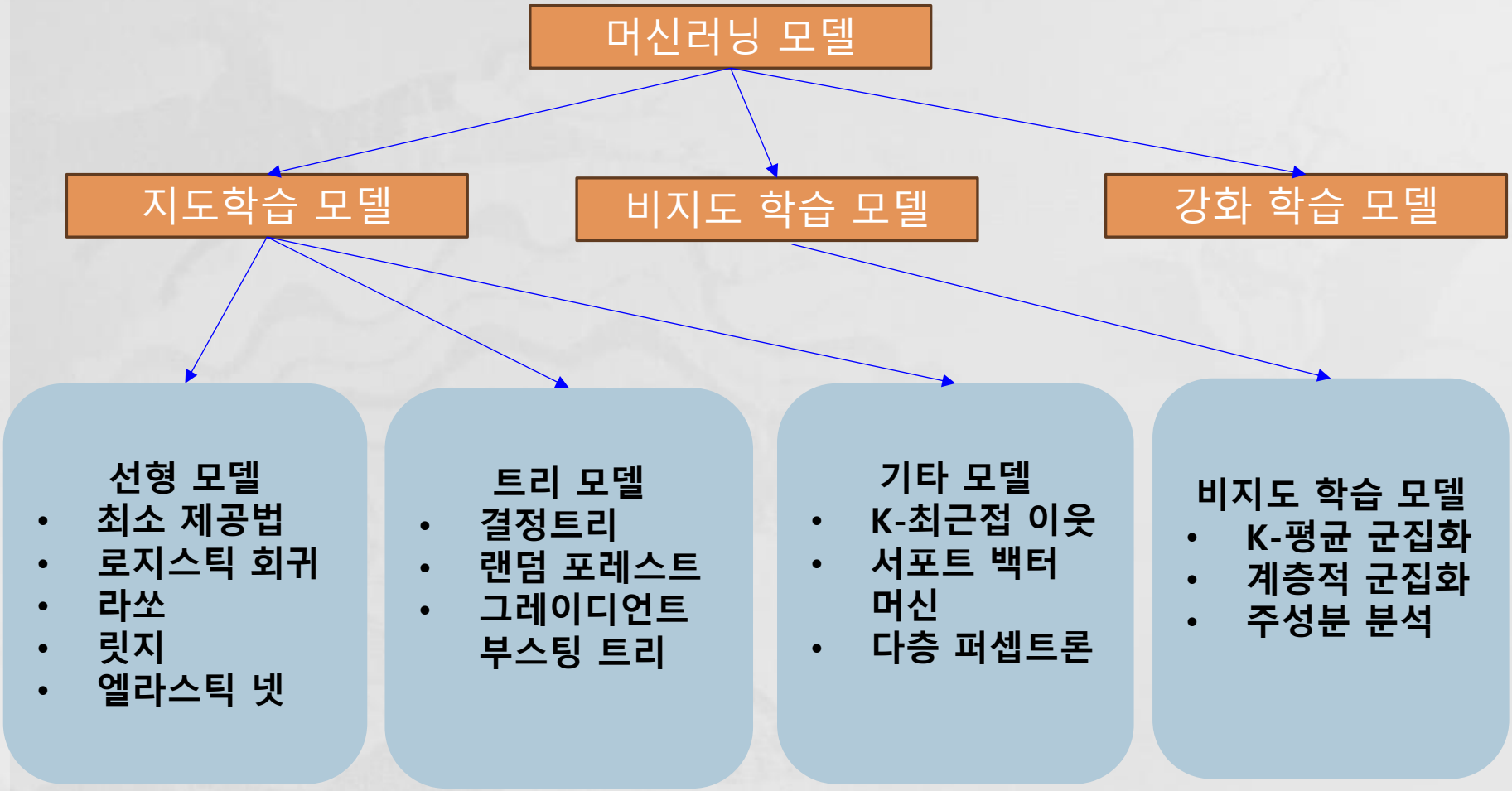
Machine Learning Model

머신러닝 모델의 계층 구조

머신러닝 모델의 계층 구조



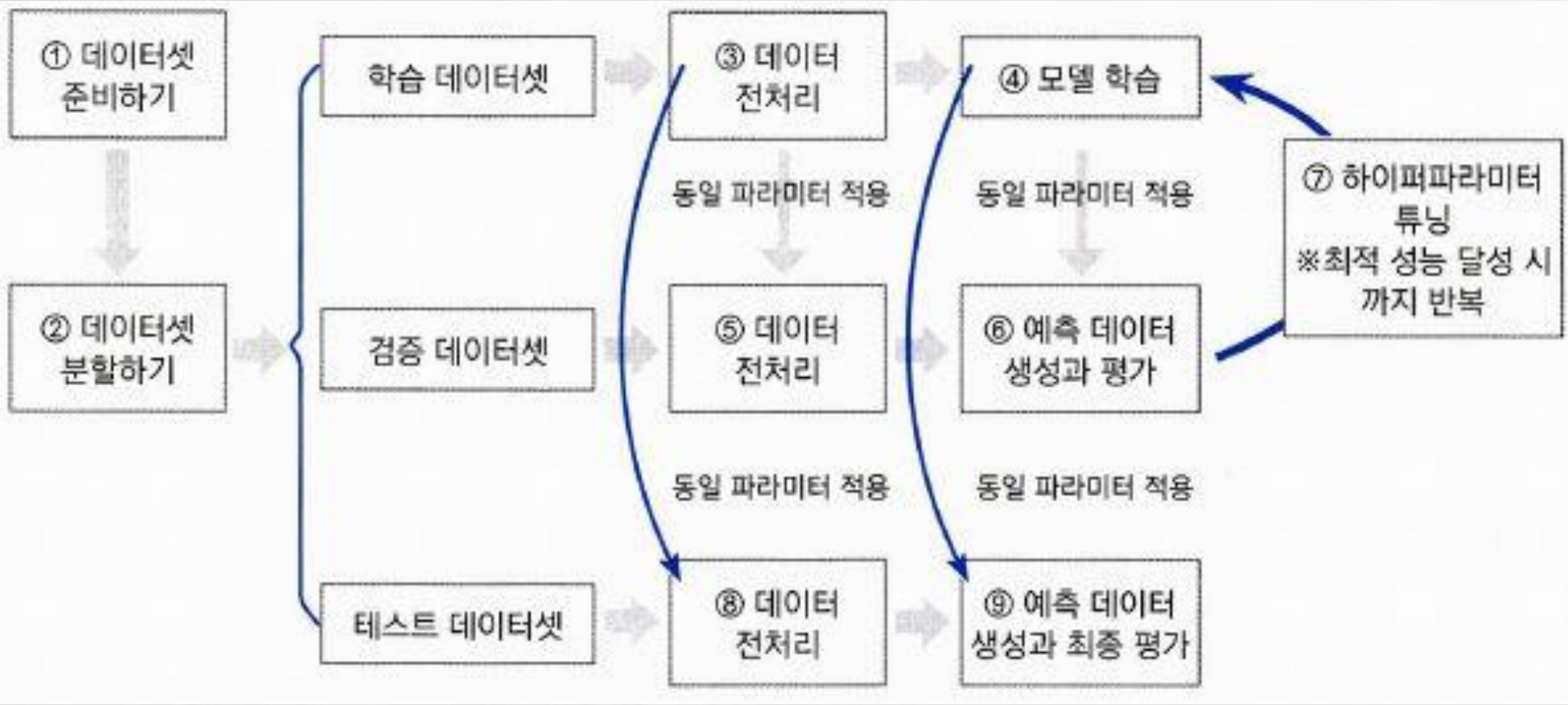
머신러닝 모델의 계층 구조

머신러닝에 쓰는 알고리즘

- 주어진 머신러닝 문제의 **최적해**를 구하는 수학적 최적화 알고리즘
- 최적화 알고리즘은 정의한 **비용 함수**를 최소하는 알고리즘,기법,접근방법
- 학습하고 구현할 때 알고리즘은 다음 요소를 종합적으로 고려
  - 시간 복잡도
    - 같은 형태의 해를 구할지라도 그 해를 계산하는 데에 걸리는 시간을 고려
  - 공간 복잡도
    - 같은 형태의 해를 구할지라도 그 해를 계산 할 때에 소요되는 메모리를 고려
  - 입력값의 작은 변동에 대한 예측값의 안정성
    - 해가 불안정하여 피처 값의 작은 변동에도 크게 변동이 없어야 함(오차항에 민감하지 않은 예측값)

머신러닝 모델의 계층 구조

머신러닝 절차



선형 모델을 이용한 지도 학습

선형 모델

- 목표값을 피처의 선행 결합으로 모델링을 하는 기법
- 최소 제곱법(ordinary least squares, OLS)
- 로지스틱 회귀 모델
- 라쏘 모델

최소 제곱법 모델

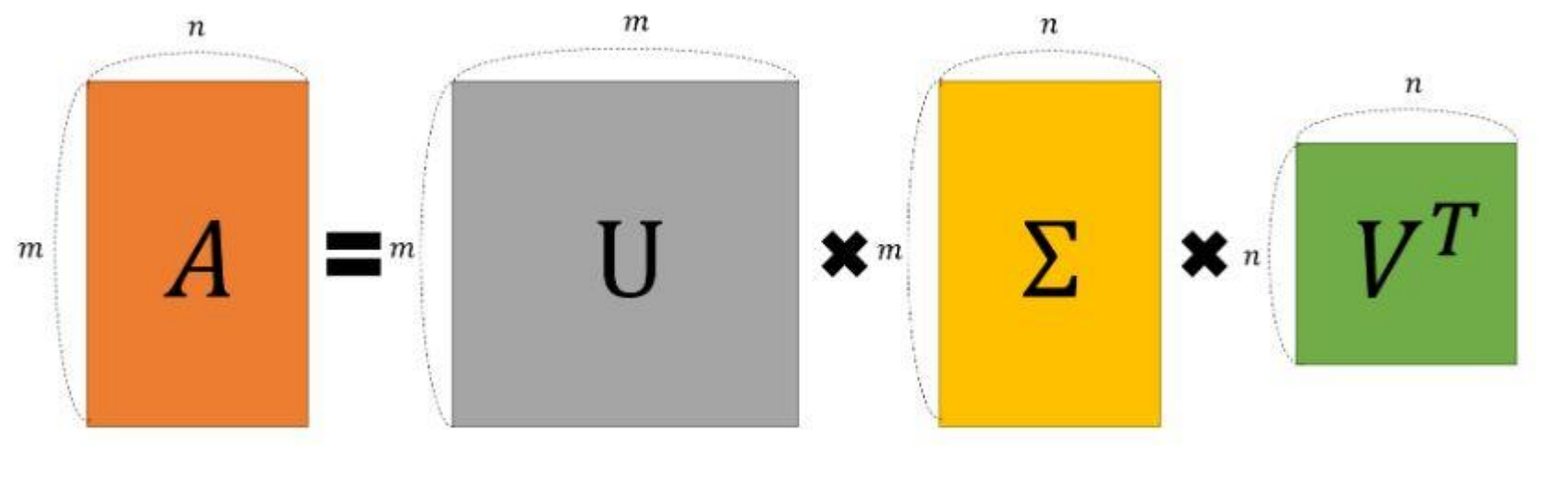
최소 제곱법 모델이란?

