20200113

Part.2

데이터 전처리: 범주형 데이터, 세트 나누기, 스케일 맞추기, 유용한 특성 선택

01
02
03
04

4.2 범주형 데이터

4.3 훈련세트와 테스트 세트로 나누기

4.4 특성 스케일 맞추기

4.5 유용한 특성 선택

4.6 랜덤포레스트의 특성 중요도 사용

01
02
03
04

범주형 데이터

순서가 있는 특성

순서가 없는 특성

4.2.2 순서 특성 매핑 mapping

범주형 문자열 → 정수

```
size_mapping = {'XL': 3,
                'L': 2,
                'M': 1}

df['size'] = df['size'].map(size_mapping)
df
```

```
inv_size_mapping = {v: k for k, v in size_mapping.items()}
df['size'].map(inv_size_mapping)
```

거꾸로 매핑하기

4.2.3 클래스 레이블 인코딩

-클래스 레이블을 0부터 할당

```
class_mapping = {label: idx for idx, label in enumerate(np.unique(df['classlabel']))}
class_mapping
```

-매핑 딕셔너리 사용해서 클래스 레이블을 정수로 변환

```
df['classlabel'] = df['classlabel'].map(class_mapping)
```

! 사이킷런의 LabelEncoder 클래스 사용하기

```
from sklearn.preprocessing import LabelEncoder

# 사이킷런의 LabelEncoder를 사용하여 레이블을 인코딩합니다.
class_le = LabelEncoder()
y = class_le.fit_transform(df['classlabel'].values)
y
```

Inverse_transform하면 원본으로 되돌리기

4.2.4 원-핫 인코딩

원-핫 인코딩: 순서 없는 특성에 들어 있는 고유한 값마다 새로운 더미 특성을 만든다. 이진 값을 사용해서 특성을 나타낸다.

-방법1:OneHotEncoder

```
from sklearn.preprocessing import OneHotEncoder

ohe = OneHotEncoder(categorical_features=[0])
ohe.fit_transform(X).toarray()
```

변환하려는 특성의 열위치 지정

-방법2:get_dummies

```
pd.get_dummies(df[['price', 'color', 'size']])
```

Drop_first=True 하면 첫번째 열 삭제

4.3 데이터셋을 훈련세트와 테스트세트로 나누기

편향되지 않은 성능을 측정하기 위해서!

```
from sklearn.model_selection import train_test_split

X, y = df_wine.iloc[:, 1:].values, df_wine.iloc[:, 0].values

X_train, X_test, y_train, y_test =₩
    train_test_split(X, y,
                     test_size=0.3,
                     random_state=0,
                     stratify=y)
```

테스트 세트 30%, 훈련 세트 70%

클래스 비율이 원래의 데이터 셋과 동일하게 유지됨.
Np.bincount(y)
Np.bincount(y_train)
Np.bincount(y_test) 를 통해 확인할 수 있다.

4.4 특성 스케일 맞추기

01

02

03

특성의 스케일을 맞추는 대표적인 방법-정규화와 표준화

정규화
(최소 최대 스케일
변환)

스케일을 0-1 범위에 맞춤.
범위가 정해진 값이 필요할 때
유용

표준
화

평균 0,표준편차 1로
최적화 알고리즘에서 사용
가중치 학습을 더 쉽게

```
from sklearn.preprocessing import MinMaxScaler

mms = MinMaxScaler()
X_train_norm = mms.fit_transform(X_train)
X_test_norm = mms.transform(X_test)
```

```
from sklearn.preprocessing import StandardScaler

stdsc = StandardScaler()
X_train_std = stdsc.fit_transform(X_train)
X_test_std = stdsc.transform(X_test)
```

4.5 유용한 특성 선택

교재 155p

01

02

03

04

과대적합 : 모델이 테스트 세트에서보다, 훈련세트에서 성능이 높을 경우

- 새로운 데이터에는 잘 일반화하지 못해서 모델 분산이 큼!

