Supervised Learning

분류와 회귀

- 지도학습의 두 종류.

- 분류
  - » 미리 정의된 가능성 있는 여러 클래스 레이블 중 하나를 예측하는 것
  - » 두 개의 클래스로 분류하는 이진 분류와 셋 이상의 클래스로 분류하는 다중 분류

- 회귀
  - » 연속적인 숫자 또는 부동소수점(실수) 데이터를 예측하는 것

- 출력 값의 연속성 여부가 두 기법을 구분하는 중요한 기준
  - » 일반적으로 연속성이 있으면 회귀, 없으면 분류
  - » 양적 데이터는 회귀, 범주형 데이터는 분류

일반화, 과대적합, 과소적합

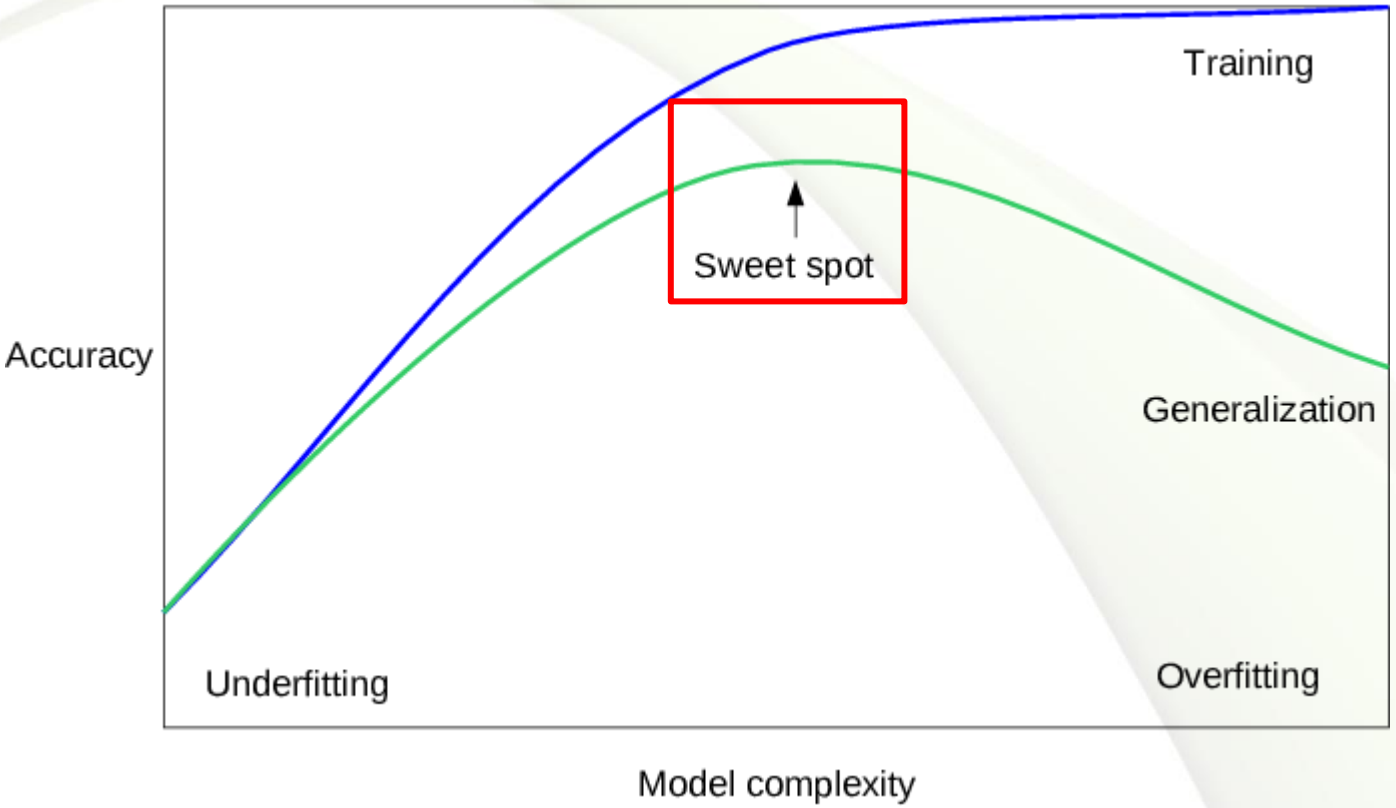
- 훈련 세트에서 테스트 세트로 일반화
  - » 모델이 처음 보는 데이터에 대해 정확하게 예측할 수 있게 되는 것
  - » 모델을 만들 때 가능한 정확하게 일반화하도록 구현해야 함

- 과대적합 (Overfitting)
  - » 모델이 훈련 세트에 너무 가깝게 맞춰져서 새로운 데이터에 일반화되기 어려운 경우
  - » 훈련 데이터는 잘 설명하지만 새로운 데이터에 대한 예측 정확도가 낮음

- 과소적합 (Underfitting)
  - » 모델을 지나치게 단순화해서 훈련 데이터와 테스트 데이터 모두에서 예측 정확도가 낮음

일반화, 과대적합, 과소적합

- 일반화 성능을 최대화 하는 모델을 찾는 것이 데이터 분석의 목표



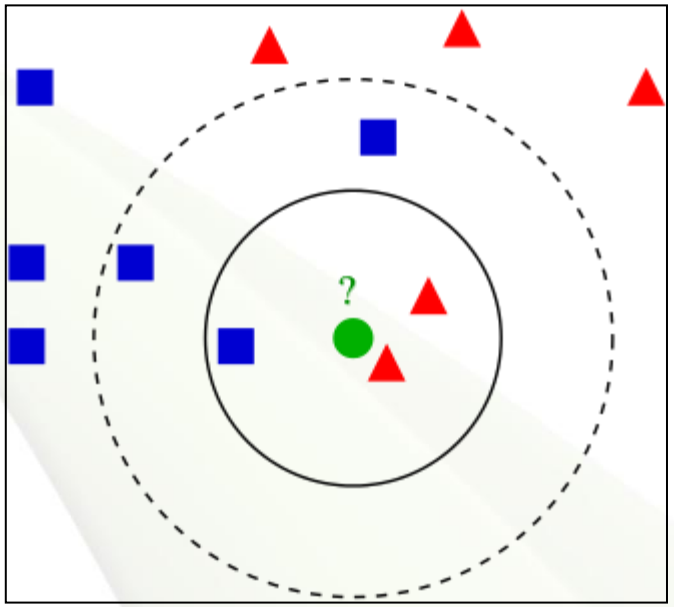
- 일반적으로 데이터가 많으면 다양성을 강화하기 때문에 큰 데이터 셋을 사용하면 과대적합 없이 복잡한 모델을 만드는 것 가능

K-Nearest Neighbors 알고리즘

KNN (K Nearest Neighbors)

- 가장 간단한 머신러닝 알고리즘으로 분류 및 회귀에 사용

- 새로운 데이터가 주어졌을 때 기존 데이터 가운데 가장 가까운 k개 이웃의 정보로 새로운 데이터를 예측하는 방법
  - 분류일 경우 이웃 데이터의 분류가 예측 값, 회귀일 경우 이웃 데이터의 종속 변수의 평균이 예측 값

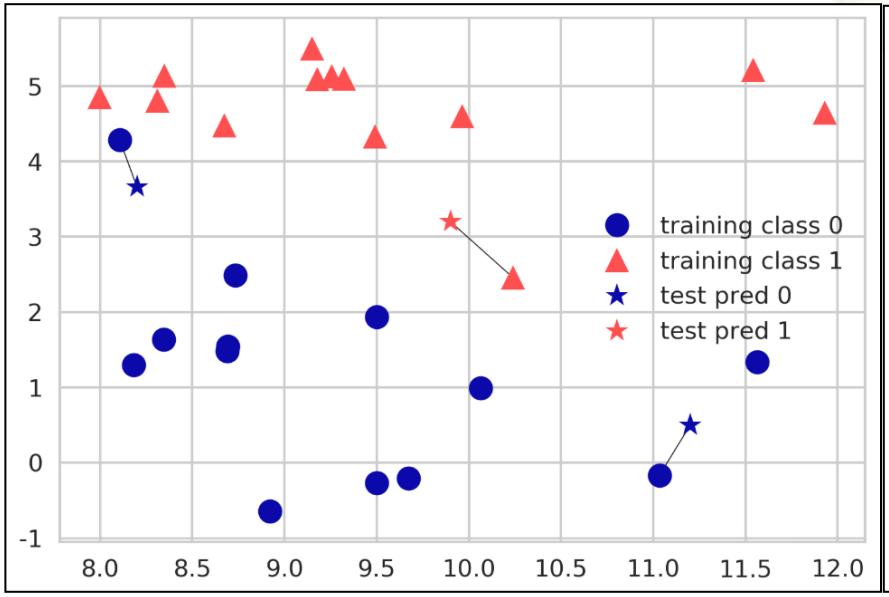


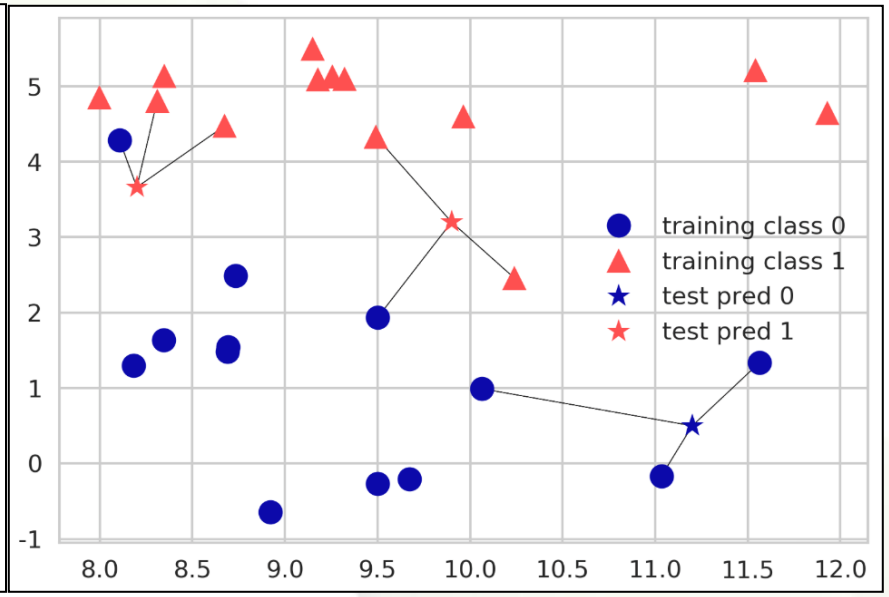
- 게으른 모델
  - 모델을 별도로 구축하지 않고 새로운 데이터가 발생했을 때 거리를 계산하고 예측

