을 표준화합니다.
features_std = scaler.fit_transform(features)

selector = VarianceThreshold() # 각 특성의 분산을 계산합니다.
selector.fit(features_std).variances_
```

특성 선택을 사용한 차원 축소

➤ 분산을 기준으로 이진 특성 선택

- 이진 범주형 특성에서 베르누이 확률 변수의 분산이 기준값 이상인 특성을 선택할 수 있도록 분산이 낮은 특성(적은 정보를 가진 특성)을 삭제합니다
- $Var(x) = p(1-p)$
- p는 클래스 1의 샘플 비율입니다.
- p값을 설정하여 샘플의 대다수가 한 개의 클래스에 속한 특성을 삭제할 수 있습니다.
- VarianceThreshold 클래스는 수치 특성, 이진 특성에 상관없이 넘파이 var 함수를 사용하여 분산을 계산합니다.

```
from sklearn.feature_selection import VarianceThreshold

features = [[0, 1, 0],      # 예제 특성 행렬
            [0, 1, 1],      # 특성 0: 80%가 클래스 0
            [0, 1, 0],      # 특성 1: 80%가 클래스 1
            [0, 1, 1],      # 특성 2: 60%가 클래스 0, 40%는 클래스 1
            [1, 0, 0]]

# 분산을 기준으로 선택합니다.
thresholder = VarianceThreshold(threshold=(.75 * (1 - .75)))
thresholder.fit_transform(features)
thresholder.variances_

import numpy as np
np.var(features, axis=0) # 넘파이 var 함수를 사용하여 분산을 계산합니다
```

특성 선택을 사용한 차원 축소

➤ 분산을 기준으로 이진 특성 선택

- 이진 특성에 var 함수를 사용하는 것은 이진 특성일 때 베르누이 확률 변수의 분산과 같기 때문입니다.

$$\begin{aligned} Var(x) &= \frac{1}{n} \sum_{i=1}^n (x_i - \mu)^2 = \frac{1}{n} \left(\sum_{i=1}^n x_i^2 - 2\mu \sum_{i=1}^n x_i + n\mu^2 \right) \\ &= \frac{1}{n} \sum_{i=1}^n x_i - 2\mu \frac{1}{n} \sum_{i=1}^n x_i + \mu^2 = \mu - 2\mu^2 + \mu^2 = \mu - \mu^2 = \mu(1 - \mu) \end{aligned}$$

- 이진 특성의 평균 μ 는 클래스 1의 샘플 비율과 같습니다.
- var 함수로 이진 특성의 분산을 계산하면 베르누이 확률 변수의 분산 $p(1-p)$ 와 같습니다.
- threshold 매개변수의 기본값은 0으로 모든 특성을 선택합니다.

특성 선택을 사용한 차원 축소

➤ 상관관계가 큰 특성 삭제

- 두 가지 특성의 상관관계가 크다면, 담고 있는 정보가 매우 비슷하므로 중복된 특성을 포함하는 것과 같습니다.
 - 특성 행렬에서 상관관계 행렬을 사용하여 상관관계가 큰 특성을 확인하고 이들 중 하나를 삭제합니다.
1. 모든 특성에 대한 상관관계 행렬을 만듭니다.
 2. 상관관계 행렬의 상삼각 행렬(upper triangle matrix)을 살펴서 크게 상관된 특성의 쌍을 확인합니다
 3. 특성 행렬에서 이런 특성 중 하나를 삭제합니다.