▶일반화 오차를 감소시키기 위한 방법

- 더 많은 훈련데이터 모음
- 규제를 통해 복잡도 제한
- 파라미터 개수가 적은 간단한 모델 선택
- 데이터 차원 줄임

▶ L2,L1규제 : 개별 가중치 값을 제한하여 모델 복잡도를 줄이는 방법

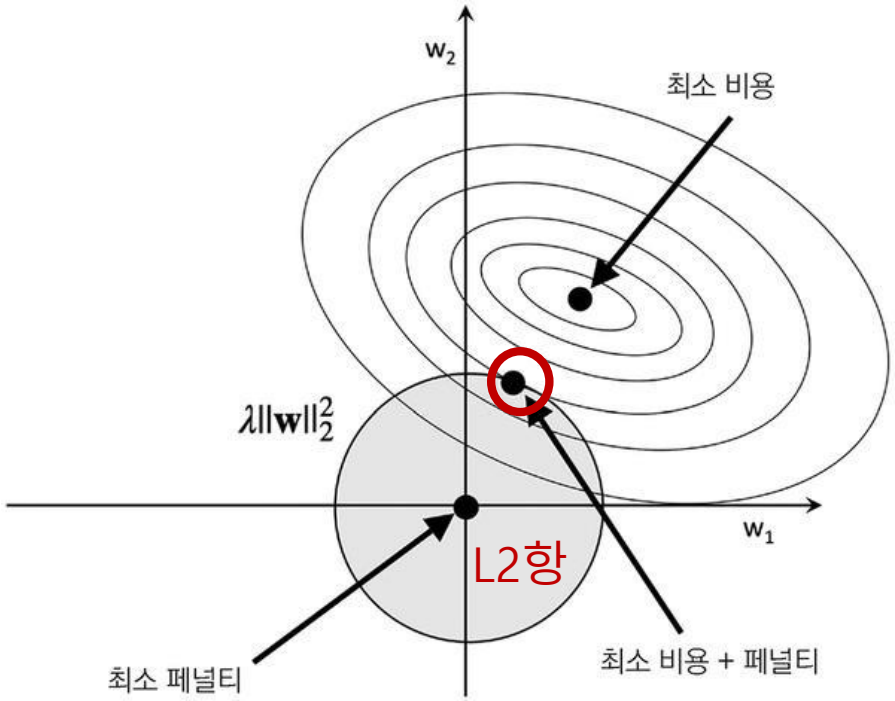
$$L2: \|\boldsymbol{w}\|_2^2 = \sum_{j=1}^{m} w_j^2$$