KNN (K Nearest Neighbors)

- KNN 분류 → K개의 이웃 데이터의 범주를 기반으로 범주 결정

neighbors = 1



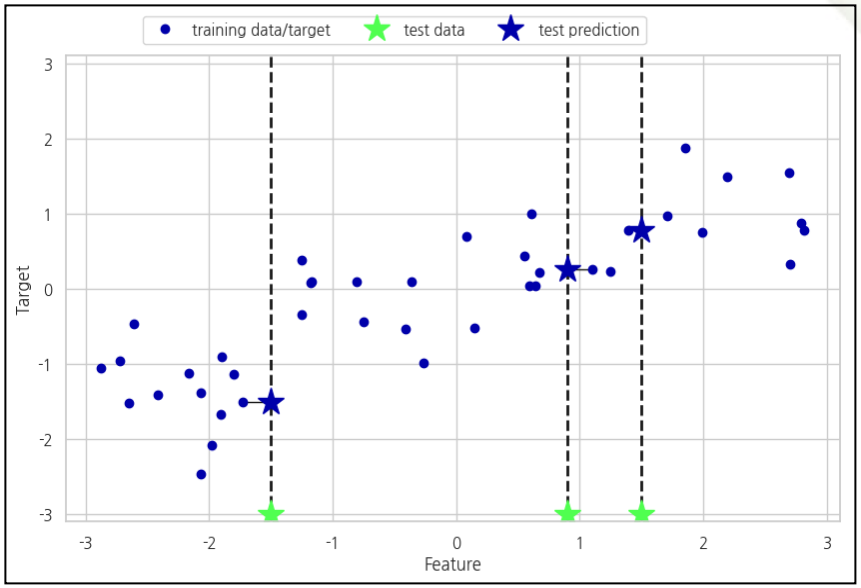


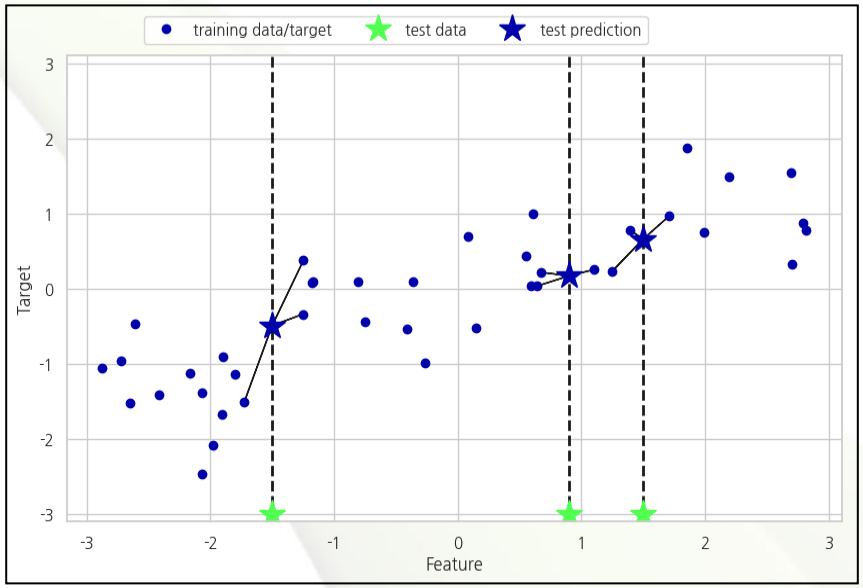
neighbors = 3

KNN (K Nearest Neighbors)

- KNN 회귀 → K개의 이웃 데이터의 평균을 기반으로 값 결정

Neighbors = 1





Neighbors =3

KNN (K Nearest Neighbor)

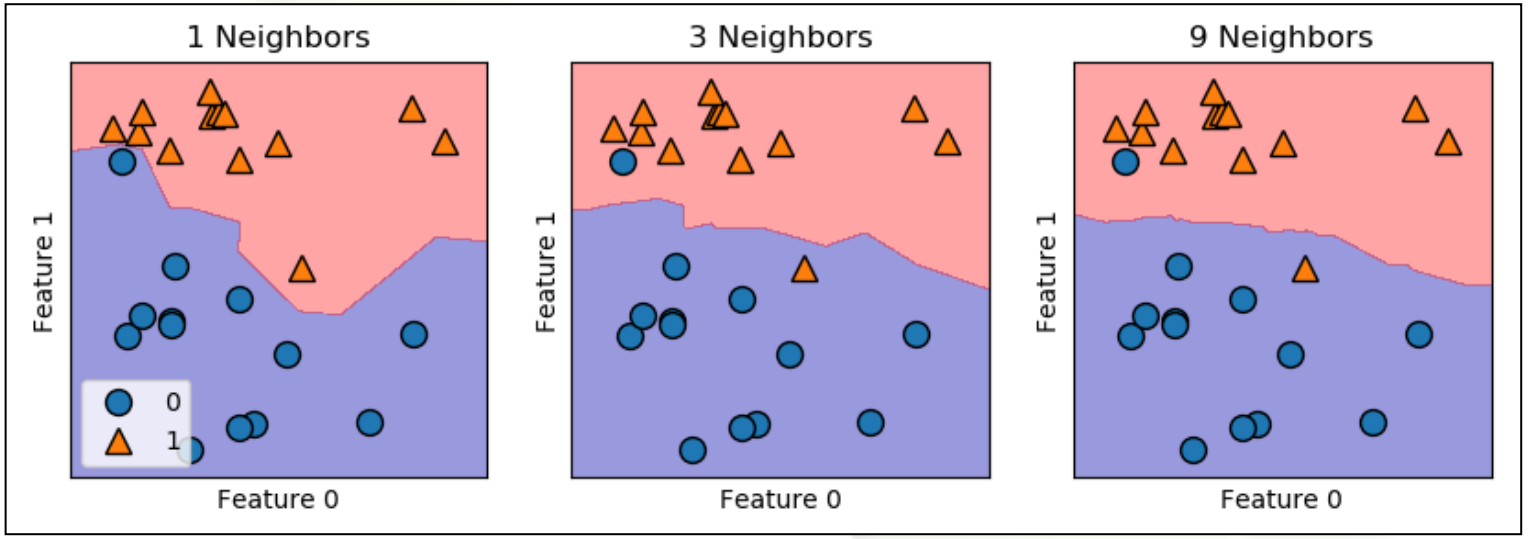
- 이웃 데이터와의 거리 측정
  - » 유클리디안, 맨하탄 등 다양한 거리 측정 방법 사용
  - » 문제의 복잡도, 데이터 타입 등에 따라 선택
  - » 가장 흔하게 사용되는 거리 척도는 유클리드 거리
  - » 거리 측정 전에 반드시 변수 정규화 필요

- 탐색할 이웃 수 (K)
  - » k가 작을 경우 데이터의 지역적 특성을 과도하게 반영 (overfitting)
  - » k가 클 경우 과도하게 정규화 (underfitting)
  - » 최적의 k값을 찾기 위해 k값을 작은 값에서 시작해서 큰 값으로 변경하면서 실험

KNN (K Nearest Neighbor)

- 탐색할 이웃 수(K)와 일반화 수준

분류



회귀

KNN (K Nearest Neighbor)

- 주요 매개 변수
  - » 데이터 포인트 사이의 거리 측정 방법 → 주로 유클리디안 거리 방식 사용
  - » 이웃의 수 → 3개 또는 5개에서 잘 동작하지만 상황에 따라 조정 필요

- 장점
  - » 이해하기 쉽고 빠르게 만들 수 있음
  - » 세밀하게 조정하지 않아도 비교적 좋은 성능 발휘 → 더 복잡한 알고리즘을 적용하기 전에 시도해 볼 수 있는 시작점으로 유용

- 단점
  - » 특성 또는 샘플의 개수가 많으면 예측이 느리게 처리됨
  - » 특성 값이 대부분 0인 데이터 세트에서는 잘 동작하지 않음
  - » 이런 이유로 현업에서는 활용도 낮음