- 목표값의 샘플로 구성된 벡터 y와 p차원의 피처 행렬 X가 주어졌을 때 예측값을 다음과 같이 모델링
- 목푯값과 본 모델에서 도출되는 예측값을 이용해 비용 함수를 정의
- 비용 함수가 평균 제곱 오차(mean squared error, MSE)인 모델을 최소 제곱법(ordinary least squares, OLS)
- 즉, 이 모델은 예측값과 목표값의 차이 인 잔차 제곱합을 최소화하는 파라미터 w를 찾음

최소 제곱법 모델

특잇값 분해를 이용한 최소 제곱법 모델

- 특잇값 분해(singular value decomposition, SVD)는 행렬을 특수한 형태의 세 행렬의 곱으로 분해하는 것을 의미
- 주성분 분석 (principal component analysis, PCA) 등 머신러닝 분야에서 폭넓게 활용



최소 제곱법 모델

LinearRegression 클래스 사용하기

- 선형 회귀 모델인 OLS 모델을 SVD를 이용해 추정
- SVD로 OLS를 푸는 scipy.linalg.lstsqO 함수를 이용하여 구현
- LinearRegression 클래스의 주요 하이퍼파라미터

하이퍼파라미터	주요값	기본값	의미
fit_intercept	bool	True	절편 포함 여부 결정
positive	bool	False	True라면 비음수 최소 제곱법(non-negative least squares, NNLS)을 수행

- 하이퍼파라미터 positive는 NNLS 적용 여부를 결정
- NNLS는 모든 계수가 0 이상이라는 조건에서 유용
- NNLS은 scipy.optimize.nnls() 함수로 구현

최소 제곱법 모델

LinearRegression 클래스 사용하기

```
import numpy as np
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

X, y = load_diabetes(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.33,
                                                    random_state=1234)
```

```
reg = LinearRegression()
reg = reg.fit(X_train, y_train)
y_pred_train = reg.predict(X_train)

print(f'학습 데이터셋 MAE:{np.abs(y_pred_train - y_train).mean(): .3f}')

y_pred = reg.predict(X_test)
print(f'테스트 데이터셋 MAE:{np.abs(y_pred - y_test).mean(): .3f}')
```

```
학습 데이터셋 MAE: 43.549
테스트 데이터셋 MAE: 42.618
```

로지스틱 회귀 모델

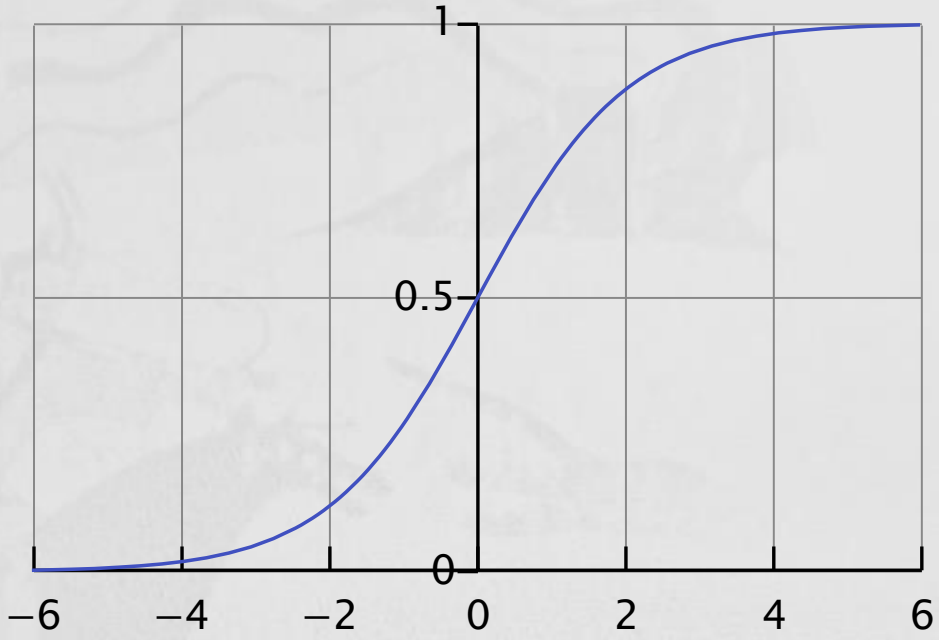
최대하강법

- 주어 진 비용 함수의 지역 최소값이나 전역 최소값을 구하는 것은 수학적 최적하 과정 중 하나임
- 경사 하강법은 이 과정을 수행하는 방법의 하나
- 넓은 의미에서 경사 하강법은 경사 혹은 그레이디언트(gradient)의 역방향으로 입력값을 차례대로 이동하며 최소의 목표값을 달성하는 모든 방법을 의미
- 최대하강법, 뉴턴법, BFGS등 다양한 알고리즘

로지스틱 회귀 모델

로지스틱 회귀 모델이란?

- 로지스틱 회귀는 분류 문제를 해결하는 가장 기본적인 머신러닝 모델
- 로지스틱 회귀라는 이름에도 불구하고 회귀가 아닌 분류에 사용
- 로지스틱 회귀는 목표 클래스의 발생 확률을 로지스틱 함수로 모델링



로지스틱 회귀 모델

LogisticRegression 클래스 사용하기

- 이진 분류binary classification와 다중 클래스 분류를 모두 수행
- L1, L2, 엘라스틱 넷 규제 또한 적용할 수 있음
- solver 하이퍼파라미터에는 비용함수 최적화 알고리즘으로 'liblinear', ' newton-cg', ' lbfgs', 'sag', ' saga' 중하나를 선택

하이퍼파라미터	의미	특징
'newton-cg'	켤레기울기법conjugate gradient method	희소 데이터셋sparse dataset에서 추천한다.
'lbfgs'	L-BFGS-B 알고리즘	작은 크기의 데이터셋에 추천하며, 데이터의 크기가 크다면 성능이 떨어질 수 있다.
'liblinear'	최적화된 좌표 하강법coordinate descent 알고리즘	좌표 하강법은 다루도록 한다. 데이터의 크기가 작을 때 추천한다.
'sag'	SAG(stochastic average gradient descent) 알고리즘	샘플 개수와 피처 개수가 모두 큰 대형 데이터셋에서 빠르게 수렴한다. 하지만 빠른 수렴은 각 피처의 스케일이 비슷할 때만 보장되므로 sklearn.preprocessing 모듈의 클래스를 사용해 데이터를 전처리하는 것이 좋다.
'saga'	SAG 알고리즘의 변형	'sag'와 마찬가지로 대형 데이터셋에서 잘 동작하고, 피처 스케일에 대한 정규화를 선행하는 것이 좋다.

로지스틱 회귀 모델

LogisticRegression 클래스 사용하기

- LogisticRegression 클래스에는 다음과 같은 주요 하이퍼파라미터

하이퍼파라미터	주요값	기본값	의미
penalty	'l1', 'l2', 'elasticnet', 'none'	'l2'	규제 페널티 선택 - 'l1': L1 규제 적용 - 'l2': L2 규제 적용 - 'elasticnet': 엘라스틱 넷 규제 적용 - 'none': 규제 미부여
tol	float>0	1e-4	학습 종료에 대한 허용 오차 설정
C	float>0	1.0	규제 페널티 크기의 역수
fit_intercept	bool	True	절편 항 포함 여부를 결정
class_weight	None, dict, 'balanced'	None	클래스별 가중치 부여 방법 - None: 동일 가중치 - dict: 사용자 지정 가중치 - 'balanced': 클래스 빈도frequency에 반비례한 가중치
random_state	int	None	solver가 'sag', 'saga', 'liblinear'일 때 데이터를 섞게 되는데, 이에 대한 랜덤성을 제어하고자 사용

로지스틱 회귀 모델

LogisticRegression 클래스 사용하기

- LogisticRegression 클래스에는 다음과 같은 주요 하이퍼파라미터

solver	'newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'	'lbfgs'	최적화 알고리즘의 선택. 강건성robustness 때문에 'lbfgs'가 기본 solver로 주어진다. 각 solver의 의미는 설명한 바와 같다.
max_iter	int>0	100	이터레이션iteration 횟수의 상한선
multi_class	'auto', 'ovr', 'multinomial'	'auto'	다중 클래스 분류 방법을 선택. 클래스가 2개라면 옵션에 따른 결과 차이는 없다. - 'ovr': OVR 분류 수행 - 'multinomial': 크로스 엔트로피 비용 함수 사용 - 'auto': solver가 'liblinear'일 때만 'ovr'을 선택하고 나머지는 'multinomial'을 선택
l1_ratio	None, float	None	엘라스틱 넷 규제가 포함된 비용 함수에서 ρ에 해당함. penalty가 'elasticnet'일 때만 유효하다.

로지스틱 회귀 모델

LogisticRegression 클래스 사용하기

- LogisticRegression 클래스에는 다음과 같은 주요 하이퍼파라미터

```
import numpy as np
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

X, y = load_breast_cancer(return_X_y=True, as_frame=True)
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.33,
                                                    random_state=1234)

X_train = X_train.iloc[:, :3]
X_test = X_test.iloc[:, :3]
```

```
clf = LogisticRegression(random_state=1234, C=100000)

clf = clf.fit(X_train, y_train)
y_train_pred = clf.predict(X_train)

y_pred = clf.predict(X_test)

print(f'학습 데이터셋 정확도:{(y_train == y_train_pred).sum() / len(y_train) * 100: .2f}%')
print(f'테스트 데이터셋 정확도:{(y_test == y_pred).sum() / len(y_test) * 100: .2f}%')
```

```
학습 데이터셋 정확도: 93.18%
테스트 데이터셋 정확도: 87.23%
```

피처 스케일링이 필요한 모델

피처 스케일링이 필요한 모델

- 거리를 기반으로 동작하거나 비용 함수가 피처의 범위에 크게 영향을 받는다면 모델보다 피처 스케일링을 먼저 진행하는 것이 필수
- OLS, 트리, 랜덤 포레스트 모델 등은 일반적으로 스케일링 여부에 예측력이 영향을 받지 않음
- 로지스틱 회귀, 라쏘, 릿지 등 비용 함수에 규제항이 포함된 모델은 스케일링에 따라 예측력이 달라짐
- 이는 규제항을 포함하면 계수 크기의 L1 혹은 L2 합 전체를 규제하게 되므로 비용 함수가 큰 계수의 값에 영향을 받기 때문

피처 선택법

과적합을 제어

- 머신러닝 모델의 성능을 향상시키는 데 가장 중요한 문제의 하나는 모델에 적절한 규제를 적용하여 학습 데이터에만 나타나는 지엽적인 규칙을 학습하는 과적합 현상을 방지하는 것
- 선형 모델, 트리 모델, 딥러닝 모델 등 현대에 사용하는 대부분의 모델은 과적합을 규제하는 장치
- 라쏘 모델은 피처 선택법이라는 유용한 성질을 이용하여 과적합을 제어

피처 선택법

피처 선택법

- 주어진 데이터의 피처 개수가 많아 시간, 공간 복잡도와 해의 안정성 면에서 모델에 모든 피처를 포함하는 것이 부적절하다면 정형화된 방법을 통해 일부 피처를 예측 모델에서 제외 할 수 있음
- 이를 피처 선택법(feature selection)이라 함
- 필터 기반(filter-based), 래퍼 기반(wrapped-based), 임베디드(emb 등 세 종류로 구분

피처 선택법

필터 기반 피처 선택법

- 단일 피처를 대상으로 수행(sklearn.feature_selection.SelectKBest 클래스)
- 분석에 좋은 피처만 포함하고자 기준을 정하고, 그 기준을 만족하는 피처만 모델에 포함
- 필터 기반 피처 선택법의 종류는 목표 변수가 수치형 변수인지 범주형 변수인지에 따라(즉, 문제가 회귀 문제인지 분류 문제인지에 따라), 그리고 대상 피처가수치형 변수인지 범주형 변수인지에 따라 달라짐
  - 목표 변수와 대상 피처가 모두 수치형 이라면 목표 변수와의 상관 계수 절댓값이 특정 값(예: 0.7) 미만인 변수는 모델에서 제외하는 상관 계수 기반 필터를 사용(f_regression())
  - 목표 변수가 범주형 변수라면 카이제곱 통계량을 기준으로 목표 변수와의 종속성이 가장 큰 변수를 선정하는 카이제곱 기반 필터를 사용(chi2(), f_classif(), mutual_info_classif())

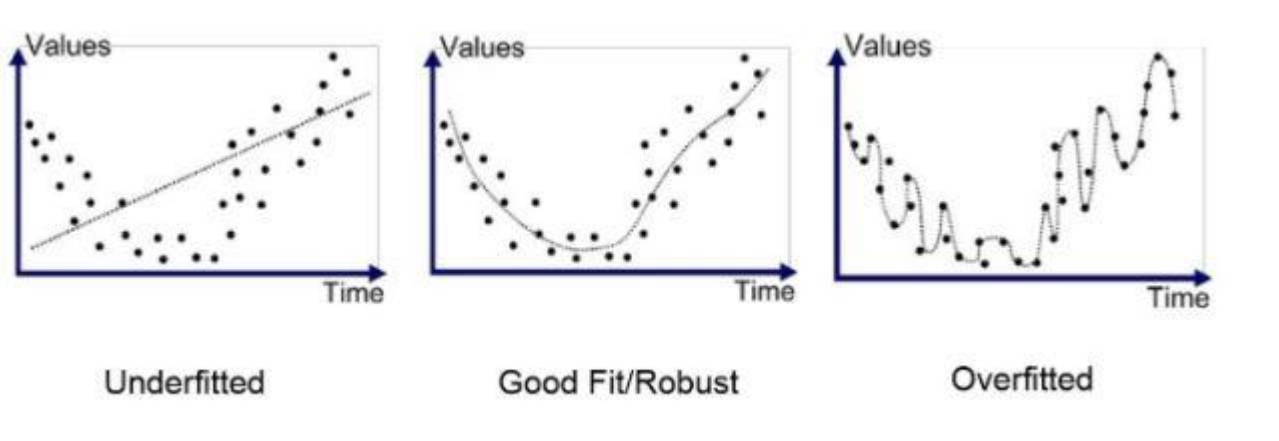
피처 선택법

래퍼 기반 피처 선택법

- 피처 선택을 적절한 피처 집합을 찾아내는 탐색 문제로 간주
- 그리디 알고리즘을 이용하여 지역 최적 조합을 순서대로 탐색
- 선택법에는 주어진 조건하에서 이터레이션 별로 피처를 추가하면서 모델링을 반복하는 전진 선택법
- 전체 피처 집합에서 피처를 줄이면서 모델링을 반복하는 후진 소거법
- sklearn.feature_selection.RFE 클래스
- sklearn.feature_selection.SequentialFeatureSelector 클래스

underfitted과 overfitting

underfitted과 overfitting



- 데이터와 직선의 차이가 아는 경우 underfitted 또는 high bias
- bias가 큰 모델은 test data를 위한 학습이 덜 된 것이 원인이고, 이는 train data와 test data 간의 차이가 너무 커서 train data로만 학습한 모델은 test data를 맞출 수가 없는 것
- 현재 데이터로는 잘 맞겠지만 다른 데이터를 사용한다면 정확한 예측을 하지 못하는 경우 overfitting 또는 high variance
- variance가 큰 모델은 train data에 over-fitting된 것이 원인이고, 이는 너무 train data에 fitting된 모델을 만들어서 test data에서 오차가 발생한 것을 의미

라쏘 모델

라쏘 모델이란?