$$L1: \|\boldsymbol{w}\|_1 = \sum_{j=1}^{m} |w_j|$$

4.5.2 L2규제의 기하학적 해석

교재 156p

페널티 항을 추가한 제곱오차합 비용함수



$$L2 : \|\boldsymbol{w}\|_2^2 = \sum_{j=1}^{m} w_j^2$$

규제파라미터 λ 가 커질수록 > 패널티 비용이 증가 > 회색공이 작아짐

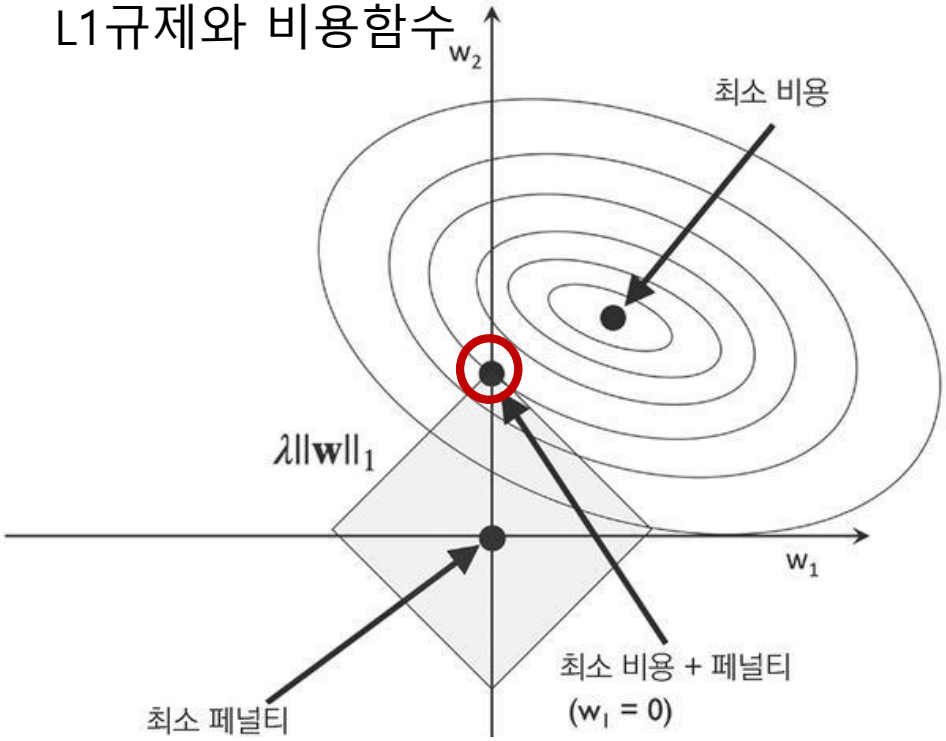
가중치 값은 회색공 바깥에 놓일 수 없다.
따라서 패널티 제약이 있을 때는 회색공과 비용함수의 등고선이 만나는 지점이 최선이다.

4.5.3 L1규제를 사용한 희소성

교재 157p

01
02
03
04

L1규제와 비용함수



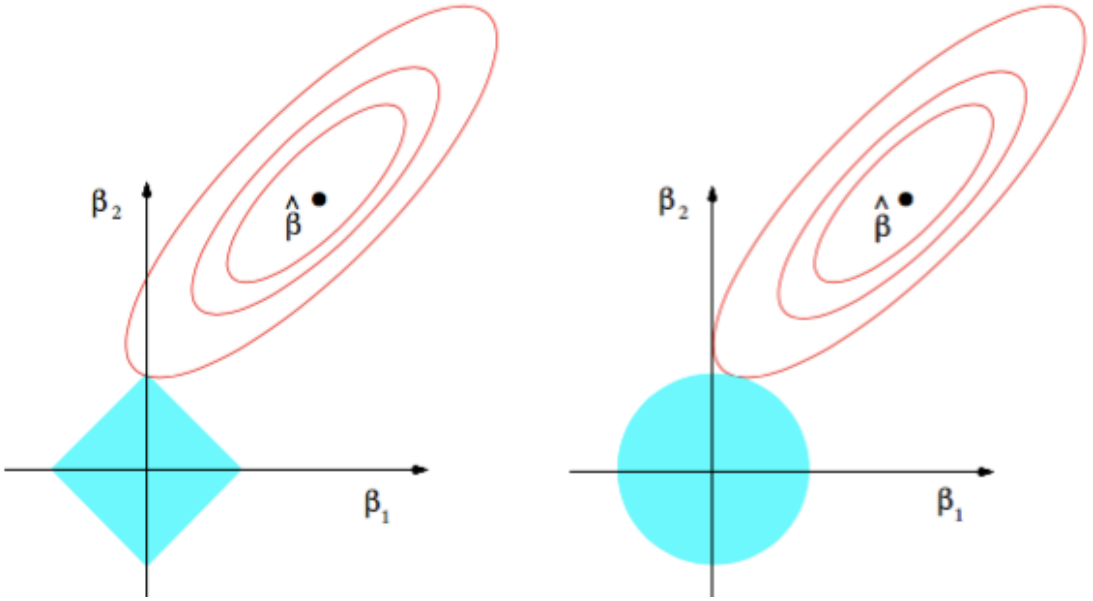
$$L1 : \|\boldsymbol{w}\|_1 = \sum_{j=1}^{m} |w_j|$$