```
import pandas as pd
import numpy as np

# 상관관계가 큰 두 개의 특성을 가진 특성 행렬을 만듭니다.
features = np.array([[1, 1, 1], [2, 2, 0], [3, 3, 1], [4, 4, 0], [5, 5, 1],
                    [6, 6, 0], [7, 7, 1], [8, 7, 0], [9, 7, 1]])

dataframe = pd.DataFrame(features) # 특성 행렬을 DataFrame으로 변환
corr_matrix = dataframe.corr().abs() # 상관관계 행렬을 만듭니다.
# 상관관계 행렬의 상삼각(upper triangle) 행렬을 선택합니다.
upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(np.bool))
# 상관 계수가 0.95보다 큰 특성 열의 인덱스를 찾습니다.
to_drop = [column for column in upper.columns if any(upper[column] > 0.95)]
```

특성 선택을 사용한 차원 축소

➤ 상관관계가 큰 특성 삭제

- 특성 행렬에서 상관관계 행렬을 사용하여 상관관계가 큰 특성을 확인하고 이들 중 하나를 삭제합니다.

```
dataframe.drop(dataframe.columns[to_drop], axis=1).head(3) # 특성을 삭제합니다.
```

```
dataframe.corr()    #상관관계 행렬  
upper              #상관관계 행렬의 상삼각 행렬
```

#상관관계 행렬은 **넘파이 corrcoef()**로 구할 수 있습니다.

#corrcoef()는 특성이 행에 놓여 있을 것으로 가정합니다.

#특성이 열에 놓여 있다고 알려주려면 rowvar 매개변수를 False로 지정합니다.

```
np.corrcoef(features, rowvar=False)
```

np.triu()는 주어진 배열에서 상삼각 행렬을 추출하여 반환합니다.

매개변수 k가 기본값 0이면 반환되는 행렬에 대각원소가 포함됩니다.

k값이 커질수록 대각원소에서 k만큼 떨어진 삼각행렬을 반환합니다.

예) k=2일 경우 주대각선에서 2만큼 떨어진 원소부터 포함됩니다.

```
np.triu(np.ones((4, 4)), k=2)
```

np.tril()는 주어진 배열에서 하삼각 행렬을 추출 반환합니다.

```
np.tril(np.ones((4, 4)), k=0)
```


특성 선택을 사용한 차원 축소

➤ 분류 작업에 관련 없는 특성 삭제

- 범주형 타깃 벡터에서 관련 없는 특성을 삭제하려면 각 특성과 타깃 벡터 사이의 **카이제곱 통계**를 계산합니다.
- 카이제곱 통계는 두 범주형 벡터의 **독립성을 평가**합니다.
- 카이제곱 통계는 범주형 특성의 각 클래스별 샘플 빈도와 이 특성이 타깃 벡터와 독립적이라면 기대할 수 있는 값 사이의 차이입니다
- 카이제곱 특성은 관찰 빈도와 전혀 관계가 없다고 기대하는 빈도 사이에 얼마나 큰 차이가 있는지 알려주는 하나의 숫자입니다
- 특성과 타깃 벡터 사이의 카이제곱 통계를 계산하면 둘 사이의 독립성을 측정할 수 있습니다.

```
from sklearn.datasets import load_iris
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2, f_classif

iris = load_iris() # 데이터 로드
features = iris.data
target = iris.target
features = features.astype(int) # 범주형 데이터를 정수형으로 변환

chi2_selector = SelectKBest(chi2, k=2) # 카이제곱 통계값이 가장 큰 특성 두 개를 선택
features_kbest = chi2_selector.fit_transform(features, target)