- 라쏘 모델은 희소 데이터셋에서 파라미터를 추정하는 선형 모델인 동시에 예측에 사용하는 변수의 숫자를 줄이는 피처 선택법의 일종
- 라쏘 모델은 피처 수를 줄여야 하는 상황에서 매우 유용하므로 라쏘나 그 변형 기법은 압축 센싱분야를 포함하여 필요한 피처를 추축
- 라쏘 비용 함수를 하방미분과 좌표 하강법으로 최적화하는 방법
- sklearn.linear_model.Lasso 클래스는 좌표 하강법을 통해 라쏘 비용 함수를 최적화하여 파라미터를 학습

라쏘 모델

Lasso 클래스

- Lasso 클래스에는 다음과 같은 주요 하이퍼파라미터

하이퍼파라미터	주요값	기본값	의미
alpha	float>0	1.0	L1 규제 페널티 계수로, 비용 함수에서의 α에 해당. alpha가 0에 가까워질수록 OLS 해에 수렴한다.
fit_intercept	bool	True	절편 포함 여부 결정 - True: 절편 포함
max_iter	int>0	1000	이터레이션 횟수의 상한선
tol	float>0	1e-4	조기 종료에 대한 허용 오차. 이터레이션에 따른 업데이트의 값이 tol보다 작을 때 추가 조건을 확인하고 조기 종료를 결정한다.
positive	bool	False	파라미터 제약 조건 - True: 모든 파라미터가 0 이상의 값을 가지도록 강제
selection	'cyclic', 'random'	'cyclic'	계수의 업데이트 방식 - 'cyclic': 정해진 순으로 계수를 업데이트 - 'random': 매 이터레이션에서 계수를 랜덤한 순서로 업데이트. 이 옵션을 선택하면 특정 조건에서 수렴 속도가 매우 빨라질 수 있다.
random_state	None, int	None	selection이 'random'일 때 랜덤성을 제어하고자 사용

라쏘 모델

Lasso 클래스

- LogisticRegression 클래스와 마찬가지로 비용 함수에 규제항이 포함되므로 특정 피처 범위 때문에 비용 함수가 큰 영향을 받지 않으려면 피처 스케일 링을 선행
- 규제 페널티의 계수 하이퍼파라미터 alpha의 값에 따라서 선택되는 피처와 학습 결과가 크게 영향을 받으므로 적절한 alpha 설정이 가장 중요

라쏘 모델

Lasso 클래스

```
scaler = StandardScaler()
reg = Lasso(alpha=10, random_state=1234)

pipe = Pipeline(steps=[("scaler", scaler), ("reg", reg)])
pipe = pipe.fit(X_train, y_train)

y_train_pred = pipe.predict(X_train)
y_pred = pipe.predict(X_test)

print(f'학습 데이터셋 MAE:{np.abs(y_train_pred - y_train).mean(): .3f}')
print(f'테스트 데이터셋 MAE:{np.abs(y_pred - y_test).mean(): .3f}')
```

```
학습 데이터셋 MAE: 47.475
테스트 데이터셋 MAE: 46.139
```

```
print('추정 파라미터의 값:₩n', reg.coef_)
```

```
추정 파라미터의 값:
 [ 0.          -0.          22.14741042  7.25438995 -0.          -0.
 -4.43633143  0.          18.8046418   0.         ]
```

릿지 회귀 모델

릿지 회귀 모델이란

- L2 규제가 있는 선형 최소 제곱법, 티호노프 규제 기법 등으로도 불리는

  릿지 회귀는 OLS이의 비용 함수에 파라미터 크기에 대한 L2 페널티를 추

  가하여 비용 함수를 정의한후 이를 최적화하여 얻은 모델

$$\sum_{i=1}^{n}\left(y_i-\beta_0-\sum_{j=1}^{p}\beta_j x_{ij}\right)^2+\lambda\sum_{j=1}^{p}\beta_j^2=\mathrm{RSS}+\lambda\sum_{j=1}^{p}\beta_j^2$$

릿지 회귀 모델

sklearn. linear_model.Ridge 클래스

- Ridge 클래스에는 다음과 같은 하이퍼파라미터

하이퍼파라미터	주요값	기본값	의미
alpha	float>=0	1.0	규제 강도를 뜻하며 alpha가 클수록 규제가 크게 작용한다.
fit_intercept	Bool	True	절편 포함 여부 결정 - True: 절편 포함
max_iter	int>0	None	이터레이션 횟수의 상한선 - None: solver에 따라 다른 기본값 부여
tol	float>0	1e-3	해의 정밀성precision. 해를 얼마나 정밀하게 구할 것인지를 결정한다.
solver	'auto', 'svd', 'cholesky', 'lsqr', 'sparse_cg', 'sag', 'saga', 'lbfgs'	'auto'	최적화 기법 선택 - 'auto': 데이터에 따라 이하 옵션 중 자동 선택 - 'svd': X에 SVD를 적용한 후 파라미터를 추정. 피처가 특이 행렬에 가깝다면 'cholesky'에 비해 더 안정적인 결과를 도출한다. - 'cholesky': 앞의 이론과 구현 부분에서 사용한 기법. scipy.linalg.solve() 함수를 사용하며, 이는 숄레스키 분해Cholesky decomposition를 통해 역행렬을 구한 후 닫힌 형태의 해를 구하는 것과 같다. - 'sparse_cg': scipy.sparse.linalg.cg() 함수로 구현한 켤레기울기법conjugate gradient method 사용. 대형 데이터셋에서는 'cholesky'보다 더 적합하다. - 'lsqr': scipy.sparse.linalg.lsqr() 함수로 구현한 최소 제곱법 사용. 희소 행렬에 적용하면 속도가 매우 빠르다. 이터레이션iteration 기반으로 수행한다. - 'sag': SAG 알고리즘 - 'saga': SAGA 알고리즘 - 'lbfgs': L-BFGS-B 알고리즘

릿지 회귀 모델

sklearn. linear_model.Ridge 클래스

- 특정 피처 범위로 말미암아 비용 함수가 큰 영향을 받는 것을 피하려면 피처 스케일링을 먼저 수행
- 라쏘와 마찬가지로 릿지 회귀에서도 규제 페널티 alpha값은 모든 파라미터의 크기와 모델 성능에 큰 영향을 끼치므로 적절한 alpha값을 선택하는 것이 중요
- 검증 데이터로 다양한 alpha 값을 테스트한 후 최적의 alpha 값을 선택하거나 sklearn.linearjnodel.RidgeCV클래스를 이용하여 교차검증법 사용

릿지 회귀 모델

sklearn. linear_model.Ridge 클래스

```
from sklearn.linear_model import Ridge
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split
import numpy as np

X, y = load_diabetes(return_X_y=True, as_frame=True)
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.33,
                                                    random_state=1234)
```

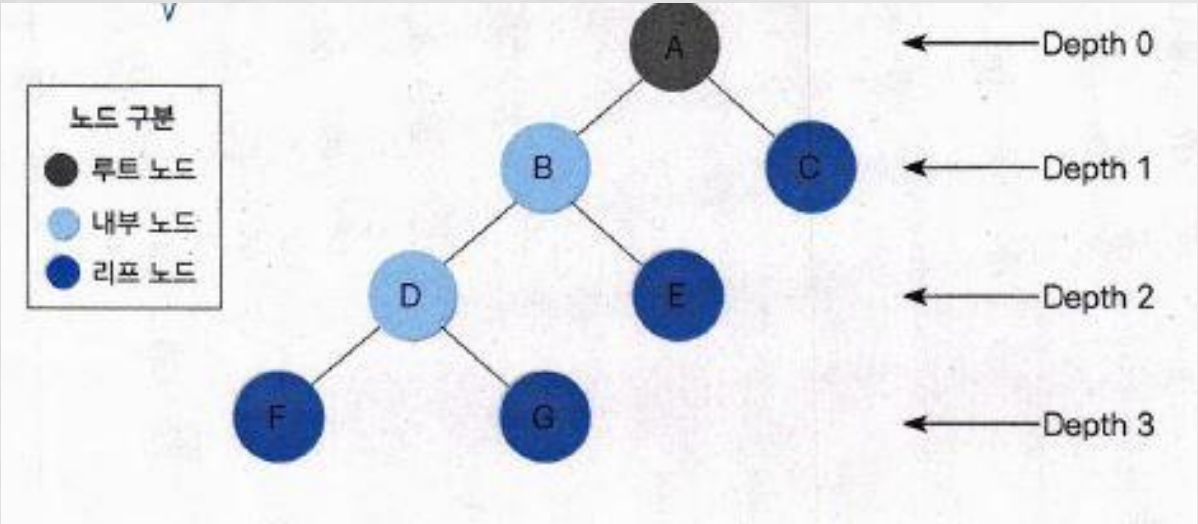
```
alphas = [0, 0.1, 1]

for alpha in alphas:
    reg = Ridge(alpha=alpha)
    reg = reg.fit(X_train, y_train)

    y_pred_train = reg.predict(X_train)
    print(f'alpha 값이 {alpha}일 경우:')
    print(f'학습 데이터셋 MAE:{np.abs(y_pred_train - y_train).mean(): .3f}')
    y_pred = reg.predict(X_test)
    print(f'테스트 데이터셋 MAE:{np.abs(y_pred - y_test).mean(): .3f}₩n')
```

```
alpha 값이 0일 경우:
학습 데이터셋 MAE: 43.549
테스트 데이터셋 MAE: 42.618

alpha 값이 0.1일 경우:
학습 데이터셋 MAE: 43.928
테스트 데이터셋 MAE: 43.366

alpha 값이 1일 경우:
학습 데이터셋 MAE: 48.966
테스트 데이터셋 MAE: 49.582
```

결정 트리모델

트리 자료 구조

- 트리 자료 구조는 노드가 나뭇가지 처럼 연결된 계층 자료 구조
- 나무를 뒤집은 모양과 비슷
- 트리 안에는 하위 트리가 있고 그 하위 트리 안에 다시 하위 트리가 있는 재귀적인 자료 구조



결정 트리모델

트리 자료 구조

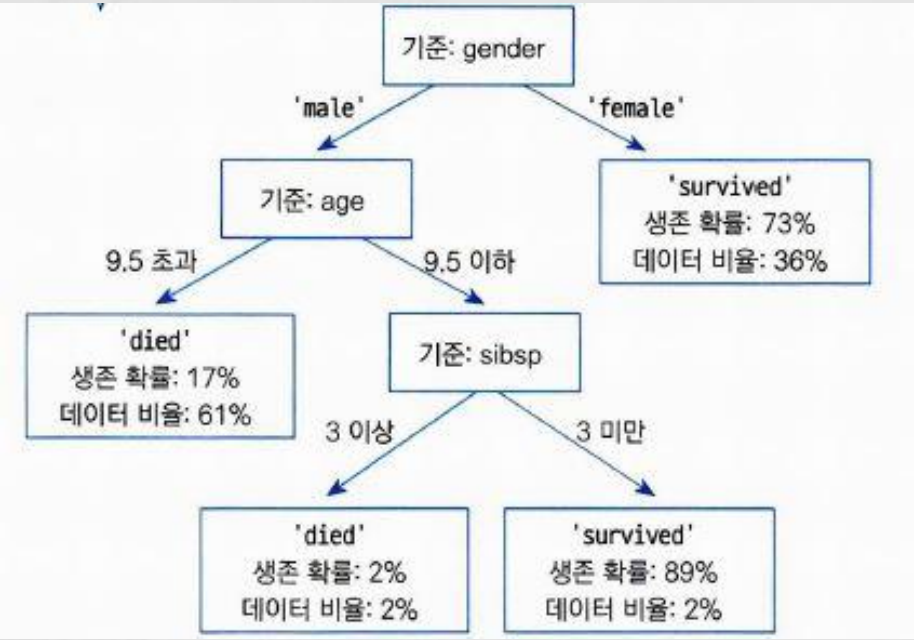
- 트리 자료 구조에서 사용하는 용어

노드	트리를 구성하는 기본 요소. 각 노드에는 자기 자신에 대한 정보와 하위 노드에 대한 연결 정보가 있다.	A, B, C, D, E, F, G
루트 노드 root node	트리 구조에서 부모가 없는 최상위 노드	A
부모 노드 parent node	자식 노드가 있는 노드	D의 부모 노드는 B
자식 노드 child node	부모 노드의 하위 노드	D의 자식 노드는 F, G
형제 노드 sibling node	부모가 같은 노드	F, G는 모두 부모가 D인 형제 노드
리프 노드 leaf node	자식 노드가 없는 노드	C, E, F, G
내부 노드 internal node	자식 노드가 있는 노드	A, B, D
깊이 depth	루트에서 특정 노드까지의 간선 수	루트 노드 A의 깊이: 0 D의 깊이: 2
최대 깊이 max depth	트리에 있는 모든 깊이 중 최댓값	3
오더 order	부모 노드가 가질 수 있는 최대 자식의 수	오더 2인 트리라면 부모 노드는 최대 2명의 자식을 가짐

결정 트리모델

결정 트리 모델이란?