L2와 달리 날카로운 모양이므로
최적점이 축에 가깝게 위치할 가능성이 높다.
-> 희소성이 나타남 (=가중치가 0이된다)

4.5.3 L1규제를 사용한 희소성

교재 157p

01
02
03
04



L2의 미분계수는 2 * 가중치
L1의 미분계수는 k(가중치와 무관한 값을 갖는 상수)

L2의 미분계수는 매번 가중치의 x%만큼 제거됨.
L1의 미분계수는 매번 가중치에서 일정 상수만큼 제거됨.

따라서 L1만이 가중치를 0으로 만드는 것이 가능. L2는 0에 가까워질 수 있지만 0으로 만들수는 없다.

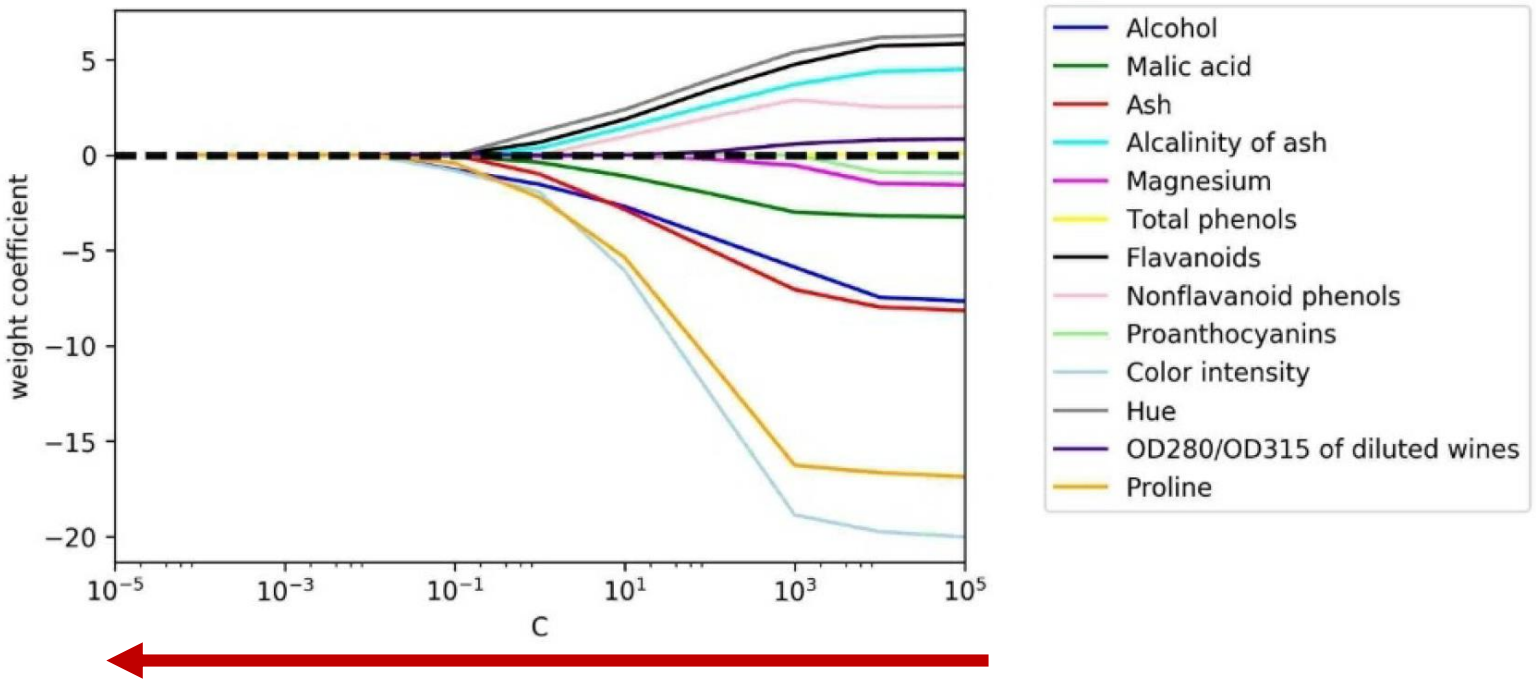
좋은 점: 필요하지 않은 특성에 대해서는 가중치를 0으로 부여하는 것이 모델의 노이즈를 줄여주고 견고한 모델을 만들어줌.