KNN (K Nearest Neighbor) 구현

- 분류는 KNeighborsClassifier에 구현
- 회귀는 KNeighborsRegressor에 구현

- 평가는 score 함수 사용
  - » 분류는 정확도 (맞게 평가한 개수 / 전체 개수)
  - » 회귀는 R^2
    - › 0 ~ 1 사이의 값
    - › 종속변수에 대한 예측 값과 실제 값의 상관계수를 제곱한 값
    - › 전체 제곱합 중에서 회귀 제곱합이 설명하는 비중

$$\sum(y_i - \bar{y})^2 = \sum(\hat{y}_i - \bar{y})^2 + \sum(y_i - \hat{y}_i)^2$$

$$\text{SST} = \text{SSR} + \text{SSE}$$

$$R^2 = \frac{\text{설명된 변동}}{\text{총 변동}} = \frac{SSR}{SST} = 1 - \frac{SSE}{SST}$$

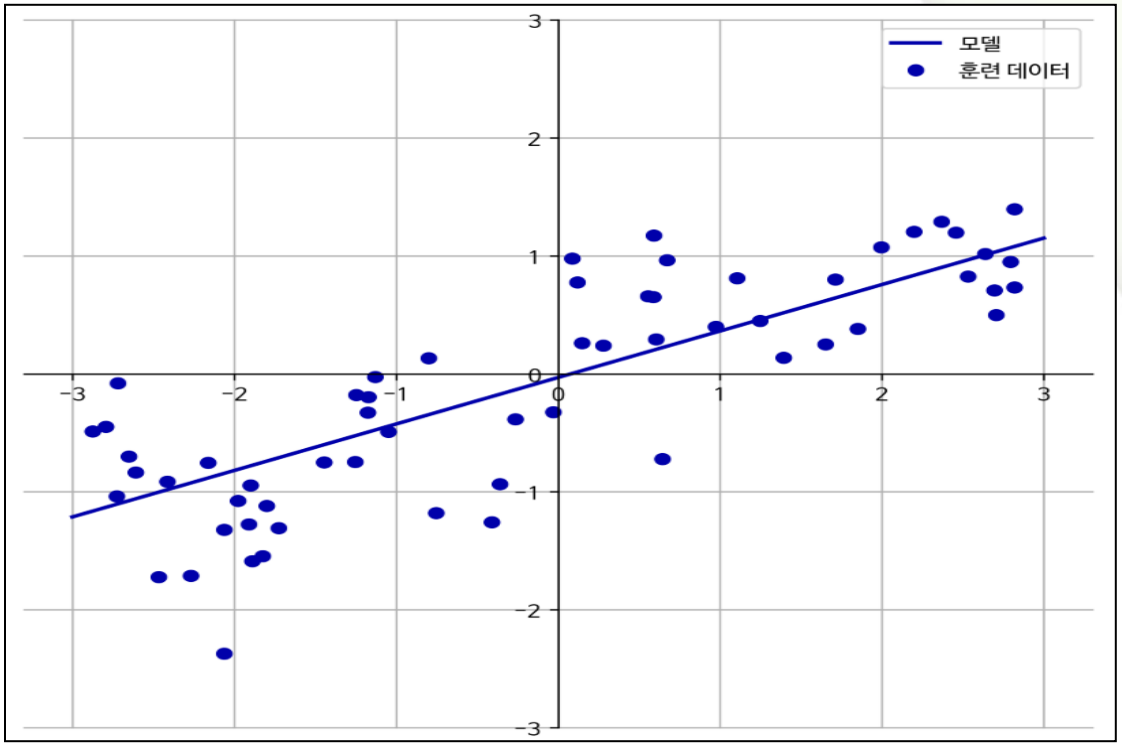
선형 모델

선형 모델

- 입력 특성에 대한 선형 함수를 만들어 예측 수행

- 선형 회귀 모델의 일반화된 예측 함수

$$y_i = \beta_1 x_{i1} + \cdots + \beta_p x_{ip} + \varepsilon_i = \mathbf{x}_i^{\mathrm{T}} \boldsymbol{\beta} + \varepsilon_i, \qquad i = 1, \ldots, n,$$



선형 회귀

- 가장 간단하고 오래된 회귀용 선형 알고리즘으로 최소 제곱법으로도 불림

- 예측과 훈련 세트에 있는 목적 변수 y 사이의 평균제곱오차(MSE)를 최소화하는 파라미터 w와 b를 추적
  - » 평균제곱오차 → (예측 값과 목적 변수 값의 차이)^2 / 데이터 개수

$$MSE = 1/n \sum_{i=1}^{n} (y_i - \hat{f}(x_i))^2$$

- 매개변수가 없는 것이 장점이지만 이로 인해 모델의 복잡도를 제어할 방법도 없음

- 모델이 과대 적합된 경우 복잡도를 제어할 수 있는 모델 필요
  - » 기본 선형 회귀 대신 릿지 회귀와 라소 회귀 모델을 널리 사용

릿지 회귀

- 최소 제곱법에서 사용한 예측 함수를 사용하지만 릿지 회귀에서 가중치 선택은 훈련 데이터를 잘 예측하는 것뿐만 아니라 추가 제약 조건을 만족시키기 위한 목적도 포함
  - » 가중치의 절대 값을 최대한 작게 만드는 것 → w의 모든 원소를 0에 가깝게 만드는 것 (기울기를 작게 만드는 것) → 모든 특성이 출력에 주는 영향을 최소화
  - » 이런 제약을 규제(regularization)라고 하며 릿지 회귀에 사용되는 규제를 L2 규제라고 함

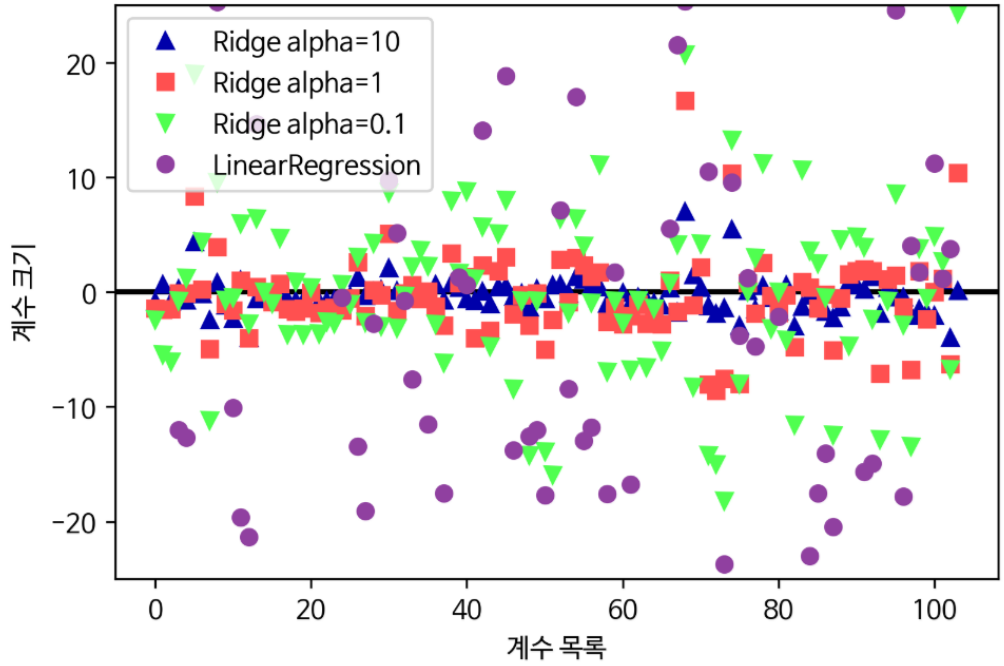
$$Cost\,(W) = RSS(W) + \lambda * (sum\ of\ squares\ of\ weights)$$

$$= \sum_{i=1}^{N} \left\{ y_i - \sum_{j=0}^{M} w_j\, x_{ij} \right\}^2 + \lambda \sum_{j=0}^{M} w_j^2$$