- 결정 트리 입력값과 출력값의 관계를 트리 자료 구조와 같이 뿌리에서 잎까지 분기하며 이어지는 나무 모양으로 모델링하는 지도 학습 기법
- 분류 문제를 푸는 분류 트리와 회귀 문제를 푸는 회귀 트리로 구분



결정 트리모델

sklearn.tree.DecisionTreeClassifier 클래스

- 범주형 피처를 입력 변수로 받지 않으므로 범주형 입력 변수가 필요할 때는 더미 변수로 전환한 다음 모델에 포함

- DecisiontreeClassifier 클래스의 주요 하이퍼파라미터

하이퍼파라미터	주요값	기본값	의미
criterion	'gini', 'entropy'	'gini'	분기 규칙 선택 - 'gini': 지니 불순도 최소화, 'entropy': 정보 이득 최대화
max_depth	None, int>0	None	깊이의 상한선 - None: 최대 깊이를 제한하지 않음. 설정 시 과적합에 조심해야 한다.
min_samples_split	int>0, float>0	2	노드에서 분기를 진행하는 최소한의 샘플 숫자 선택 - int: min_samples_split 사용 - float: n과 min_samples_split의 곱을 계산 후 이를 올림한 정수값을 사용
min_samples_leaf	int>0, 0<float<1	1	리프 노드에 있을 샘플 개수의 최솟값을 선택. 과적합을 막고자 사용하며, 나중에 설명할 회귀 트리에서 더욱 유용하다. - int: min_samples_leaf 사용 - float: n과 min_samples_leaf의 곱을 계산 후 이를 올림한 정수값을 사용

결정 트리모델

sklearn.tree.DecisionTreeClassifier 클래스

- DecisiontreeClassifier 클래스의 주요 하이퍼파라미터

max_features	None, $1 \le int \le p$, $0 < float \le 1$, 'sqrt', 'log2'	None	각 노드에서 분기를 위해 확인할 피처 수. 하지만 단 하나의 분기 규칙도 못 찾았을 때는 지정한 수를 넘어서도 계속 확인한다는 점에 유의하도록 한다. - None: p개 전체 확인 - int: max_features개 피처 확인 - float: p와 max_features의 곱을 계산 후 이를 버림한 값만큼의 피처를 확인 - 'sqrt': $\sqrt{p}$개의 피처 확인 - 'log2': $\log_2 p$개 피처 확인
random_state	int	None	랜덤성 제어. 결정 트리에 랜덤성이 필요한 이유는 다음과 같다. - max_features < p일 때 알고리즘은 max_features 수에 해당하는 피처를 랜덤하게 선택한다. - max_features == p일 때도 지니 불순도나 정보 이득이 같아지는 피처가 두 개 이상 발생할 수 있는데, 이때 선택되는 피처는 랜덤하다.
class_weight	None, dict, 'balanced'	None	클래스 레이블별 가중치 설정 - None: 모든 클래스에 같은 가중치 설정 - dict: 사용자 설정 가중치 설정 - 'balanced': 클래스의 비율에 반비례한 가중치 설정
ccp_alpha	$float \ge 0$	0	Breiman et al.(1984)의 가지치기 알고리즘에서 사용하는 복잡도 하이퍼파라미터. 과적합을 방지하기 위한 가지치기의 개념으로 사용한다. 0이라면 가지치기를 수행하지 않는다.

결정 트리모델

sklearn.tree.DecisionTreeClassifier 클래스

- 결정 트리는 과적합되기 쉬우므로 과도하게 많은 피처를 학습에 사용하지 않도록 유의
- 트리의 크기를 제어하는 max_depth를 기본값 그대로 둘 경우 트리의 크기가 매우 커질 수 있음
- 이는 과적합과 메모리 낭비를 일으킬 수 있으므로 데이터셋이 너무 크다면 이들 하이퍼파라미터를 적절히 선택하여 트리 크기와 복잡도를 제한
- min_samples_split나 min_samples_leaf를 낮게 설정할수록 과적합 가능성이 커지므로 선택할 때 조심

결정 트리모델

sklearn.tree.DecisionTreeClassifier 클래스

```
import pandas as pd
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier

X, y = load_iris(return_X_y=True, as_frame=False)
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.33,
                                                    random_state=1234)
```

```
clf = DecisionTreeClassifier(criterion='entropy',
                             max_depth=3,
                             min_samples_split=3,
                             random_state=1234)
clf = clf.fit(X_train, y_train)
```

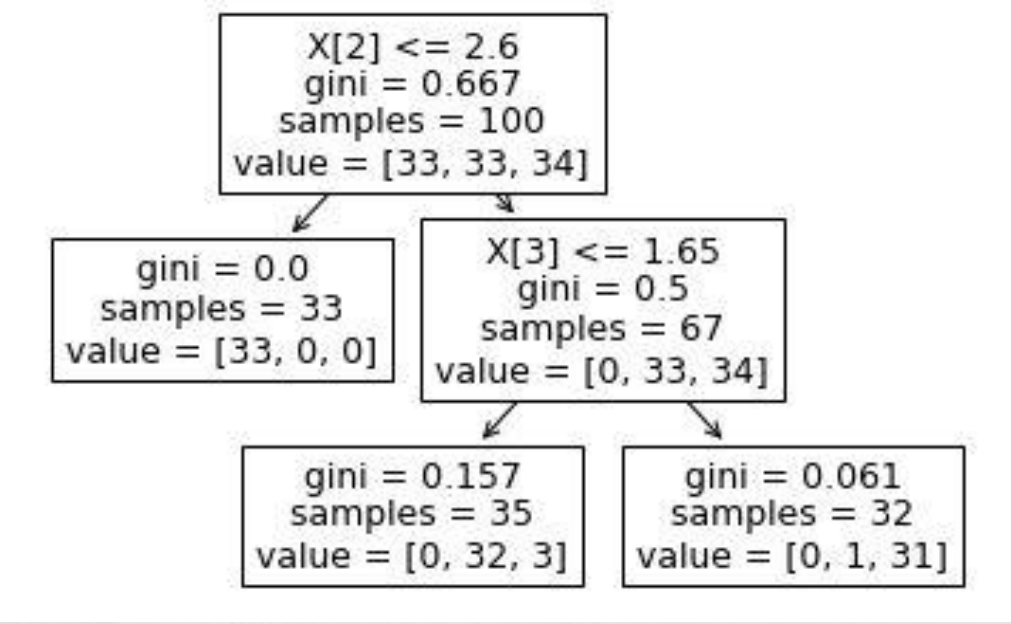
```
y_pred = clf.predict(X_test)
acc = (y_pred == y_test).sum() / len(y_test)
print(f'정확도:{acc * 100: .2f}%')
```

```
정확도: 98.00%
```

결정 트리모델

sklearn.tree.DecisionTreeClassifier 클래스

- 노드는 루트 노드를 포함하여 5개가 형성되었으며, 최대 깊이는 2
- 입력 샘플이 주어졌을 때 루트 노드에서 2번 피처0부터 카운트) 의 값을 기준으로 2.6 이하라면 왼쪽 자식 노드로, 그 외는 오른쪽 자식 노드



랜덤 포레스트 모델

앙상블 학습 기법

- 앙상블 기법은 여러 개의 베이스 학습기를 준비하여 학습을 수행한 후 이들 학습 결과를 종합하여 최종 결과를 예측하는 기법
- 앙상블 기법에는 배킹(bootstrap aggregating)과 부스팅(boosting)
- 배깅은 주어진 데이터셋으로 수많은 부트스트랩 샘플을 생성한 후 이에 각각 독립적인 다수의 학습기를 만들어 병렬로 학습
- 예측 단계에서는 각각의 학습기로 얻은 예측값을 대상으로 평균이나 투표 등의 방식을 적용하여 최종 예측값을 산출

랜덤 포레스트 모델

앙상블 학습 기법

- 앙상블 모델은 단일 학습기에 비해 분산variance이 감소하는 효과가 있으므로 베이스 학습기가 낮은 편향과 높은 분산을 보일 때 유용
- 베이스학습기의 예로는 최대 깊이의 크기가 큰 결정 트리, 다층 퍼셉트(multilayer perceptron, MLP)론서포트 벡터 머신(support vector machine, SVM), K가 작은 K-최근접 이웃(K-nearest neighbors, KNN) 모델

랜덤 포레스트 모델

앙상블 학습 기법

- 부스팅은 여러 베이스 학습기를 만든 후 이를 순차적으로 학습하는 직렬 앙상블 모델
- 부스팅 모델은 각각의 베이스 학습기가 잘 학습하지 못하는 부분을 뒤의 베이스 학습기로 넘겨서 순서대로 편향을 줄이는 것을 목표
- 이에 따라 분산이 낮고 편향이 상대적으로 높은 베이스 학습기에 적용하면 유용
- 베이스 학습기의 예로는 최대 깊이의 크기가 작은 결정 트리, 로지스틱 회귀 모델, K가 큰 KNN 모델
- 부스팅 모델의 예시는 대표적으로 에이다부스트(AdaBoost), 그레이디언트 부스팅 트리 (gradient hosting trees, GBT) 등

랜덤 포레스트 모델

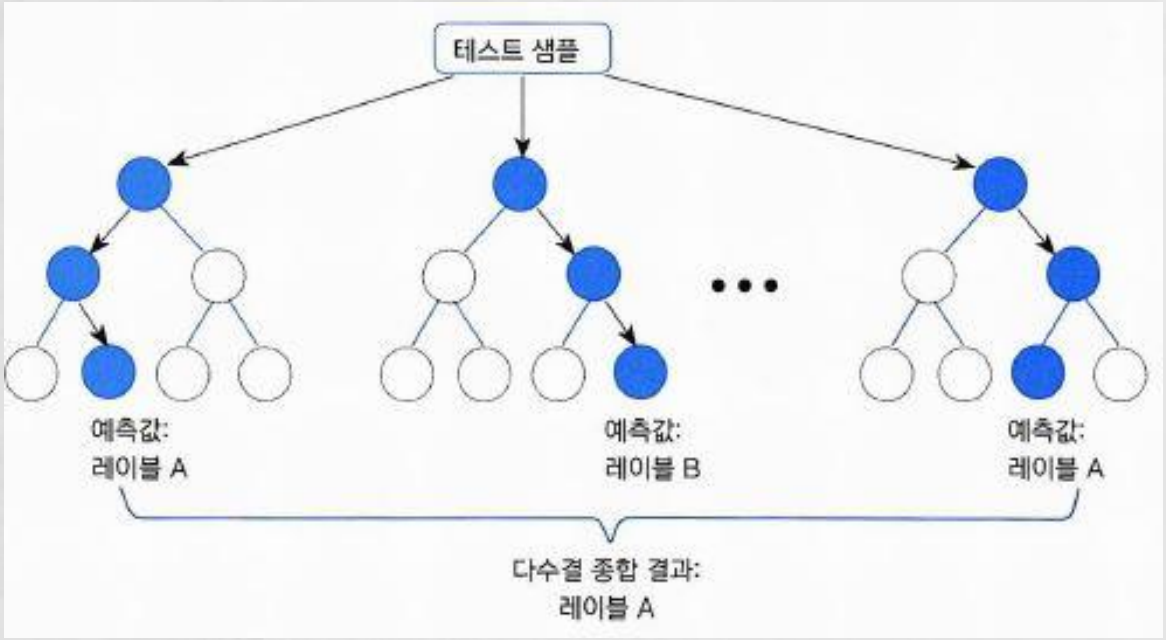
랜덤 포레스트 모델이란

- 여러 개의 결정 트리를 학습하고 그 결과를 종합하는 앙상블 학습 모델의 하나
- 비지도 학습에 적용할 수도 있으나 주로 분류, 회귀 등의 지도 학습
- 분류 문제에서는 각 결정 트리의 예측 클래스 레이블을 투표한 후 최다 득표 레이블을 선택
- 회귀 문제에서는 각 결정 트리 예측값의 평균을 예측값

랜덤 포레스트 모델

랜덤 포레스트 모델이란

- 학습된 랜덤 포레스트 이진 분류 모델에 테스트 샘플이 입력되었을 때 트리 별로 예측값이 얻어진 후 결과가 종합되는 과정
- 랜덤 포레스트는 결정 트리의 과적합 문제를 줄일 수 있음



랜덤 포레스트 모델

sklearn.ensemble.RandomForestClassifier 클래스

- RandomForest Classifier 클래스의 하이퍼파라미터는 기본적으로 sklearn. tree.DecisionTreeClassifier클래스의 하이퍼파라미터를 포함

하이퍼파라미터	주요값	기본값	의미
n_estimators	int>0	100	트리 개수
bootstrap	bool	True	부트스트랩 샘플 사용 여부 - True: 개별 트리를 학습할 때 부트스트랩 샘플을 사용 - False: 개별 트리를 학습할 때 부트스트랩 샘플 대신 항상 전체 데이터셋 사용. 이때도 max_features에 따라 트리별 랜덤성이 발생하므로 랜덤 포레스트 모델에서의 각각의 트리는 서로 다른 구조를 지닌다.
oob_score	bool	False	bootstrap=True일 때만 유효하며, 일반화 점수[generalization score]를 계산할 때 OOB 샘플을 사용할지 선택
max_samples	None, int, 0<float≤1	None	bootstrap=True일 때만 유효하며, 각각의 베이스 학습기를 학습하기 위해 추출할 샘플 개수를 지정 - None: 전체 샘플 추출 - int: max_samples 건 추출 - float: n과 max_samples의 곱에 해당하는 샘플 개수를 추출