4.5.3 L1규제를 사용한 희소성

교재 157p

01
02
03
04

규제 강도에 따른 가중치 변화



규제 강도가 커질수록 모든 가중치가 0이되는 것을 확인!
따라서 규제 강도를 높여 희소성을 더 강하게 할 수 있다.

4.5.3 L1규제를 사용한 희소성 : 코드

교재 158p

01 02 03 04

```
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression(solver='liblinear', multi_class='auto',
                        penalty='l1', C=1.0, random_state=42)
# C=1.0이 기본값입니다. 값을 증가시키거나 줄여서
# 규제 효과를 크게 또는 약하게 할 수 있습니다.
lr.fit(X_train_std, y_train)
print('훈련 정확도:', lr.score(X_train_std, y_train))
print('테스트 정확도:', lr.score(X_test_std, y_test))
```

```
훈련 정확도: 1.0
테스트 정확도: 1.0
```

<- 두 데이터셋에 완벽하게 작동한다는 것을 알 수 있다.

```
lr.intercept_
```

```
array([-1.26356604, -1.21599454, -2.37043523])
```

<- 절편을 확인
: 첫 번째 절편은 클래스 1을 클래스 2·3과 구분하는 모델에 속한 것. 두 번째는 클래스 2를 클래스 1·3과 구분하는 모델의 절편. 세 번째는 클래스 3을 클래스 1·2와 구분하는 모델의 절편.

```
np.set_printoptions(8)
```

```
lr.coef_[lr.coef_!=0].shape
```

```
(23,)
```

```
lr.coef_
```

```
array([[ 1.24571379,  0.18047746,  0.74447184, -1.16206611,  0.        ,
         0.        ,  1.16544663,  0.        ,  0.        ,  0.        ,
         0.        ,  0.55203518,  2.50981212],
       [-1.53722618, -0.38723608, -0.99490375,  0.36490375, -0.05976457,
         0.        ,  0.66773248,  0.        ,  0.        , -1.9341522 ,
         1.23402387,  0.        , -2.23179839],
       [ 0.13508255,  0.16979104,  0.35787068,  0.        ,  0.        ,
         0.        , -2.43274437,  0.        ,  0.        ,  1.56234781,
        -0.81813588, -0.49713467,  0.        ]])
```

클래스마다 벡터 하나씩 세 개의 행이 있는 가중치 배열.
각 행은 13개의 가중치를 가지고, 각 가중치와 13차원의 Wine 데이터셋의 특성을 곱해 최종 입력을 계산한다.

$$z = w_1 x_1 + \cdots + w_m x_m = \sum_{j=0}^{m} x_j w_j = \boldsymbol{w}^T \boldsymbol{x}$$

4.5.4 순차 특성 선택 알고리즘

교재 161p

01 02 03 04

과대적합을 피하는 다른 방법은 특성 선택을 통한 **차원 축소**(dimensionality reduction)

특성 선택

어떤 feature가 유용한지 아닌지 확인 후,
원본 특성에서 일부를 선택

순차 후진 선택(Sequential Backward Selection, SBS)
: 초기 d 차원의 특성 공간을 k < d인 k 차원의 특성 부분 공간으로 축소.
새로운 특성의 부분 공간이 목표하는 특성 개수가 될 때까지 전체 특성에서 순차적으로 특성을 제거한다.