- scikit-learn의 Ridge 사용
  - » 알파 값을 크게 해서 규제의 강도를 높이면 일반화 성능이 향상됨

릿지 회귀 분석

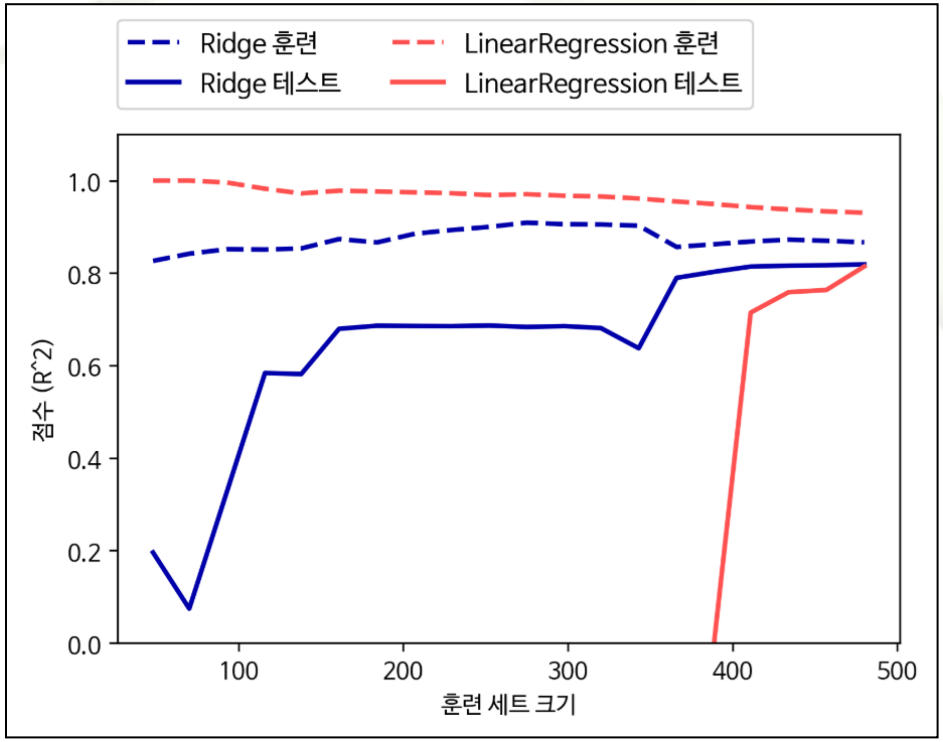
- 알파 값에 따른 가중치 계수 변경 추이



» 알파 값이 클수록 가중치 계수의 변동폭이 작아지는 것을 확인할 수 있음

릿지 회귀 분석

- 알파 값을 고정하고 데이터 세트의 크기 변경에 따른 모델 추이



훈련 세트의 크기가 커질수록 R^2 값 향상

» 데이터를 충분히 주면 규제 항의 중요도가 낮아져서 릿지 회귀와 선형 회귀의 성능이 같아질 것

라쏘(Lasso) 회귀

- 릿지 회귀와 같이 가중치 계수를 0에 가깝게 만드는 작업을 하지만 릿지 회귀와 다른 방식으로 처리 → L1 규제

- L1 규제의 결과로 어떤 가중치 계수는 실제 0이 되기도 함
  - » 모델에서 완전히 제외되는 특성이 발생
  - » 특성 선택이 자동으로 이루어지는 것으로 해석할 수 있음
  - » 일부 계수를 0으로 만들면 모델을 이해하기 쉬워지고 모델의 중요한 특성을 구분할 수 있음

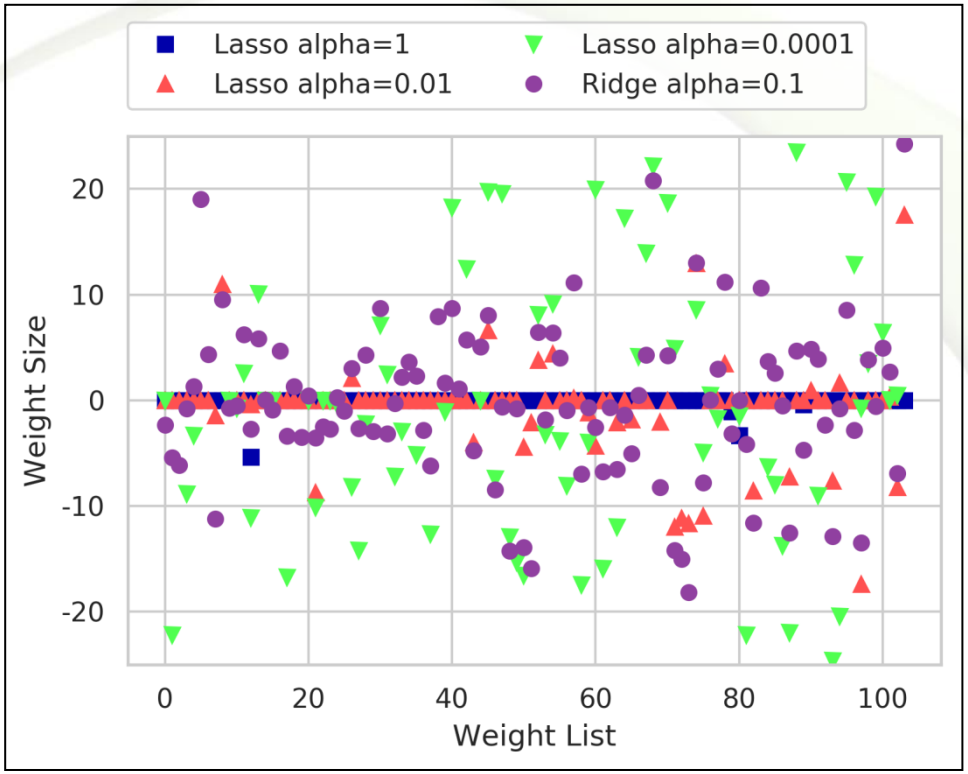
$$Cost\,(W) = RSS(W) + \lambda * (sum\ of\ absolute\ value\ of\ weights)$$

$$= \sum_{i=1}^{N} \left\{ y_i - \sum_{j=0}^{M} w_j\, x_{ij} \right\}^2 + \lambda \sum_{j=0}^{M} |w_j|$$

- scikit-learn의 Lasso 사용

라쏘 회귀 분석

- 알파 값의 변경에 따라 모델들의 가중치 계수의 추이 표시



» 알파 값이 작을수록 가중치 계수의 변동폭이 커지는 것을 확인할 수 있음

이진 분류용 선형 모델

- 주어진 속성의 선형결합을 바탕으로 분류 수행

- 이진 분류 선형 방정식

$$\hat{y} = w[0] \times x[0] + w[1] \times x[1] + \cdots + w[p] \times x[p] + b > 0$$

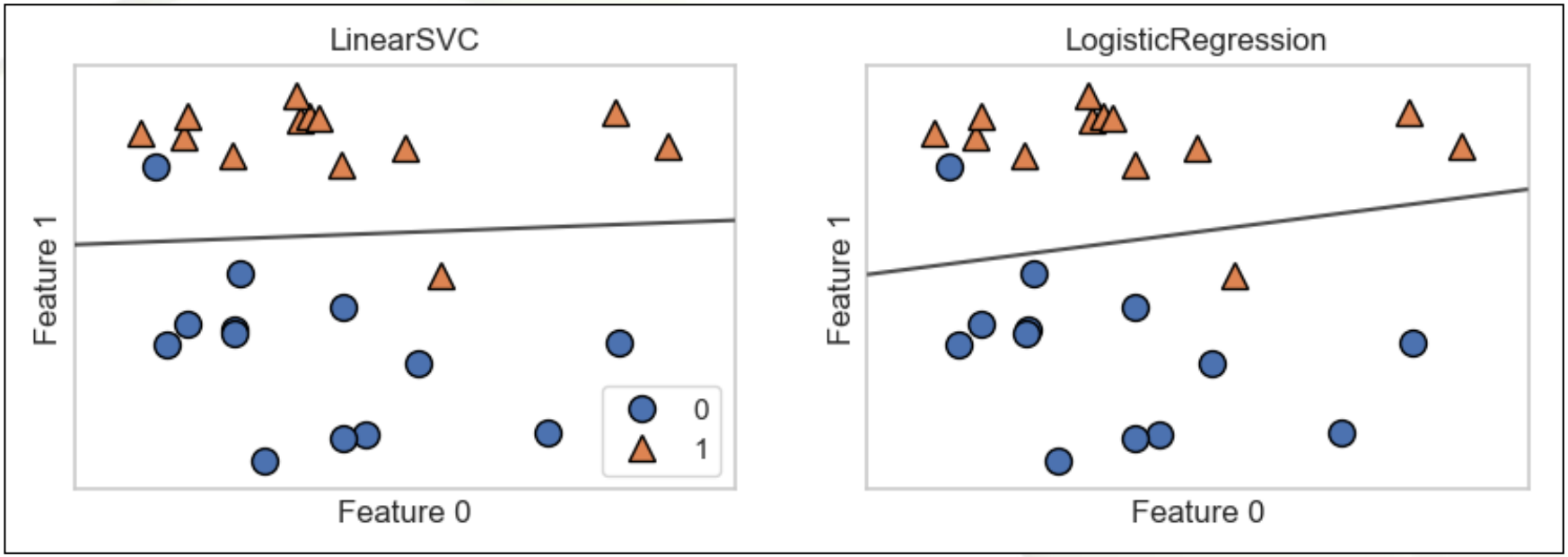
  » 결정 경계가 입력의 선형 함수 → 선, 평면, 초평면을 사용해서 두 개의 클래스를 구분하는 분류기

- 대표적인 선형 분류 알고리즘은
  » 로지스틱 회귀(Logistic Regression)와
  » 서포트 벡터 머신(Support Vector Machine, SVM)