랜덤 포레스트 모델

sklearn.ensemble.RandomForestRegressor 클래스

- RandomForestRegressor 클래스는 랜덤 포레스트 회귀 모델을 구현

하이퍼파라미터	주요값	기본값	의미
n_estimators	int>0	100	트리 개수
criterion	'squared_error', 'absolute_error', 'poisson'	'squared_error'	분기 기준 'squared_error', 'absolute_error', 'poisson'은 DecisionTreeRegressor 클래스에서와 의미가 같다. 'friedman_mse'는 RandomForestRegressor에서는 구현되지 않았다.
bootstrap	bool	True	개별 트리를 학습할 때 부트스트랩 샘플을 사용할지 결정 - False: 개별 트리를 학습할 때마다 부트스트랩 샘플 대신 전체 데이터셋 사용
oob_score	bool	False	bootstrap=True일 때만 유효하며, 일반화 점수를 계산할 때 OOB 샘플을 사용할지 선택
max_samples	None, int, 0<float≤1	None	Bootstrap=True일 때만 유효하며, 각각의 베이스 학습기를 학습하기 위해 추출할 샘플 개수를 지정 - None: 전체 샘플 추출 - int: max_samples개 추출 - float: n과 max_samples의 곱에 해당하는 샘플 개수를 추출

랜덤 포레스트 모델

sklearn.ensemble.RandomForestClassifier 클래스

```
from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

X, y = load_iris(return_X_y=True, as_frame=True)
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.33,
                                                    random_state=1234)
```

풀이

```
clf = RandomForestClassifier(random_state=1234,
                             n_estimators=50,
                             criterion='gini',
                             max_depth=4,
                             max_features='log2')

y_pred = clf.fit(X_train, y_train).predict(X_test)
print(f'정확도:{(y_pred == y_test).mean() * 100: .2f}%')
```

```
정확도: 98.00%
```

랜덤 포레스트 모델

sklearn.ensemble.RandomForestRegressor클래스

```
from sklearn.ensemble import RandomForestRegressor

reg = RandomForestRegressor(n_estimators=5,
                            random_state=1234,
                            max_depth=3,
                            min_samples_split=4)
y_pred = reg.fit(X_train, y_train).predict(X_test)
print(f'RandomForestRegressor로 학습한 모델의 테스트 데이터셋 기준 MAE:{(np.abs(y_pred - y_test)).mean(): .2f}')
```

```
RandomForestRegressor로 학습한 모델의 테스트 데이터셋 기준 MAE: 45.39
```

그레이디언트 부스팅 트리 모델

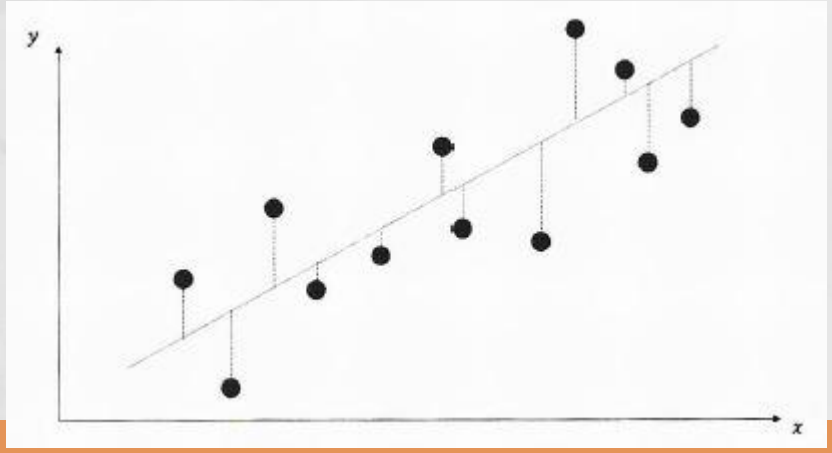
에이다부스트 모델 이론

- 에이다부스트(AdaBoost)는 GBT보다 이전 세대에 등장한 부스팅 트리 기법
- 앞 단계의 모델이 잘 풀지 못하는 어려운 케이스 쪽에 집중할 수 있게 앞쪽 모델에서 틀린 샘플의 추출 확률을 높인다는 점에서 차이
- 각 단계의 복원 추출 과정에서 이전에 틀린 샘플의 추출 확률이 높아지기 때문에 틀린 샘플이 학습 데이터로 여러번 등장할 가능성이 커짐

그레이디언트 부스팅 트리 모델

그레이디언트 부스팅 트리 모델

- 그레이디언트 부스팅 트리(gradient boosting trees, GBT)는 부스팅 기법의 하나인 GBT을 결정 트리에 적용한 앙상블 모델
- 회귀와 분류 문제에 모두 적용할 수 있으며 높은 성능
- 단계별로 샘플의 가중치를 조정하는 에이다부스트와 달리 GBT에서는 이전 단계에서 학습하지 못한 잔차를 새로운 목표 변수로 두고 학습한다는 아이디어에 기반



그레이디언트 부스팅 트리 모델

sklearn.ensemble 모듈

- GradientBoostingClassifier 클래스와 Gradient Boosting
- Regressor 클래스는 각각 GBT 분류 모델과 GBT 회귀 모델을 구현
- GBT 분류 모델과 GBT 회귀 모델은 모두 DecisionTreeRegressor 클래스를 기반
- GBT가 연속적인 손실 함수를 최소화하는 방식으로 업데이트 하기 때문

그레이디언트 부스팅 트리 모델

sklearn.ensemble 모듈

- GradientBoostingRegressor 클래스는 기본적으로 DecisionTreeRegressor 클래스의 하이퍼파라미터를 포함

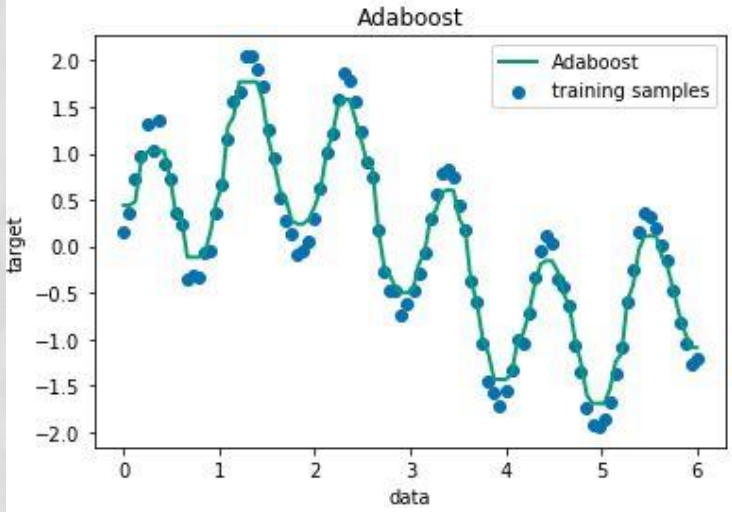
하이퍼파라미터	주요값	기본값	의미
loss	'squared_error', 'absolute_error', 'huber', 'quantile'	'squared_error'	손실 함수 선택 - 'squared_error': 제곱 오차 손실 사용 - 'absolute_error': 절대 오차 손실 사용 - 'huber': Huber 손실 사용 - 'quantile': 분위수 손실 사용
learning_rate	float>0	0.1	모델 규제를 위해 각 부스팅 단계에 적용하는 학습률
n_estimators	int>0	100	부스팅 단계의 수. GBT 모델은 일반적으로 과적합에 강건(robust)하므로 큰 n_estimators 값이 선호될 수 있다.
subsample	float	1.0	모델의 규제를 위해 각 트리를 만들 때 전체 데이터 대신 subsample의 비율만을 랜덤 샘플링하여 사용. 일반적으로는 비복원 추출을 사용하나 복원 추출도 사용할 수 있다. subsample을 작게 설정하면 모델의 분산은 감소하지만 편향은 증가한다.

그레이디언트 부스팅 트리 모델

AdaBoostRegressor 클래스

```
from sklearn.ensemble import AdaBoostRegressor

regr_2 = AdaBoostRegressor(DecisionTreeRegressor(max_depth=4),
                           n_estimators=300,
                           random_state=rng).fit(X, y)
y_2 = regr_2.predict(X)

plt.scatter(X, y, color=colors[0], label="training samples")
plt.plot(X, y_2, color=colors[2], label="Adaboost", linewidth=2)
plt.xlabel("data"), plt.ylabel("target"), plt.title("Adaboost"), plt.legend()
```

```
(Text(0.5, 0, 'data'),
 Text(0, 0.5, 'target'),
 Text(0.5, 1.0, 'Adaboost'),
 <matplotlib.legend.Legend at 0x1ae7767f320>)
```



그레이디언트 부스팅 트리 모델

GradientBoostingRegressor 클래스

```
from sklearn.datasets import load_diabetes
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import train_test_split
import numpy as np

df = load_diabetes(as_frame=True)['frame']
X = df.iloc[:, :-1]
y = df.iloc[:, -1]

X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.33,
                                                    random_state=1234)
```

```
reg = GradientBoostingRegressor(learning_rate=0.05,
                                n_estimators=150,
                                max_depth=5,
                                random_state=1234)

y_pred = reg.fit(X_train, y_train).predict(X_test)
print(f'테스트 데이터셋 기준 RMSE:{np.sqrt(((y_pred - y_test)**2).mean()): .2f}')
```

```
테스트 데이터셋 기준 RMSE: 54.36
```

그레이디언트 부스팅 트리 모델

GBT 모델 이후의 부스팅 기법

- XGBoost(extreme gradien,boosting)의 기본 원리는 GBT 모델과 비슷하지만 GBT 모델에 병렬 처리,하드웨어 최적화, 과적합 규제 페널티 등의 여러 개념을 도입하여 최적화한 모델
- XGBoost에서는 한 결정 트리를 학습할 때 모든 임계값을 기준으로 분할해서 불순도를 비교하는 대신 데이터셋을 수많은 서브데이터의 구획으로 분할한 다음, 그 서브데이터 내에서의 최적값을 각 병렬 환경에서 찾음
- XGBoost는 오픈 소스 소프트웨어 라이브러리

그레이디언트 부스팅 트리 모델

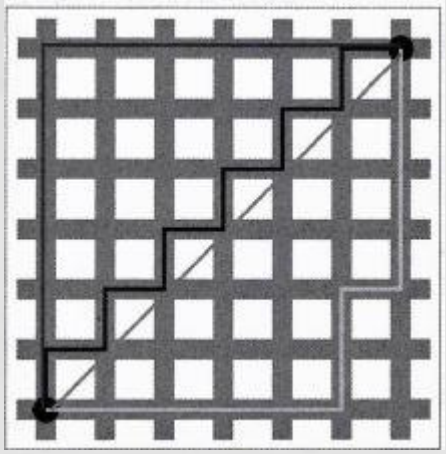
GBT 모델 이후의 부스팅 기법

- LightGBM은 XGBoost의 장점은 계승하고 단점은 보완하는 방식으로 개발된 모델
- XGBoost에 비하여 학습에 걸리는 시간과 메모리 사용량을 줄이고 다양한 기능을 추가로 구현한 가벼운 GBT 모델을 제공
- LightGBM은 오픈 소스 소프트웨어이며 파이썬에서는 lightgbm 패키지로 제공

K-최근접 이웃 모델

거리 메트릭

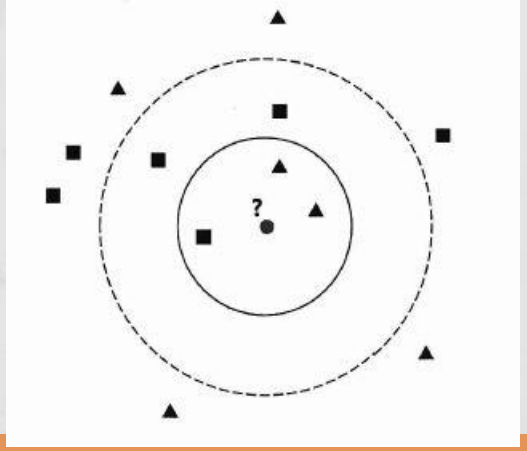
- KNN을 포함하여 거리에 기반을 두고 동작하는 모델은 거리 메트릭에 따라 모델링 결과가 달라짐
- 거리 메트릭은다양한 방법으로 정의
- 민코프스키 메트릭
- 유클리드 거리와 맨해튼 거리를 비교



K-최근접 이웃 모델

K-최근접 이웃 모델

- 관측한 입력 피처로 목표 변수를 예측하는 가장 직관적인 방법의 하나로, 입력 피처와 가장 비슷한 K개의 (알려진) 피처값과 이에 대응하는목표 변수의 쌍
- K-최근접 이웃(K-nearest neighbors, KNN) 모델은 입력값에서 가장 가까운 거리에 있는 K개 샘플의 출력값을 종합하여 입력값의 출력값을 예측하는 기법



K-최근접 이웃 모델

sklearn. neighbors.KNeighborsClassifier 클래스

- KNeighborsClassifier 클래스의 주요 하이퍼파라미터

- n은 학습 데이터의 샘플 개수, K는 최근접 이웃 개수

하이퍼파라미터	주요값	기본값	의미
n_neighbors	int>0	5	최근접 이웃의 수 K 설정
weights	'uniform', 'distance'	'uniform'	예측 가중치 설정 - 'uniform': 최근접 이웃 안에 포함되기만 하면 동등한 가중치 부여. 즉, K개의 최근접 이웃에 대한 가중치는 모두 $1/K$로, 그 외의 데이터의 가중치는 0으로 설정한다. 가장 간단한 형태의 옵션으로 생각할 수 있다. - 'distance': 최근접 이웃에 대하여 거리의 역수에 해당하는 가중치를 설정. 따라서 가까운 거리에 있는 이웃의 클래스가 최종 클래스 결정에 더 많이 기여한다.
algorithm	'auto', 'ball_tree', 'kd_tree', 'brute'	'auto'	최근접 이웃 탐색 알고리즘 선택 - 'ball_tree': Ball 트리 알고리즘 사용 - 'kd_tree': KD 트리 알고리즘 사용 - 'brute': 완전 탐색 알고리즘 사용 - 'auto': 입력 데이터를 기준으로 추천 알고리즘을 선택 후 진행. 예를 들어, 피처 개수가 15보다 크거나 $K \geq n/2$일 때는 완전 탐색 알고리즘을 사용한다.