1. 알고리즘을 k=d로 초기화합니다. d는 전체 특성 공간 X_d의 차원입니다.
2. 조건 $x^- = \arg\max J(X_k - x)$를 최대화하는 특성 x^-를 결정합니다.
(제거했을때 성능 손실이 최대가 되는 특성)
3. 특성 집합에서 특성 x^-를 제거합니다 $X_{k-1} := X_k - x^-; k := k-1$

4. k가 목표하는 특성 개수가 되면 종료합니다. 아니면 단계 **2**로 돌아갑니다.

4.5.4 순차 특성 선택 알고리즘 : 코드

교재 161p

사이킷런에 구현되어있지 않아서 파이썬으로 직접 구현한다

01
02
03
04

```
from sklearn.base import clone
from itertools import combinations
import numpy as np
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split


class SBS():
    def __init__(self, estimator, k_features, scoring=accuracy_score,
                 test_size=0.25, random_state=1):
        self.scoring = scoring
        self.estimator = clone(estimator)
        self.k_features = k_features
        self.test_size = test_size
        self.random_state = random_state

    def fit(self, X, y):

        X_train, X_test, y_train, y_test = W
            train_test_split(X, y, test_size=self.test_size,
                             random_state=self.random_state)

        dim = X_train.shape[1]
        self.indices_ = tuple(range(dim))
        self.subsets_ = [self.indices_]
        score = self._calc_score(X_train, y_train,
                                 X_test, y_test, self.indices_)
        self.scores_ = [score]
```

데이터셋의 크기

fit 메서드의 while 루프 안에서 itertools.combination 함수에 의해 생성된 특성 조합을 평가하고 원하는 차원이 남을 때까지 특성을 줄인다.

```
    while dim > self.k_features:
        scores = []
        subsets = []

        for p in combinations(self.indices_, r=dim - 1):
            score = self._calc_score(X_train, y_train,
                                     X_test, y_test, p)
            scores.append(score)
            subsets.append(p)

        best = np.argmax(scores)
        self.indices_ = subsets[best]
        self.subsets_.append(self.indices_)
        dim -= 1

        self.scores_.append(scores[best])
    self.k_score_ = self.scores_[-1]

    return self

def transform(self, X):
    return X[:, self.indices_]

def _calc_score(self, X_train, y_train, X_test, y_test, indices):
    self.estimator.fit(X_train[:, indices], y_train)
    y_pred = self.estimator.predict(X_test[:, indices])
    score = self.scoring(y_test, y_pred)
    return score
```