- scikit-learn의 LogisticRegression과 LinearSVC 사용

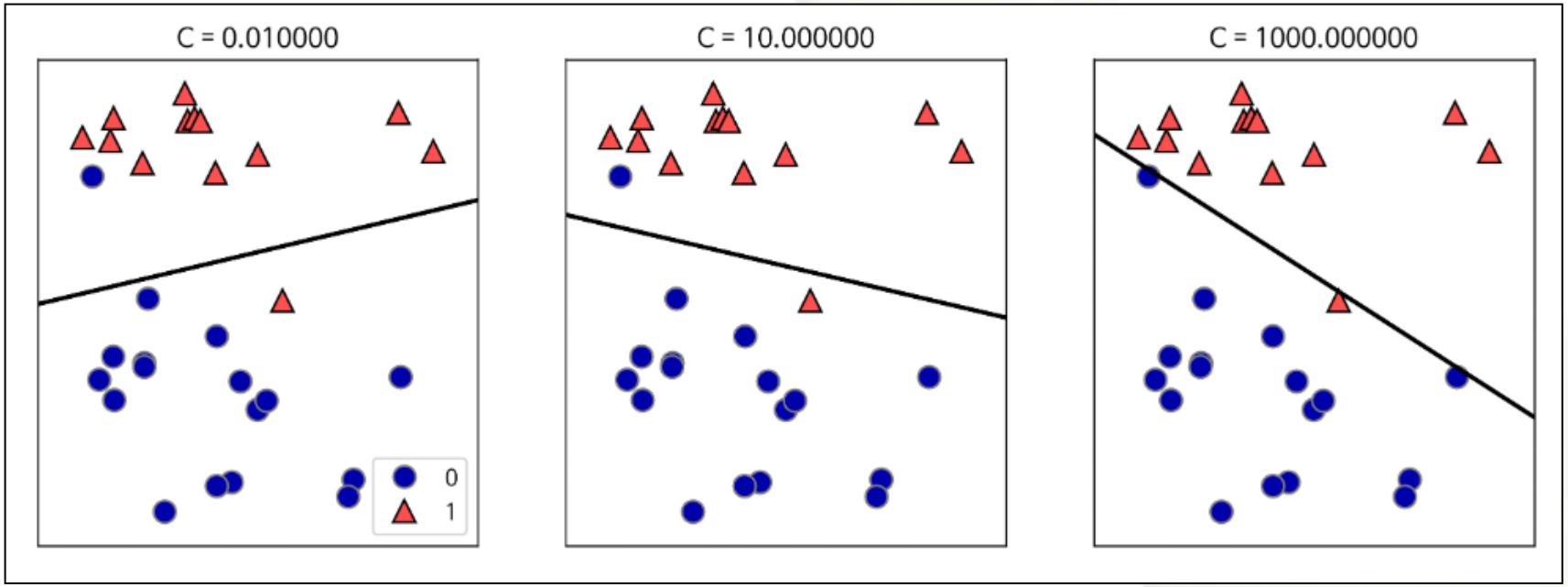
이진 분류용 선형 모델

- scikit-learn의 LogisticRegression과 LinearSVC



선형 분류 분석

- LinearSVC, LogisticRegression 선형 분류 모델의 규제
  - » 두 모델 모두 L2 규제 사용
  - » 규제 강도를 결정하는 매개변수는 C
    - › 이 값이 크면 훈련 세트에 최대한 맞추고 작으면 가중치 계수 w가 0에 가까워지도록 조정 (릿지와 라쏘의 알파 매개변수와 반대)



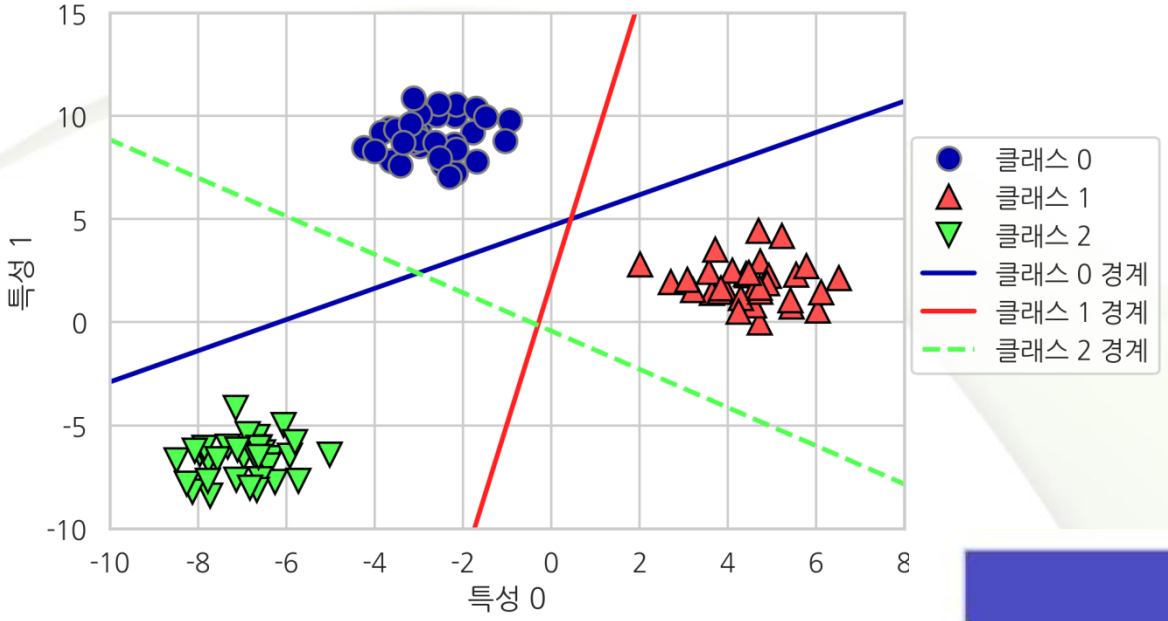
  - » C 값이 작은 왼쪽은 규제 많이 적용, C 값이 큰 오른쪽은 규제 적게 적용

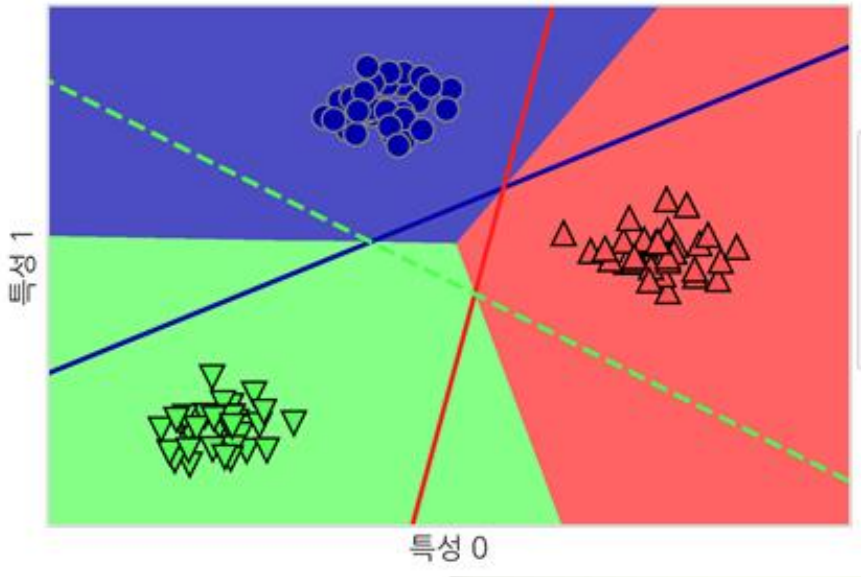
다중 클래스 분류용 선형 모델

- 로지스틱 회귀만 제외하고 많은 선형 분류 모델은 태생적으로 이진 분류만 지원

- 이진 분류 알고리즘을 다중 클래스 알고리즘으로 확장하는 보편적인 방법은 일대다(one-vs-rest) 방법
  - » 각 클래스를 다른 모든 클래스와 구분하도록 이진 분류 모델 학습
  - » 클래스의 수 만큼 이진 분류 모델 생성
  - » 모든 이진 분류기 중에서 가장 높은 점수를 내는 분류기의 클래스를 예측 값으로 사용

다중 클래스 분류기의 결정 경계





요약

- 선형 모델의 주요 매개변수는
  - » 회귀 모델에서는 알파
  - » LinearSVC와 LogisticRegressor에서는 C

- 알파 값이 클수록, C 값이 작을수록 모델이 단순해짐

- L1 규제와 L2 규제 선택 결정
  - » 기본적으로 L2 규제 사용
  - » 중요한 특성이 많지 않은 경우 L1 규제 사용

- 장점
  - » 학습 속도가 빠르고 예측도 빠름
  - » 매우 큰 데이터 세트와 희소한 데이터 세트에도 잘 동작

나이브 베이즈 분류기

나이브 베이즈를 사용한 분류

- 18세기 수학자 토마스 베이즈의 업적으로부터 유래

- LogisticRegression이나 LinearSVC 같은 선형 분류기보다 훈련 속도가 빠른 편이지만 일반화 성능은 다소 뒤지는 편

- 분류기 종류
  - » GuassianNB → 독립변수가 정규분포인 데이터에 적용
  - » BernoulliNB → 독립변수가 이항분포인 데이터에 적용
  - » MultinomialNB → 독립변수가 다항분포인 데이터에 적용

- BernoulliNB와 MultinomialNB는 대부분 텍스트 데이터 분석에 사용

나이브 베이즈를 사용한 분류

- 적용 사례
  - » 정크 이메일 필터링과 같은 문서 분류
  - » 침입자 검출 또는 컴퓨터 네트워크에서 이상 행동 검출
  - » 관찰된 증상을 고려한 질병 진찰