K-최근접 이웃 모델

sklearn. neighbors.KNeighborsClassifier 클래스

- KNeighborsClassifier 클래스의 주요 하이퍼파라미터

leaf_size	int>0	30	algorithm이 'ball_tree'이거나 'kd_tree'일 때만 유효하다. 이 값은 최근접 이웃 탐색 결과에는 영향을 끼치지 않으나 알고리즘의 수행 속도와 메모리에 영향을 끼치며 최적값은 데이터의 형태에 따라 다르다.
p	int≥1	2	p=1(맨해튼 거리)나 p=2(유클리드 거리)를 주로 사용한다.
metric	'minkowski', 'manhattan', 'mahalanobis' 등 DistanceMetric 클래스[2]의 메트릭 함수	'minkowski'	거리 메트릭 선택 - 'minkowski': 가장 기본적인 형태이자 기본값으로, 민코프스키 메트릭을 계산한다. p 하이퍼파라미터와 같이 사용한다. - 'manhattan': 맨해튼 거리 사용. metric='minkowski'로 하고 p=1로 설정한 결과와 같다. - 'mahalanobis': 마할라노비스 거리[Mahalanobis distance] 사용

K-최근접 이웃 모델

sklearn. neighbors.KNeighborsClassifier 클래스

- KNN 분류 모델은 거리 기반으로 동작하므로 sklearn.preprocessing 모듈의 StandardScaler 클래스나 MinMaxScaler 클래스 등으로 피처 스케일 링을 먼저 수행하는 것이 좋음
- 극단적으로 K가 학습 데이터의 크기와 같다면 K개의 최근접 이웃은 테스트 샘플의 피처 값에 관계없이 모든 학습 데이터 샘플을 가르킴
- 모든 테스트 샘플의 예측값은 학습 데이터 샘플 목표값의 최빈값(분류 모델)이나 평균값(회귀 모델)으로 얻어지므로 지향하는것이 좋음

K-최근접 이웃 모델

sklearn. neighbors.KNeighborsClassifier 클래스

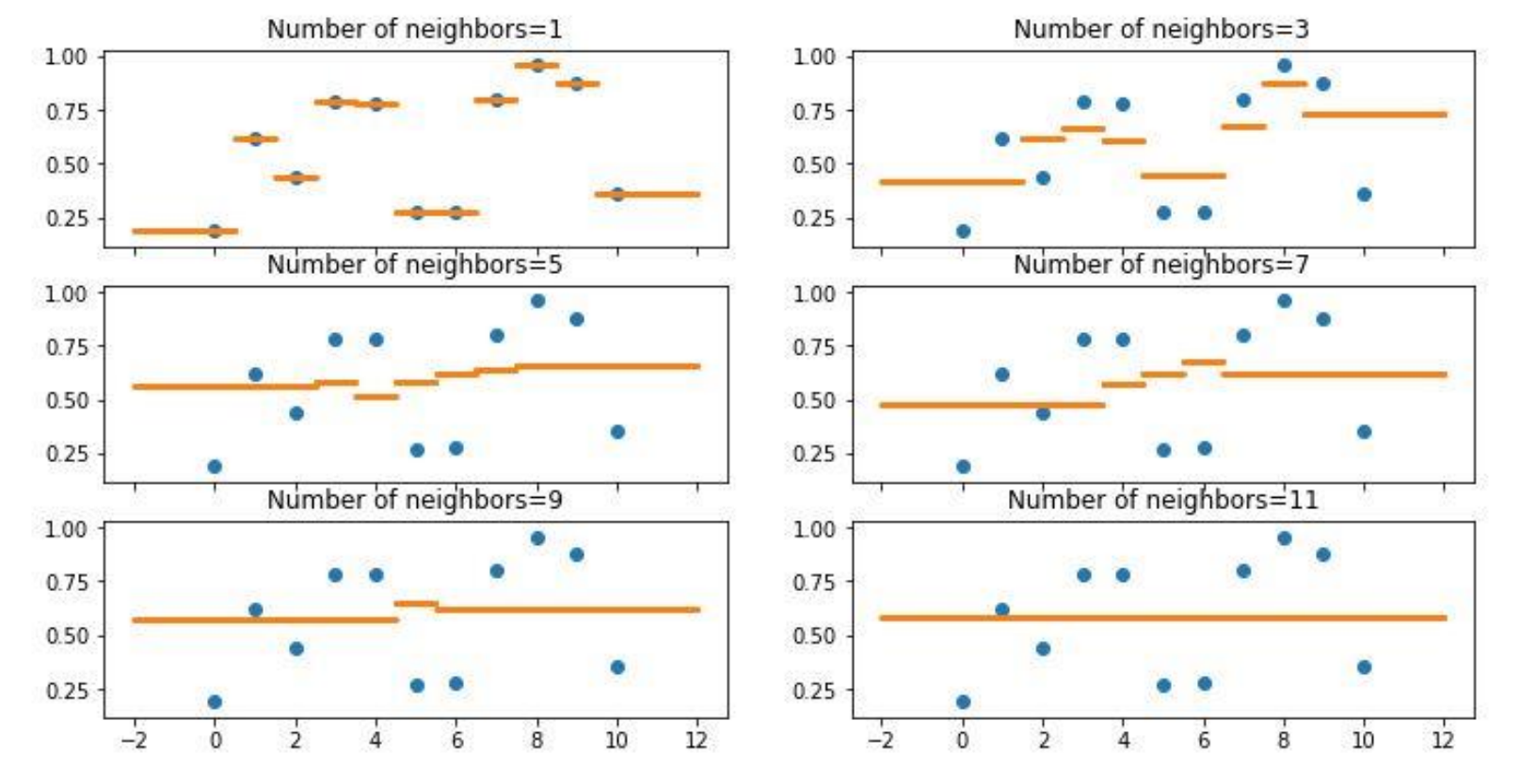
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsRegressor
%matplotlib inline

X_train = pd.DataFrame({'x': [x for x in range(11)]})

np.random.seed(1234)
y_train = np.random.uniform(0, 1, 11)
```

```
X_test = pd.DataFrame({'x': np.linspace(-2, 12, 1000)})
```

```
fig, axs = plt.subplots(3, 2, sharex=True, figsize=(12, 6))
for ind in range(6):
    i = int(ind / 2)
    j = ind % 2
    n_neighbors = 2 * ind + 1
    reg = KNeighborsRegressor(n_neighbors=n_neighbors).fit(X_train, y_train)
    y_test = reg.predict(X_test)
    axs[i, j].scatter(X_train, y_train)
    axs[i, j].scatter(X_test, y_test, s=3)
    axs[i, j].set_title('Number of neighbors=' + str(n_neighbors))
```

K-최근접 이웃 모델

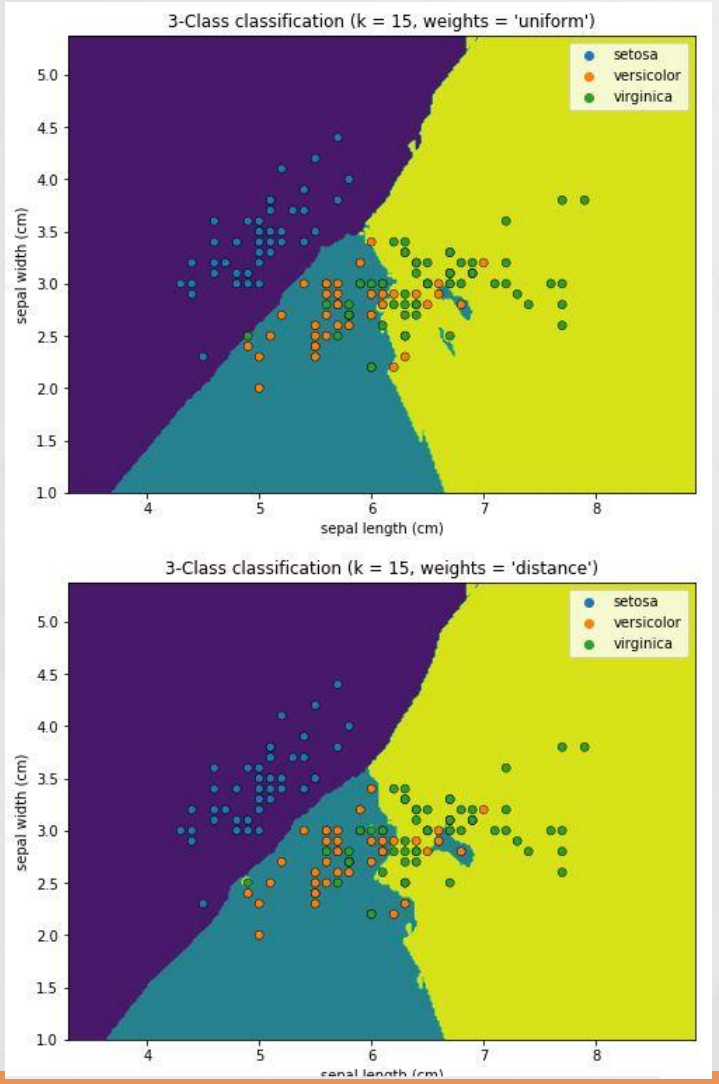
sklearn. neighbors.KNeighborsClassifier 클래스



K-최근접 이웃 모델

sklearn. neighbors.KNeighborsClassifier 클래스

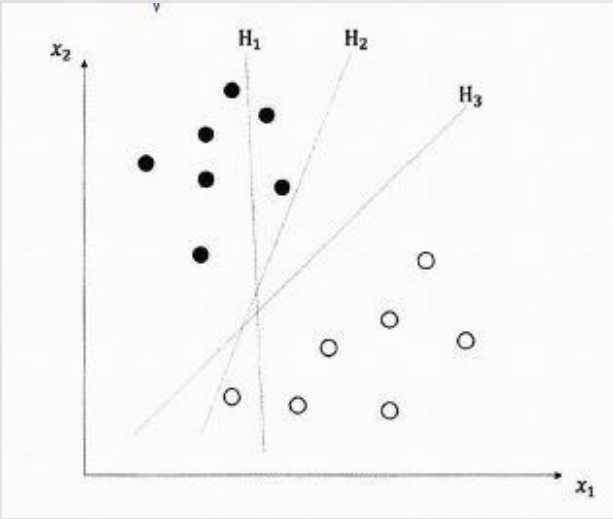
```
n_neighbors = 15
h = 0.02  # mesh의 단계 크기

for weights in ["uniform", "distance"]:
    clf = KNeighborsClassifier(n_neighbors, weights=weights).fit(X, y)

    # 결정 경계를 meshgrid로 표현
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])

    Z = Z.reshape(xx.shape)
    plt.figure(figsize=(8, 6))
    plt.contourf(xx, yy, Z)