print("원본 특성 개수:", features.shape[1]) # 결과 확인
print("줄어든 특성 개수:", features_kbest.shape[1])
```

특성 선택을 사용한 차원 축소

➤ 분류 작업에 관련 없는 특성 삭제

- 특성 선택에서 카이제곱을 사용하려면 각 특성과 타깃 벡터 사이의 카이제곱 통계를 계산하고 카이제곱 통계가 가장 좋은 특성을 선택해야 합니다.

$$\chi^2 = \sum_{i=1}^n \frac{(O_i - E_i)^2}{E_i}$$

- 사이킷런에서는 SelectBest를 사용하여 통겅값이 가장 좋은 특성을 선택할 수 있습니다.
- 매개변수 k는 선택하려는 특성의 개수를 결정합니다
- 카이제곱 통계는 두 범주형 벡터 사이에서만 계산할 수 있습니다.
- 특성 선택으로 카이제곱을 사용하려면 타깃 벡터와 특성이 범주형이어야 합니다.
- 수치형 특성이 있다면 수치형을 범주형 특성으로 변환하여 카이제곱 특성을 사용할 수 있습니다.
- 카이제곱 방식을 사용하려면 모든 값이 음수가 아니어야 합니다.

특성 선택을 사용한 차원 축소

➤ 분류 작업에 관련 없는 특성 삭제

- 특성이 수치형 특성이라면 `f_classif` 사용하여 각 특성과 타깃 벡터 사이에 분산 분석(ANOVA)와 F-값 통계를 계산할 수 있습니다.
- F-값 점수는 타깃 벡터로 수치형 특성을 그룹핑하여 각 그룹의 평균이 크게 차이 나는지 평가합니다.
- 예] 이진 타깃 벡터인 성별과 수치형 특성인 시험 점수가 있다면, F-값 점수는 남성의 평균 테스트 점수가 여성의 평균 테스트 점수보다 다른지를 설명합니다.

```
# F-값이 가장 높은 특성 두 개를 선택합니다.
```

```
fvalue_selector = SelectKBest(f_classif, k=2)
```

```
features_kbest = fvalue_selector.fit_transform(features, target)
```

```
print("원본 특성 개수:", features.shape[1]) # 결과 확인
```

```
print("줄어든 특성 개수:", features_kbest.shape[1])
```

```
# 특정 특성 개수를 선택하는 대신 Selectpercentile를 사용하여 특성의 상위 n 퍼센트를 선택할 수 있습니다.
```

```
from sklearn.feature_selection import SelectPercentile
```

```
# 가장 큰 F-값의 상위 75% 특성을 선택합니다.
```

```
fvalue_selector = SelectPercentile(f_classif, percentile=75)
```

```
features_kbest = fvalue_selector.fit_transform(features, target)
```

```
print("원본 특성 개수:", features.shape[1]) # 결과 선택
```

```
print("줄어든 특성 개수:", features_kbest.shape[1])
```

특성 선택을 사용한 차원 축소

- 분류 작업에 관련 없는 특성 삭제
 - 카이제곱 계산

```
target
```

```
#특성 행렬의 차원을 (3, 50, 4)로 바꾸어 클래스별 합을 구합니다.
```

```
observed = np.sum(features.reshape(3, 50, 4), axis=1)
```

```
observed
```

```
#특성 타깃과 전혀 관계없다면 기대 빈도는 전체 합을 클래스 개수 3으로 나눈 값이 됩니다.
```

```
expected = features.sum(axis=0) / 3
```

```
Expected
```

```
#카이제곱 공식에 위해서 구한 observed와 expected를 대입합니다.
```

```
np.sum((observed - expected)**2 / expected, axis=0)
```

```
#카이제곱 값이 큰 세 번째, 네 번째 특성이 선택됩니다. chi2_selector객체의 scores_속성에 저장
```

```
chi2_selector.scores_
```

특성 선택을 사용한 차원 축소

➤ 분류 작업에 관련 없는 특성 삭제

- ANOVA는 각 특성이 독립적으로 평가되기 때문에 일변량 분석이라고도 부릅니다.

$$F = \frac{SS_{between} / (k - 1)}{(SS_{tot} - SS_{between}) / (n - k)} \quad SS_{between} = \sum_{j=1}^k n_j (\bar{x}_j - \bar{x})^2, \quad SS_{tot} = \sum_{i=1}^n (x_i - \bar{x})^2$$

```
#ANOVA 직접 계산
##전체 평균과 클래스 평균을 계산
total_mean = np.mean(features, axis=0)
total_mean
class_mean = np.mean(features.reshape(3, 50, 4), axis=1)
class_mean