- 장단점
  - » 선형 모델의 장단점과 비슷

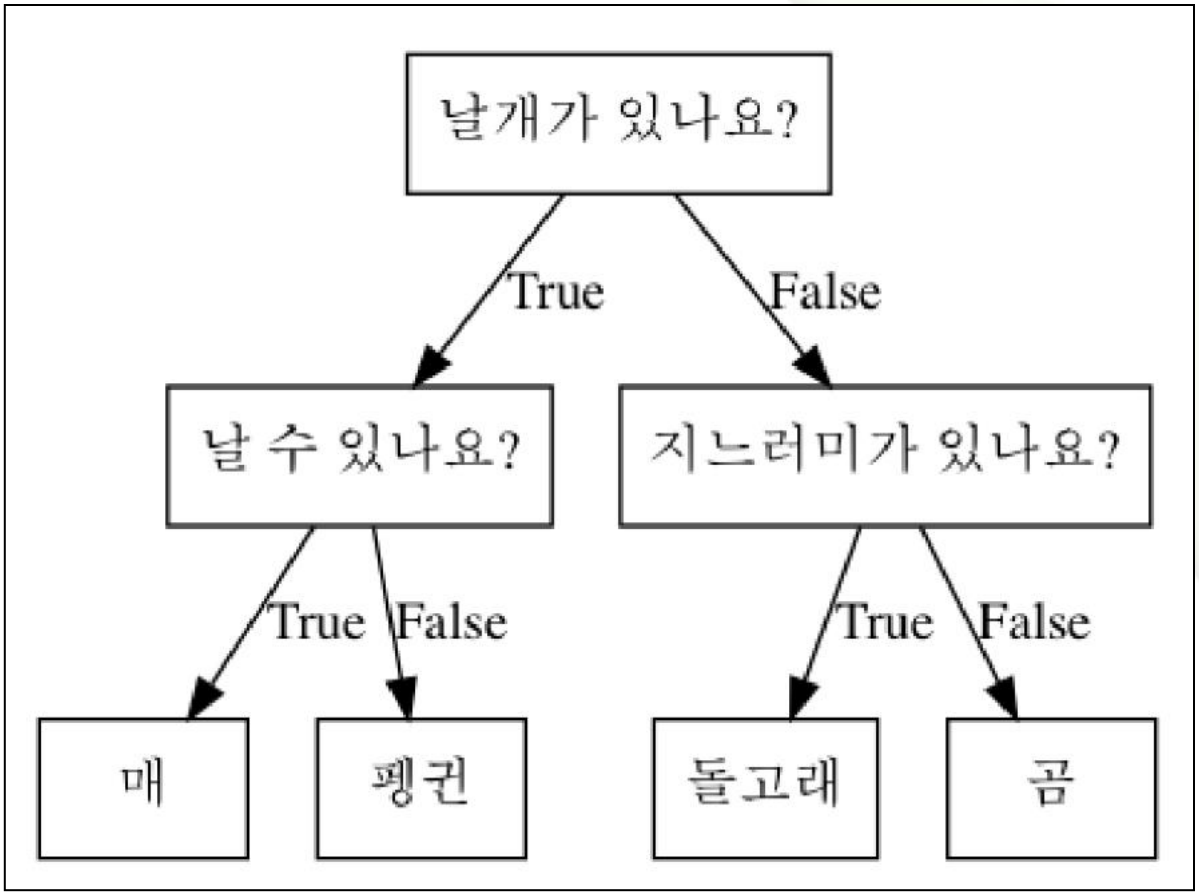
장점	단점
• 단순하고 빠르며 효과적 • 노이즈와 결측 데이터가 있어도 잘 수행됨 • 훈련 데이터의 양에 영향을 받지 않음(상대적으로 적은 사례 사용)	• 수치 속성으로 구성된 많은 데이터 세트에 대해 이상적이지 않음

결정 트리

결정 트리

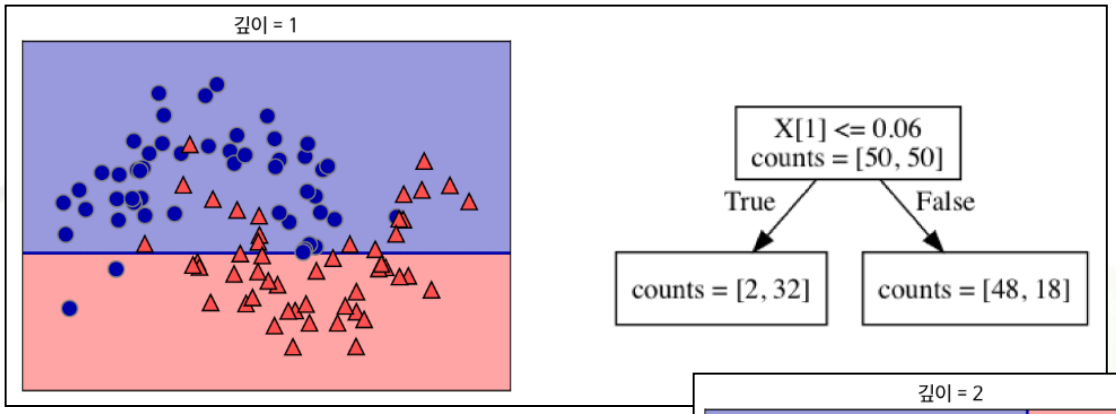
- 분류와 회귀에 광범위하게 사용되는 모델

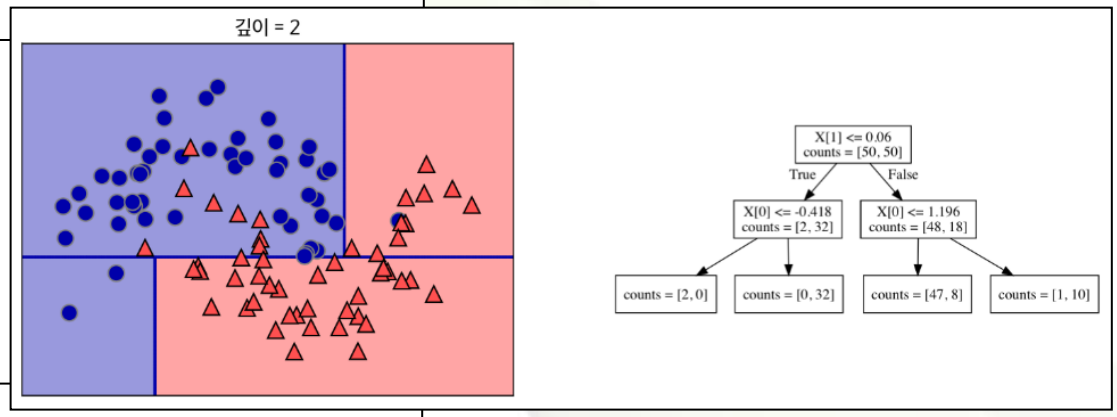
- 결정에 다다르기 위해 예/아니오 질문을 이어 나가면서 학습

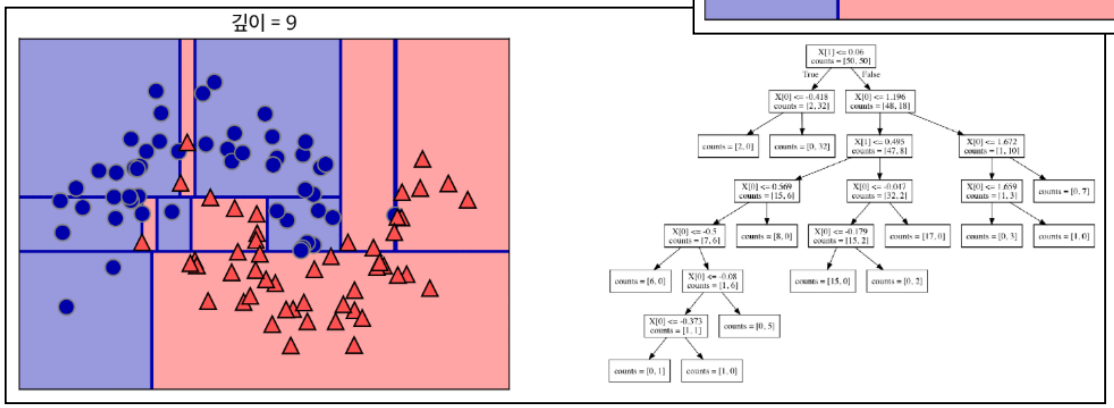


트리 구조의 모델 형성

결정 트리 만들기


깊이 = 1
X[1] <= 0.06
counts = [50, 50]
True
False
counts = [2, 32]
counts = [48, 18]


깊이 = 2
X[1] <= 0.06
counts = [50, 50]
True
False
X[0] <= -0.418
counts = [2, 32]
X[0] <= 1.196
counts = [48, 18]
counts = [2, 0]
counts = [0, 32]
counts = [47, 8]
counts = [1, 10]


깊이 = 9

복잡도 제어

- 순수 노드 → 하나의 타겟 클래스로 구성된 노드

- 모든 리프 노드가 순수 노드가 될 때까지 진행하면 모델이 매우 복잡해지고 훈련 데이터에 과대적합됨 → 순수 노드로 이루어진 트리는 훈련 데이터에 100% 정확하게 맞는 모델

- 과대적합을 막는 방법은
  - » 트리 생성을 일찍 중단하기 (사전 가지치기)
  - » 데이터 포인트가 적은 노드를 삭제하거나 병합 (사후 가지치기)