    # 학습 데이터를 scatter로 표현
    sns.scatterplot(x=X[:, 0],
                    y=X[:, 1],
                    hue=iris.target_names[y],
                    alpha=1.0,
                    edgecolor="black")
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    plt.title("3-Class classification (k = %i, weights = '%s')" %
              (n_neighbors, weights))
    plt.xlabel(iris.feature_names[0])
    plt.ylabel(iris.feature_names[1])
```



서버트 벡터 머신 모델

서버트 벡터 모신 모델이란?

- 서포트 벡터머신(support vector machine, SVM)은 분류와 회귀 문제 모두에 적용할 수 있는 지도 학습 모델
- SVM은 좋은 경 계선을 찾는 모델이 라고 할 수 있음
- 기본적 인 SVM은 두 클래스 레이블이 분포된 영역을 최대한 잘분리하는 결정 경계를 찾는 것이 목표



서버트 벡터 머신 모델

sklearn.svm.SVC 클래스

- sklearn.svm.SVC 클래스는 C-SVC 분류 모델을 구현

- SVC 클래스의 주요 하이퍼파라미터

하이퍼파라미터	주요값	기본값	의미
C	float>0	1.0	규제 강도의 역수
kernel	'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'	'rbf'	사용할 커널의 종류. 사용자 지정 함수를 사용할 수 있다.
probability	bool	False	확률적 추정을 사용할지의 여부. 교차검증법cross validation을 사용하므로 속도가 떨어질 수 있다.
tol	float>0	1e-3	정지 조건에 대한 허용 오차
class_weight	None, 'balanced', dict	None	클래스 가중치 설정. 클래스 불균형class imbalance 효과를 줄이고자 할 때 유용하다. - None: 모든 클래스에 같은 가중치 부여 - 'balanced': 클래스 빈도에 반비례하는 가중치 설정 - dict: 사용자 지정 가중치 부여

서버트 벡터 머신 모델

sklearn.svm.SVC 클래스

- sklearn.svm.SVC 클래스는 C-SVC 분류 모델을 구현

- SVC 클래스의 주요 하이퍼파라미터

max_iter	int>0 또는 int=-1	-1	이터레이션의 상한선 - max_iter=-1: 상한선 없음. tol에 의한 정지 조건에 도달할 때까지 계속 학습한다.
decision_function_shape	'ovr', 'ovo'	'ovr'	다중 클래스일 때 반환할 결정 함수의 종류 설정. 이진 분류라면 무시한다. - 'ovr': OVR(one-vs-rest) 결정 함수 - 'ovo': libsvm의 OVO(one-vs-one) 결정 함수
random_state	None, int	None	probability=True에 한해 데이터 셔플링(data shuffling)에 대한 랜덤성을 제어

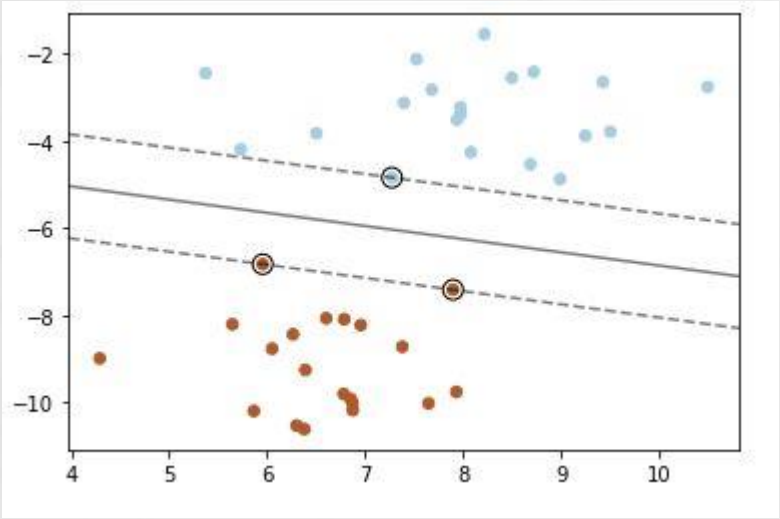
서버트 벡터 머신 모델

sklearn.svm.SVC 클래스

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm
from sklearn.datasets import make_blobs

X, y = make_blobs(n_samples=40, centers=2, random_state=6)
```

```
clf = svm.SVC(kernel="linear", C=1000).fit(X, y)
```

```
plt.scatter(X[:, 0], X[:, 1], c=y, s=30, cmap=plt.cm.Paired)

ax = plt.gca()
xlim, ylim = ax.get_xlim(), ax.get_ylim()

xx = np.linspace(xlim[0], xlim[1], 30)
yy = np.linspace(ylim[0], ylim[1], 30)
YY, XX = np.meshgrid(yy, xx)
xy = np.vstack([XX.ravel(), YY.ravel()]).T
Z = clf.decision_function(xy).reshape(XX.shape)

ax.contour(XX,
           YY,
           Z,
           colors="k",
           levels=[-1, 0, 1],
           alpha=0.5,
           linestyles=["--", "-", "--"])
ax.scatter(clf.support_vectors_[:, 0],
           clf.support_vectors_[:, 1],
           s=100,
           linewidth=1,
           facecolors="none",
           edgecolors="k")
```



서버트 벡터 머신 모델

sklearn.svm.SVC 클래스

```
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_breast_cancer
from sklearn.svm import SVC

X, y = load_breast_cancer(return_X_y=True, as_frame=False)
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.33,
                                                    random_state=1234)
```

```
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler

kernels = ['linear', 'poly', 'rbf']

for kernel in kernels:
    clf = make_pipeline(StandardScaler(),
                        SVC(kernel=kernel)).fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    acc = (y_test == y_pred).mean() * 100
    print(f'커널: {kernel}, 정확도:{acc: .2f}%')
```

```
커널: linear, 정확도: 95.21%
커널: poly, 정확도: 84.57%
커널: rbf, 정확도: 94.15%
```

다층 퍼셉트론 모델

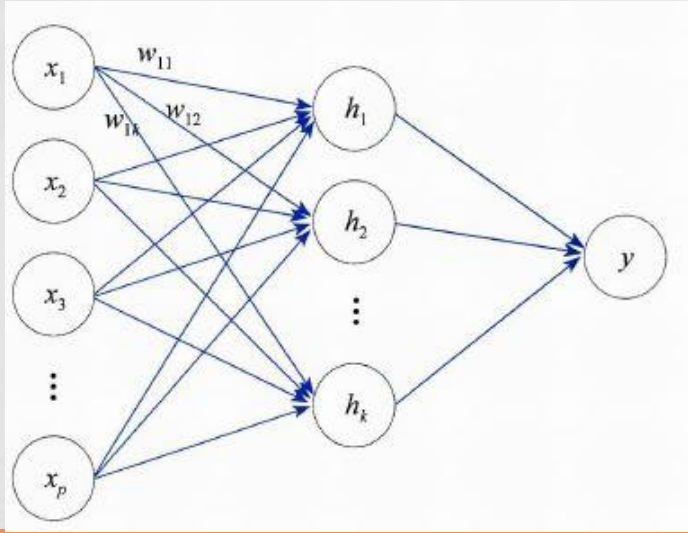
확률적 최적화 알고리즘

- 확률적 최적화 알고리즘은 대형 데이터셋에 적용하기 좋은 최적화 기법
- 전체 데이터셋을 한 번에 학습하는 것이 아니라 일부 데이터셋 만을 샘플링 한다는 것이 기본 아이디어
- 확률적 경사하강법 (stochastic gradient descent, SGD)은 가장 기본적 인 확률적 최적화 알고리즘의 하나
- 데이터셋을 미니 배치라고 불리는 서브데이 터셋으로 랜덤하게 분할한 후 순서대로 경사 하강법을 적용
- MLP뿐만 아니라 로지스틱 회귀 모델이나 선형 SVM 등 다른 기법에도 적용

다층 퍼셉트론 모델

다층 퍼셉트론(multilayer perceptron, MLP)

- 생물의 신경망 구조에 착안하여 발전된 인공 신경망(neural network) 모델의 하나
- MLP는 여러 개의 레이어 또는 층으로 구성
- 레이어는 1개의 입력층과 1개의 출력층 그리고 입력층과 출력층사이에 있는 1 개 이상의 은닉층 layer으로 구분



다층 퍼셉트론 모델

sklearn.neural_network.MLPClassifier 클래스

- sklearn.neural_network.MLPClassifier 클래스는 MLP 분류 모델을 구현

- MLPClassifier의 주요 하이퍼파라미터

하이퍼파라미터	주요값	기본값	의미
hidden_layer_sizes	tuple	(100,)	튜플[tuple] 순서대로 해당 은닉층의 뉴런의 수를 의미
activation	'identity', 'logistic', 'tanh', 'relu'	'relu'	은닉층에 사용할 활성화 함수 선택 - 'identity': 활성화 과정 없이 입력값을 그대로 반환하는 항등 함수 - 'logistic': 시그모이드 함수 - 'tanh': 하이퍼볼릭 탄젠트 함수 - 'relu': ReLU 함수
solver	'lbfgs', 'sgd', 'adam'	'adam'	최적화 알고리즘 선택 - 'lbfgs': L-BFGS-B 알고리즘 - 'sgd': SGD 알고리즘 - 'adam': Adam 알고리즘
alpha	float≥0	0.0001	L2 규제항의 계수

다층 퍼셉트론 모델

sklearn.neural_network.MLPClassifier 클래스

- MLPClassifier의 주요 하이퍼파라미터

batch_size	'auto', int	'auto'	미니 배치mini-batch의 크기를 결정하며 확률적 최적화 알고리즘에서만 적용 - 'auto': min(200, n)의 미니 배치 크기 사용
learning_rate_init	float>0	0.001	확률적 최적화 알고리즘에서의 초기 학습률
max_iter	int>0	200	에포크epoch의 최대 횟수. tol에 따른 학습 조기 종료가 일어나지 않는다면 max_iter번 후 학습을 종료한다. 확률적 최적화 알고리즘에서는 max_iter가 파라미터의 업
shuffle	bool	True	확률적 최적화 알고리즘에서 이터레이션마다 데이터 셔플링data shuffling 여부를 결정
random_state	int	None	랜덤성의 제어. 이는 학습 파라미터 초기화 early_stopping이 True일 때의 검증 데이터셋 분할, solver가 'sgd'나 'adam'일 경우 배치 추출 등에서의 랜덤성을 제어하는 데 적용한다.
tol	float	1e-4	학습 조기 종료에 관련된 허용 오차. 손실 함수의 값이나 성능이 n_iter_no_change번의 이터레이션 동안 tol 이상만큼 향상하지 않고 learning_rate가 'adaptive'가 아닌 경우 학습이 종료된다.
early_stopping	bool	False	확률적 최적화 알고리즘에서만 적용되며, 학습 조기 종료를 판단할 때 검증 데이터셋을 사용할 것인지를 선택한다. - True: n_iter_no_change회 에포크 동안 검증 데이터셋에서 tol만큼의 개선이 없다면 학습 조기 종료 - False: n_iter_no_change회 에포크 동안 학습 데이터셋 전체에서 tol만큼의 개선이 없다면 학습 조기 종료
validation_fraction	0<float<1	0.1	early_stopping이 True일 때만 적용하며 학습 조기 종료 판단 시 학습 데이터셋 중 검증 데이터셋의 비율 설정
n_iter_no_change	int	10	확률적 최적화 알고리즘에서 tol 개선 여부에 따라 학습 조기 종료를 판단할 에포크의 수

다층 퍼셉트론 모델

sklearn.neural_network.MLPClassifier 클래스

- MLPClassifier의 주요 하이퍼파라미터

하이퍼파라미터	주요값	기본값	의미
learning_rate	'constant', 'invscaling', 'adaptive'	'constant'	학습률 크기 설정 - 'constant': learning_rate_init값 계속 적용 - 'invscaling': 학습률을 점진적으로 감소. t번째 단계에서 $\frac{\text{learning\_rate\_init}}{t^{\text{power\_t}}}$ 로 설정한다. - 'adaptive': 학습 데이터셋의 비용 함수 값이 계속해서 적절히 감소하는 한 학습률을 learning_rate_init로 유지한다. 연속한 두 에포크에서 학습 데이터셋의 손실이 tol 이하로 감소하지 않거나 early_stopping=True에서 검증 데이터셋 기준 성능이 tol 만큼 증가하지 못한다면 현재의 학습률을 1/5배로 감소한다.
power_t	float>0	0.5	learning_rate가 'invscaling'일 때 학습률의 감쇄 지수
momentum	0<float≤1	0.9	경사 하강법을 이용한 파라미터 업데이트에서의 모멘텀$^{\text{momentum}}$
nesterovs_momentum	bool	True	momentum>0일 때만 적용하며 Nesterov 모멘텀을 사용할지를 결정