d>k인 동안 루프설정

선택된 특성 열로 구성된 새로운 데이터 배열 반환

4.5.4 순차 특성 선택 알고리즘 : 코드

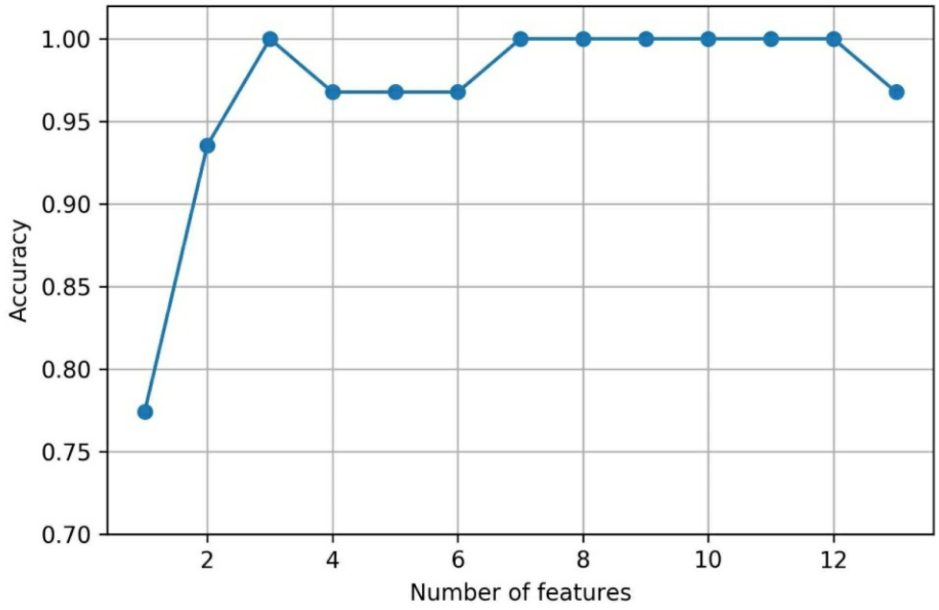
교재 161p

KNN분류기의 정확도를 알아보기.

```
01
02
03
04
```

```
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifie

knn = KNeighborsClassifier(n_neighbors=5)

# 특성을 선택합니다.
sbs = SBS(knn, k_features=1)
sbs.fit(X_train_std, y_train)

# 특성 조합의 성능 그래프를 출력합니다.
k_feat = [len(k) for k in sbs.subsets_]

plt.plot(k_feat, sbs.scores_, marker='o')
plt.ylim([0.7, 1.02])
plt.ylabel('Accuracy')
plt.xlabel('Number of features')
plt.grid()
plt.tight_layout()
plt.show()
```

각 단계에서 가장 좋은 특성 조합의 점수를 모음



K=3,7,8,9,10,11,12에서 분류기가 100%정확도 달

4.6. 랜덤 포레스트의 특성 중요도 사용

교재 167p

01

02

03

04

데이터셋에서 유용한 특성을 선택하는 또 다른 방법은 랜덤 포레스트를 사용하는 것

- 앙상블에 참여한 모든 결정트리에서 계산한 평균적인 불순도 감소로 특성 중요도를 측정할수 있다.
- Scikit-Learn에서는 어떠한 특성을 사용한 노드가 불순도(impurity)를 얼마나 감소시키는지를 계산하여 각 특성마다 상대적 중요도를 측정한다.

```
from sklearn.ensemble import RandomForestClassifier

feat_labels = df_wine.columns[1:]

forest = RandomForestClassifier(n_estimators=500,
                                random_state=1)

forest.fit(X_train, y_train)
importances = forest.feature_importances_

indices = np.argsort(importances)[::-1]

for f in range(X_train.shape[1]):
    print("%2d) %-*s %f" % (f + 1, 30,
                            feat_labels[indices[f]],
                            importances[indices[f]]))
```

500개의 트리를 가진 랜덤 포레스트를 훈련

```
 1) Proline                        0.185453
 2) Flavanoids                     0.174751
 3) Color intensity                0.143920
 4) OD280/OD315 of diluted wines   0.136162
 5) Alcohol                        0.118529
 6) Hue                            0.058739
 7) Total phenols                  0.050872
 8) Magnesium                      0.031357
 9) Malic acid                     0.025648
10) Proanthocyanins                0.025570
11) Alcalinity of ash              0.022366
12) Nonflavanoid phenols           0.013354
13) Ash                            0.013279
```

정규화된 특성 중요도

4.6. 랜덤 포레스트의 특성 중요도 사용

교재 169p

임계값을 0.1로 하여 가장 중요한 특성을 뽑아냄

```
01
02
03
04
```

```
from sklearn.feature_selection import SelectFromModel

sfm = SelectFromModel(forest, threshold=0.1, prefit=True)
X_selected = sfm.transform(X_train)
print('이 임계 조건을 만족하는 샘플의 수:', X_selected.shape[1])
```

```
이 임계 조건을 만족하는 샘플의 수: 5
```

```
for f in range(X_selected.shape[1]):
    print("%2d) %-*s %f" % (f + 1, 30,
                            feat_labels[indices[f]],
                            importances[indices[f]]))
```

```
 1) Proline                        0.185453
 2) Flavanoids                     0.174751
 3) Color intensity                0.143920
 4) OD280/OD315 of diluted wines   0.136162
 5) Alcohol                        0.118529
```