결정 트리 구현 및 분석

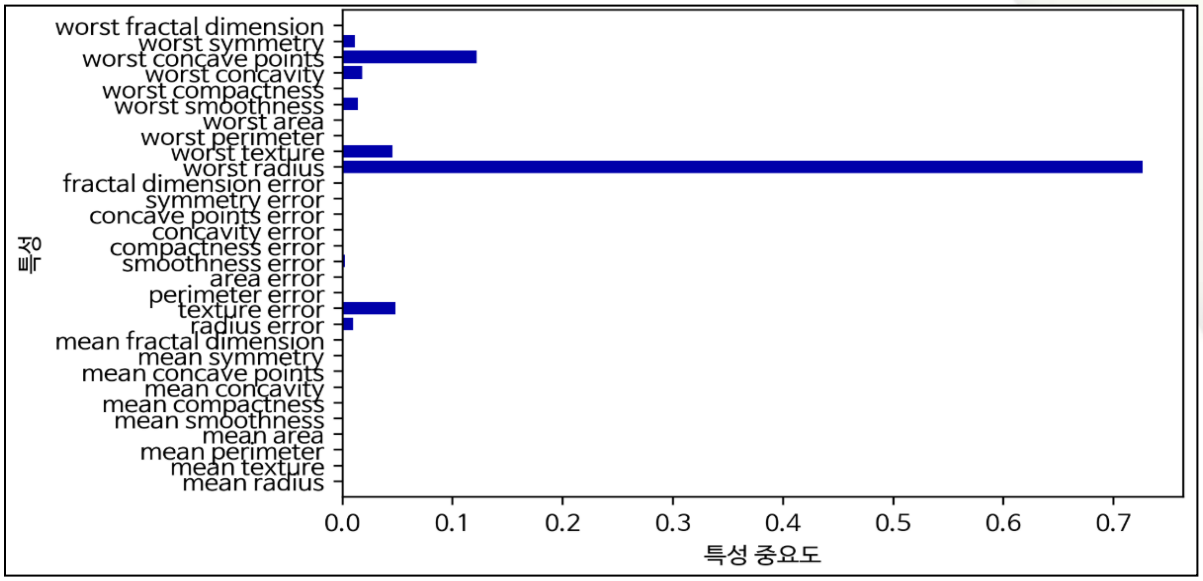
- scikit-learn의 DecisionTreeRegressor와 DecisionTreeClassifier 사용

- scikit-learn은 사전 가지치기만 지원
  - » max_depth, max_leaf_nodes, min_samples_leaf 등의 파라미터 사용

- graphviz 모듈을 사용해서 트리 시각화
  - » 알고리즘의 예측 프로세스를 이해할 수 있으며
  - » 비전문가에게 알고리즘 설명 쉬움

트리의 특성 중요도

- 전체 트리를 살펴보는 대신 트리가 어떻게 동작하는지 요약
- 가장 널리 사용되며 트리를 만드는 결정에 각 특성이 얼마나 중요한지 평가
- 0과 1사이의 숫자로 표현
  - » 0은 전혀 사용되지 않음 / 1은 완벽하게 목표 클래스 예측
  - » 특성 중요도의 전체 합은 1



트리의 특성 중요도

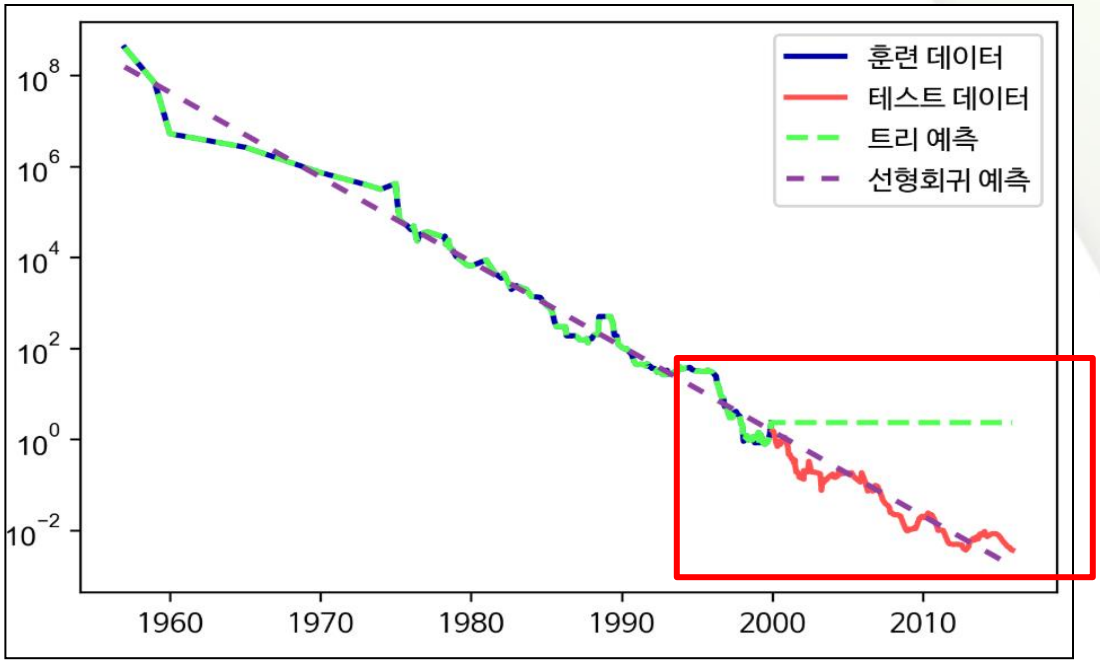
- 특성 중요도가 낮은 것이 특성이 유용하지 않은 것을 의미하는 것은 아님
  - » 트리가 그 특성을 선택하지 않았다는 의미 (다른 특성이 동일한 정보를 지니고 있기 때문일 수 있음)

- 특성 중요도의 값은 항상 양수
  - » 값이 큰 것은 중요도만 제시할 뿐이며 양성 / 음성을 판단하는데 사용할 수 없음

회귀 트리

- scikit-learn의 DecisionTreeRegressor 사용해서 구현
  - » 사용법과 분석 방법은 분류 트리와 비슷
  - » 리프 노드에 포함된 훈련 데이터의 평균 값이 출력 값

- 훈련 데이터의 범위 밖에 있는 데이터 포인트에 대해 예측 불가능
  - » 범위를 벗어나면 마지막 포인트를 이용해서 예측



장단점

- 장점
  - » 만들어진 모델을 쉽게 시각화할 수 있어서 비전문가도 이해하기 쉬움
  - » 데이터의 스케일에 구애받지 않음 → 정규화 또는 표준화 처리 불필요

- 단점
  - » 사전 가지치기를 사용해도 과대적합되는 경향이 강해서 일반화 성능이 좋지 않음 → 대안으로 앙상블 방법 사용

결정 트리 앙상블

앙상블 (Ensemble)

- 여러 머신러닝 모델을 연결해서 더 강력한 모델을 만드는 기법

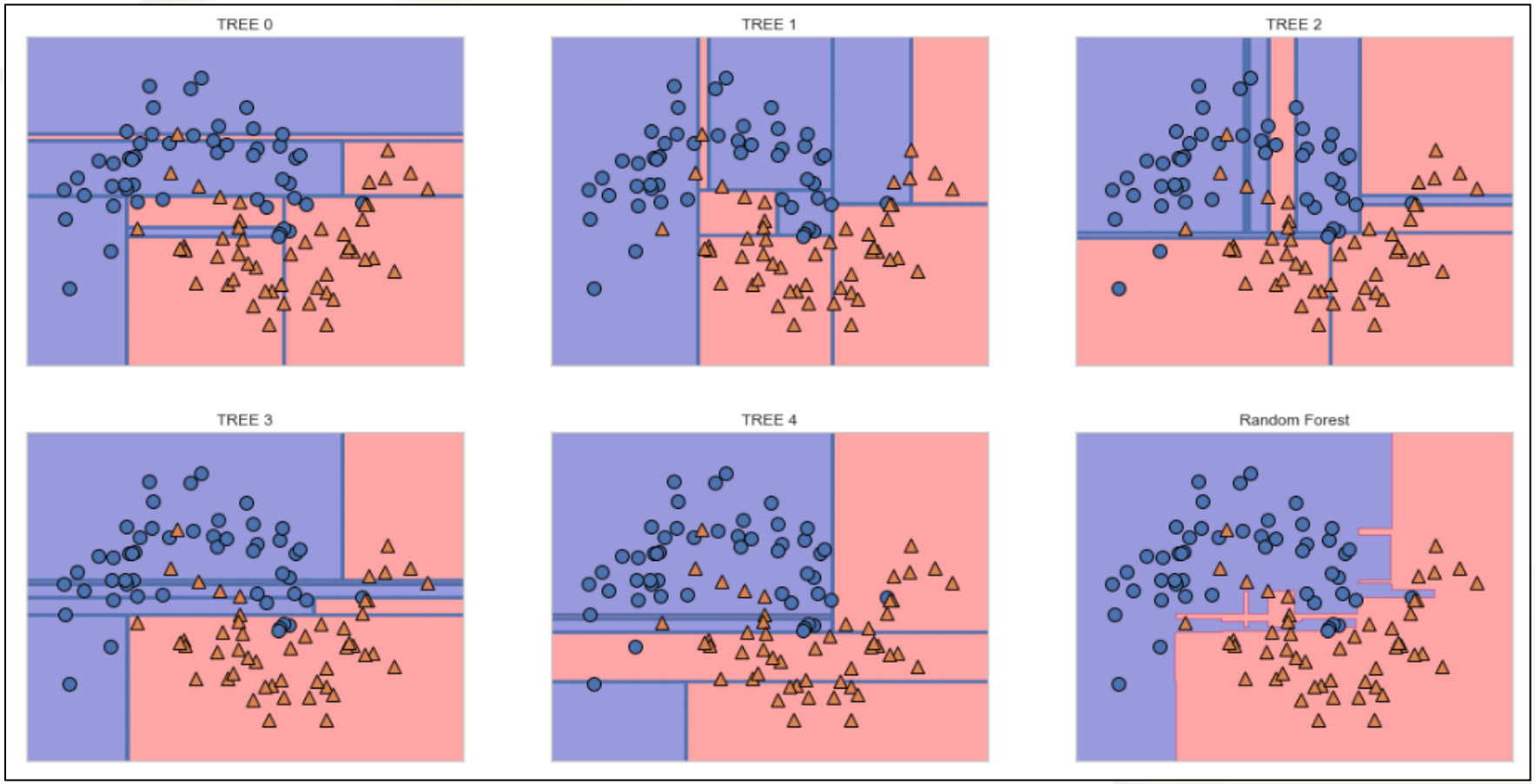
- 두 개의 모델이 분류와 회귀의 다양한 데이터 세트에서 효과적으로 동작
  - » 랜덤 포레스트 (Random Forest)
  - » 그레디언트 부스팅 결정 트리 (Gradient Boosting Decision Tree)