다층 퍼셉트론 모델

sklearn.neural_network.MLPClassifier 클래스

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier

X, y = load_breast_cancer(return_X_y=True, as_frame=False)
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.30,
                                                    random_state=1234)
```

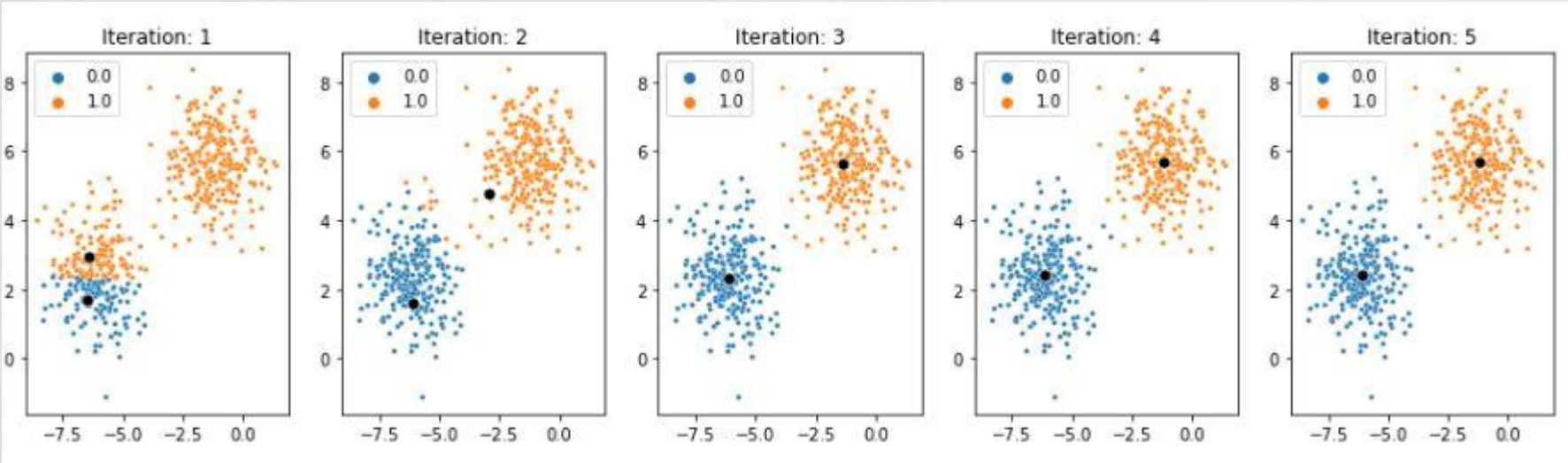
```
from sklearn.metrics import recall_score

clf = MLPClassifier(hidden_layer_sizes=(20, 10),
                    solver='adam',
                    max_iter=1000,
                    random_state=1234,
                    early_stopping=True)

y_pred = clf.fit(X_train, y_train).predict(X_test)

print(f'테스트셋 재현율:{recall_score(y_test, y_pred) * 100: .2f}%')
```

```
테스트셋 재현율: 91.43%
```

K-평균군집화모델

K-평균 군집화 (clustering) 모델이란?

- 주어진 K에 대하여 각 샘플과 그 샘플이 속한 군집의 중심까지 거리의 제곱합을 최소화하는 K개의 군집을 찾아내는 기법
- K-평균 군집화기 법은 큰 데이터셋에서도 효율적이며 다양한 분야에 서널리 사용



K-평균군집화모델

모델의 성능 평가

- 실루엣 계수는 -1 과 1 사이의 값이며, 같은 군집 내의 원소는 가깝고 다른 군집의 원소는 먼 좋은 군집화 결과에서는 1에 가깝고 부적절한 군집화 결과일수록 -1 에 가깝다는 장점
- 실루엣 계수는 샘플별로 계산할 수 있고, 이를 평균하여 전체 군집화 결과의 실루엣 계수값 또한 얻을 수 있음

K-평균군집화모델

sklearn.cluster.KMeans 클래스

K평균 군집화 기법은 거리를 기반으로 작동하므로 모델링 전에 각 피처를 스케일링

하이퍼파라미터	주요값	기본값	의미
n_clusters	int>0	8	생성할 클러스터의 수
init	'k-means++', 'random'	'k-means++'	초기화 방법 선택 - 'k-means++': 수렴 속도를 높이기 위한 초기 중심값을 설정 - 'random': 임의로 초기 중심값을 선택
n_init	int>0	10	설정된 알고리즘을 반복하는 횟수. 반복마다 서로 다른 중심 초깃값을 사용한다. 전체 수행 후 비용 함수가 가장 작은 결과를 최종 결과로 선택한다.
max_iter	int>0	300	각각의 알고리즘 반복에서 이터레이션의 상한값
tol	float>0	1e-4	조기 종료를 위한 허용 오차 설정. 두 이터레이션 동안의 군집 중심 변화를 Frobenius norm으로 구한 후 이 값이 tol 보다 작다면 종료한다.
random_state	int, None	None	중심 초기화에 대한 랜덤성 제어
algorithm	'lloyd', 'elkan'	'lloyd'	학습에 사용할 알고리즘 선택 - 'lloyd': 로이드 알고리즘(기본적인 EM-알고리즘) - 'elkan': 삼각 부등식triangle inequality을 이용하는 엘칸 알고리즘을 적용. 효율적으로 학습할 수 있으나 메모리를 많이 사용할 수 있다.

K-평균군집화모델

sklearn.cluster.KMeans 클래스

```
from sklearn.cluster import KMeans
import numpy as np

X = np.array([[1, 2], [1, 4], [1, 0], [10, 2], [10, 4], [10, 0]])
```

```
kmeans = KMeans(n_clusters=2, random_state=0).fit(X)
print(f'샘플별 군집 번호: {kmeans.labels_}')
```

```
샘플별 군집 번호: [1 1 1 0 0 0]
```

```
import numpy as np
from sklearn.cluster import KMeans
from sklearn import metrics
from sklearn.datasets import load_iris

X, y = load_iris(return_X_y=True)

kmeans_model = KMeans(n_clusters=3, random_state=1234).fit(X)
labels = kmeans_model.labels_
print(f"X의 군집화 결과의 실루엣 계수:{metrics.silhouette_score(X, labels, metric='euclidean'): .4f}")
```

```
X의 군집화 결과의 실루엣 계수: 0.5528
```

주성분 분석 모델

차원 축소

- 차원 축소(dimensionality reduction)는 주어진 피처 공간의 차원 수를 줄이는 기법
- 많은 머신러닝 모델은 차원의 수에 따라 시간과 공간 복잡도가 기하급수로 증가하며, 주어진 샘플 개수에 비해 차원의 수가 증가할 경우 차원의 저주 현상이 발생해 왜곡된 분석 결론
- 차원 축소는 데이터의 전처리 과정에서 주로 사용
- 차원 축소의 핵심은 최대한 정보의 손실을 적게 하며 차원의 수를 감소

주성분 분석 모델

주성분 분석 모델이란?

- 주성분 분석(principal component analysis, PCA)은 차원 죽소를 위한 비지도 학습 모델
- PCA에서는 원데이터가 가진 전체 정보를 최대한 훼손하지 않는 범위에서 차원을 축소
- 데이터에서 분산이 높은 부분(방향)은 최대한 유지하고 분산이 낮은 부분은 들어내는 방향으로 분석을 진행
- 원 데이터의 분산을 최대한 보존하며 차원을 축소하는데, 이를 위해 변수 간의 공분산 행렬을 분석

주성분 분석 모델

주성분 분석 모델이란?

- PCA에서는 원 데이터 공간에서 분산이 가장 커지는축을 첫 번째 주성분 축으로 진행
- 첫 번째 축에 수직인 차원 공간 중 분산이 두 번째로 커지는 축을 두 번째 주성분 축으로 함
- 필요한 주성분의 수만큼 반복
- 새로운 피처는 원래 피처를 선형 결합하여 얻을 수 있으며, 이를 주성분
- PCA는선형 차원축소기법

주성분 분석 모델

sklearn.decomposition.PCA클래스

- SVD에 기반을 두고 데이터를 저차원공간으로 투영하는 PCA모델을 구현
- pca클래스는SVD 이전 데이터에 대해 평균을 0으로 맞추는 평균 중심화

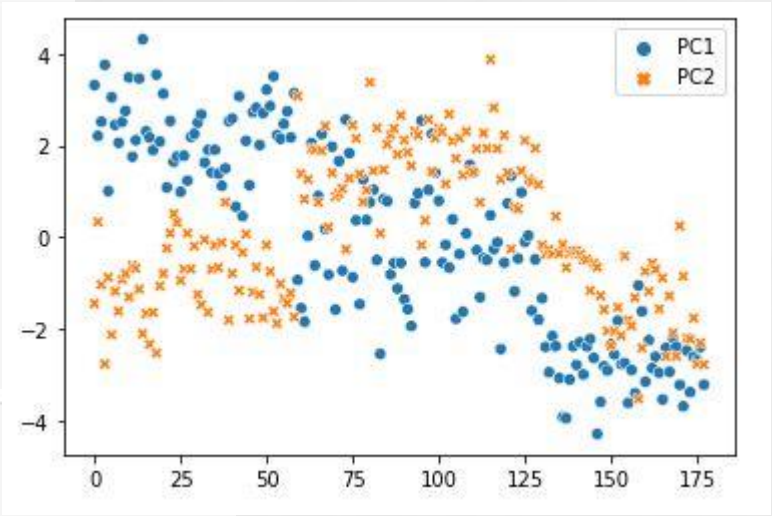
하이퍼파라미터	주요값	기본값	의미
n_components	None, int, 0<float<1, 'mle'	None	추출할 주성분의 수 - None: n_components=min(n, p)가 되어 전체 정보를 보존하는 최대 주성분을 선택하지만 svd_solver='arpack'일 때는 주성분의 수는 n과 p보다 모두 작아야 하므로 min(n, p)-1을 선택한다. - float: svd_solver='full'이면 전체 데이터에서 n_components이 넘는 비율의 분산이 설명될 때까지의 주성분을 추출 - 'mle': svd_solver가 'full'이나 'auto'이면 Minka(2000)에 소개된 최대 가능도 추정(maximum likelihood estimation, MLE) 기법으로 차원의 수를 결정
whiten	bool	False	분산 조정 여부 - True: 각각의 components_ 벡터에 $\sqrt{n}$을 곱하고 각각의 특잇값singular value을 나누어 출력 벡터의 각 분산을 1로 조정하고 출력값의 상관성을 제거한다. 이 과정은 신호signal의 분산 정보를 제거하므로 일부 정보 손실이 생길 수 있으나 이를 통해 PCA 이후에 진행될 머신러닝 모델에서 성능이 증가할 수도 있다.

주성분 분석 모델

sklearn.decomposition.PCA클래스

```
from sklearn.datasets import load_wine
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

X, _ = load_wine(return_X_y=True)
```

```
X = StandardScaler().fit_transform(X)

pca = PCA(n_components=2, random_state=1234)

X_extracted = pca.fit_transform(X)
X_extracted = pd.DataFrame(X_extracted, columns=['PC1', 'PC2'])

sns.scatterplot(data=X_extracted)
```



주성분 분석 모델

sklearn.decomposition.PCA클래스

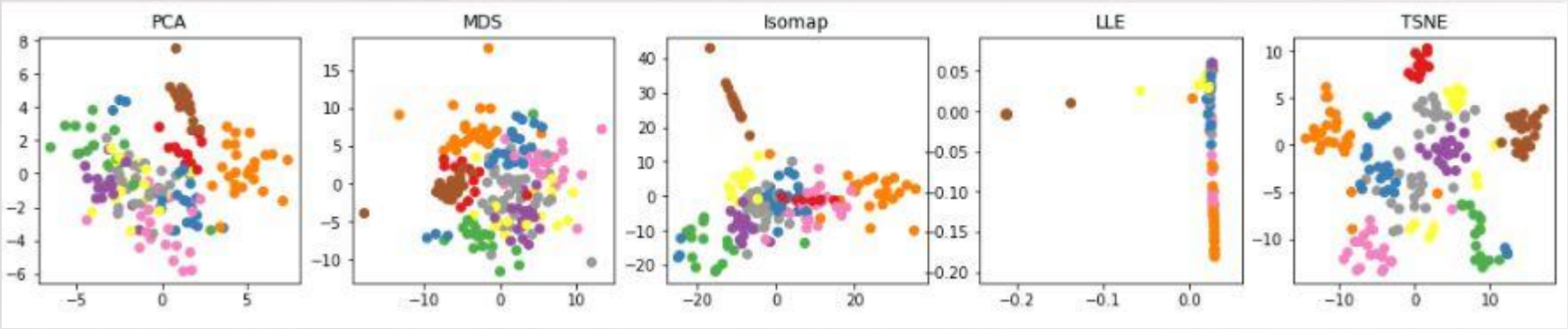
```
import matplotlib.pyplot as plt
from sklearn.datasets import load_digits
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.manifold import MDS, Isomap, LocallyLinearEmbedding, TSNE
from sklearn.pipeline import make_pipeline
%matplotlib inline

df = load_digits(as_frame=True)['frame'].sample(frac=0.1, random_state=1234)
X, y = df.drop(['target'], axis=1), df['target']
```

```
pca = make_pipeline(StandardScaler(), PCA(n_components=2, random_state=1234))
mds = make_pipeline(StandardScaler(), MDS(n_components=2, random_state=1234))
isomap = make_pipeline(StandardScaler(), Isomap(n_components=2))
lle = make_pipeline(StandardScaler(),
                    LocallyLinearEmbedding(n_components=2, random_state=1234))
tsne = make_pipeline(StandardScaler(), TSNE(n_components=2, random_state=1234))

methods = [("PCA", pca), ("MDS", mds), ("Isomap", isomap), ("LLE", lle),
           ("TSNE", tsne)]
```

```
fig, axs = plt.subplots(1, 5, figsize=(18, 3))
for i, (name, model) in enumerate(methods):
    X_embedded = model.fit_transform(X)
    axs[i].scatter(X_embedded[:, 0], X_embedded[:, 1], c=y, cmap="Set1")
    axs[i].set_title(name)
```

주성분 분석 모델

sklearn.decomposition.PCA클래스



정리

정리

- 최소 제곱법 모델
- 로지스틱 회귀 모델
- 라쏘 모델
- 릿지 회귀 모델
- 결정 트리 모델
- 랜덤 포레스트 모델
- 그레이디언트 부스팅 트리 모델
- K-최근접 이웃 모델
- 서포트 백터 머신 모델
- 다중 퍼셉트론 모델