#ss_total 계산
ss_between = np.sum(50 * (class_mean - total_mean)**2, axis=0)
ss_between
ss_total = np.sum((features - total_mean)**2, axis=0)
ss_total

#ss_beteen과 ss_tatal을 F-값 공식에 대입
f = (ss_between/(3-1)) / ((ss_total-ss_between)/(150-3))
f

fvalue_selector.scores_ #F-값 scores_속성에서 확인
```

특성 선택을 사용한 차원 축소

➤ 재귀적 특성 제거 (recursive feature elimination)

- 모델 성능이 나빠질 때까지 특성을 제거하면서 반복적으로 모델을 훈련하면 자동으로 최선의 특성만 남게 됩니다.
- 사이킷런의 RFECV를 사용하여 재귀적 특성 제거를 교차 검증(cross-validation)으로 수행할 수 있습니다

```
from sklearn.datasets import make_regression
from sklearn.feature_selection import RFECV
from sklearn import datasets, linear_model

# 특성 행렬과 타깃 벡터를 생성합니다.
features, target = make_regression(n_samples = 10000,
                                  n_features = 100,
                                  n_informative = 2,
                                  random_state = 1)

# 선형 회귀 모델을 만듭니다.
ols = linear_model.LinearRegression()

# 재귀적으로 특성을 제거합니다.
rfecv = RFECV(estimator=ols, step=1, scoring="neg_mean_squared_error")
rfecv.fit(features, target)
rfecv.transform(features)
```

특성 선택을 사용한 차원 축소

➤ 재귀적 특성 제거 (recursive feature elimination)

- 교차검증(CV)를 사용하여 RFE 과정에서 남길 특성의 최적 개수를 찾을 수 있습니다.
- 구체적으로 매 반복 후에 CV 를 사용한 RFE에서 교차검증을 사용하여 모델을 평가합니다.
- 특성을 제거한 후에 모델의 CV 결과가 향상되었다면 다음 반복으로 계속 진행합니다.
- 어떤 특성을 제거한 후에 모델의 CV 결과가 더 나빠지면 삭제한 특성을 다시 복원하고 이 특성 조합을 최선으로 선택합니다.
- 사이킷런의 RFECV는 CV를 사용한 REF 구현으로 estimator 매개변수에는 훈련한 모델의 객체를 전달합니다
- step 매개변수는 매 반복에서 삭제할 특성의 개수나 비율을 정합니다.
- scoring 매개변수에는 교차검증 동안 사용할 모델의 평가 지표를 설정합니다.

```
rfecv.n_features_      # 최선의 특성 개수
rfecv.support_         # 선택된 특성이 표시된 불리언 마스크
rfecv.ranking_         # 특성의 순위: 최고(1)에서 최악(96)까지

from sklearn.feature_selection import RFE

rfe = RFE(estimator=ols, n_features_to_select=3)
rfe.fit(features, target)
rfe.transform(features)
# rfe객체가 선택한 특성이 rfecv 객체가 선택한 특성과 동일한지 확인하기 위해 불리언 마스크를 비교
np.all(rfe.support_ == rfecv.support_)
```

특성 선택을 사용한 차원 축소

➤ 재귀적 특성 제거 (recursive feature elimination)

- step 매개변수의 기본값은 1입니다
- scoring 매개변수를 지정하지 않으면 estimator에 지정된 모델의 score ()를 사용합니다.
- n_jobs 매개변수에서 교차검증을 위해 사용할 CPU 코어 수를 지정할 수 있습니다
- n_jobs 매개변수의 기본값은 1입니다.
- cv 매개변수는 k-폴드 교차검증의 k값을 결정합니다. 기본값은 5입니다.
- 사이킷런은 교차검증을 사용하지 않는 재귀적 특성 제거 방법인 RFE 클래스를 제공합니다.
- RFECV 클래스와 마찬가지로 남길 최소 특성의 개수를 n_features_to_select 매개변수에서 지정할 수 있지만 RFECV와 달리 입력 특성의 절반이 기본값입니다.