랜덤 포레스트 (Random Forest)

- 결정트리의 주요 단점인 훈련 데이터에 과대적합되는 경향을 회피하는 방법
- 조금씩 다른 여러 결정트리의 묶음
- 기본적으로 예측력이 좋으면서 서로 다른 방향으로 과대적합된 트리를 많이 만들어 그 결과를 평균 내면 과대적합된 양을 줄일 수 있다는 것이 수학적으로 검증됨
- 트리들이 서로 달라지도록 트리 생성시 무작위성 주입
  - » 데이터 포인트를 무작위로 선택하는 방법
  - » 분할 테스트에서 특성을 무작위로 선택하는 방법

랜덤 포레스트 (Random Forest)

- 개별 트리와 Random Forest

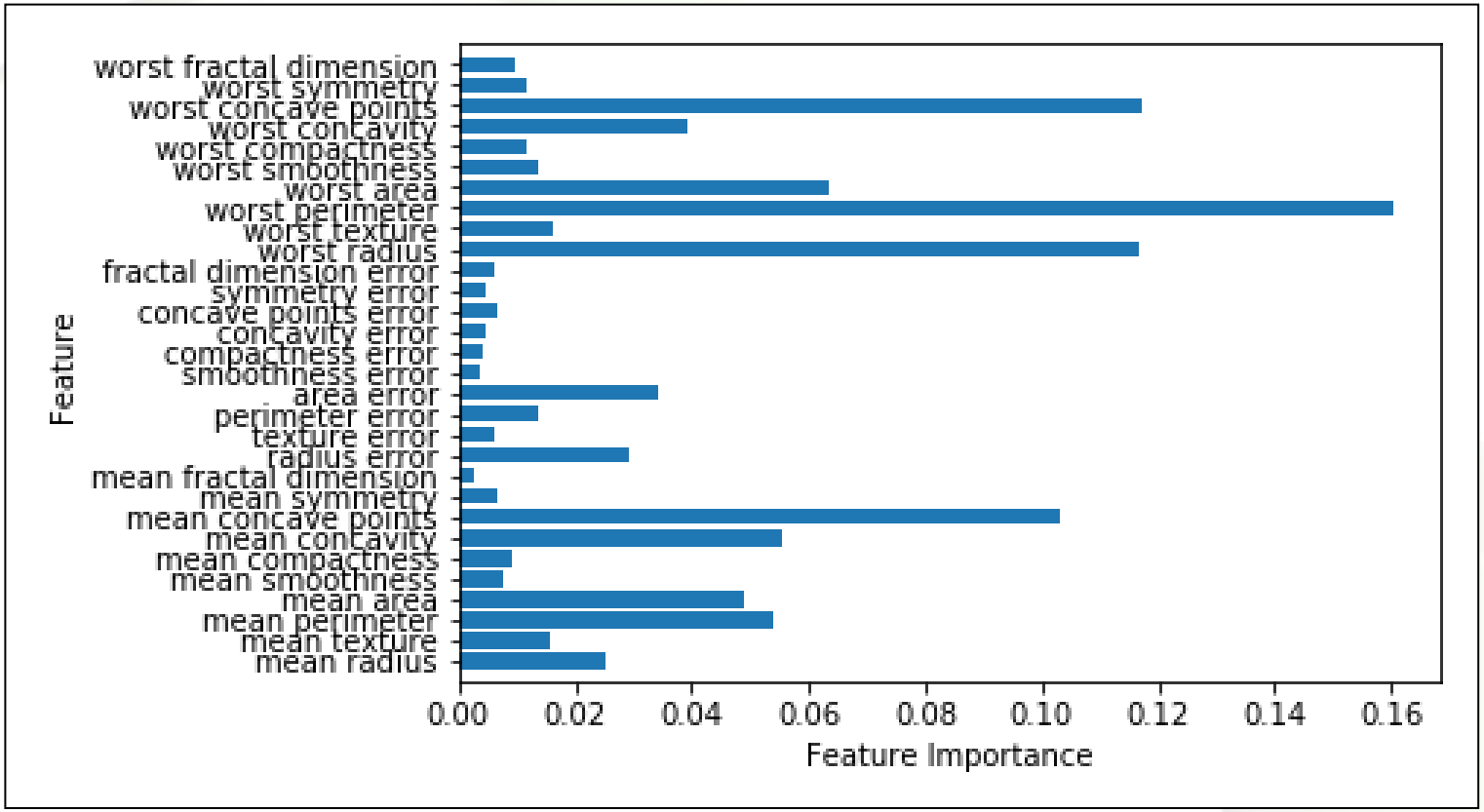


랜덤 포레스트 구축

- 생성할 트리 개수 결정
  - » RandomForestRegressor, RandomForestClassifier의 n_estimators 매개 변수
- 데이터의 부트스트랩 샘플 생성
  - » 원래 데이터 세트의 크기와 같지만 누락 및 중복을 허용하는 무작위 데이터 추출
- 전체 특성으로 테스트하지 않고 각 노드에서 후보 특성을 무작위로 선택한 후 이 후보들 중에서 최선의 테스트 도출
- 모든 트리의 예측을 만든 후
  - » 회귀의 경우에는 이 예측들을 평균하여 최종 예측 도출
  - » 분류의 경우에는 약한 투표 전략 사용 → 가능성 있는 출력 레이블의 확률 제공 → 가장 높은 확률을 가진 클래스가 예측 범주

랜덤 포레스트 구축

- 속성별 중요도 정보 도출 가능



랜덤 포레스트 (Random Forest)

- 장점
  - » 성능이 매우 뛰어나고
  - » 매개변수 튜닝을 많이 하지 않아도 잘 작동하며
  - » 데이터의 스케일을 맞출 필요도 없음

- 단점
  - » 텍스트 데이터와 같이 매우 차원이 높고 희소한 데이터에는 잘 작동하지 않음 → 선형 모델이 더 적합
  - » 선형 모델에 비해 많은 메모리를 사용하며 훈련과 예측이 느림

그래디언트 부스팅 결정 트리

- 여러 개의 결정 트리를 묶어 강력한 모델을 만드는 방법 (랜덤 포레스트와 동일) → 회귀와 분류 모두에 사용 가능

- 이전 트리의 오차를 보완하는 방식으로 순차적으로 트리 생성 (랜덤 포레스트와 차이)
  - » 이전 트리의 오차를 얼마나 강하게 보정할 것인지 설정 (learning_rate)

- 무작위성 없음 → 대신 과적합화를 막기 위해 강력한 사전 가지치기 사용

- 1 ~ 5 정도의 낮은 트리를 사용하기 때문에 메모리 사용량이 적고 예측도 빠름

- 트리가 많이 추가될수록 예측 성능 향상됨

그레디언트 부스팅 결정 트리

- 장점
  - » 특성의 스케일 조정 필요 없음

- 단점
  - » 매개변수를 잘 조정해야 의미 있는 결과 도출
  - » 훈련 시간이 오래 걸림
  - » 희소한 고차원 데이터에 대해 잘 작동하지 않음

커널 서포트 벡터 머신

커널 서포트 벡터 머신

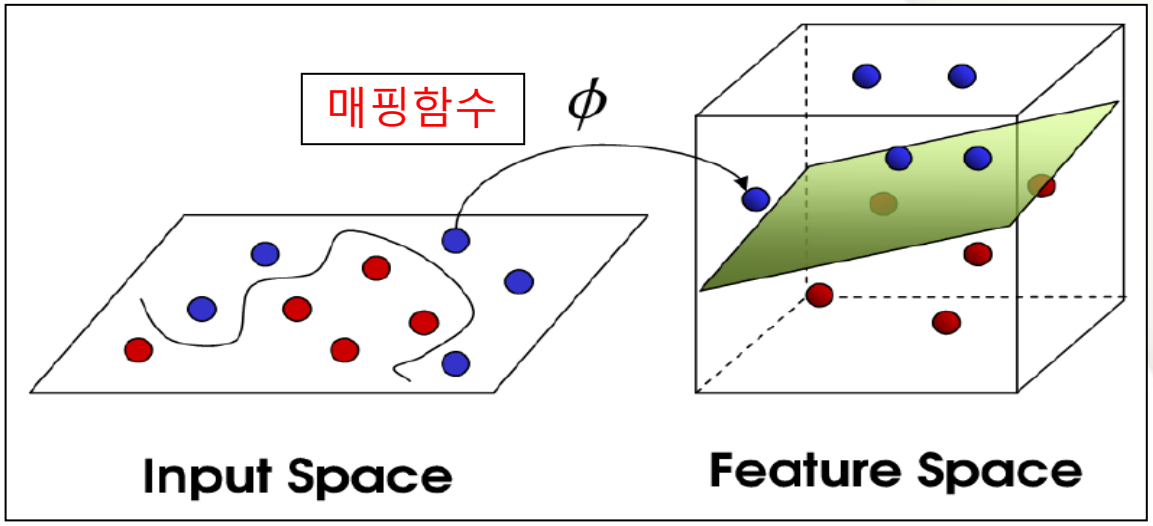
- 입력 데이터가 단순한 초평면(hyperplane)으로 정의되지 않는 더 복잡한 모델을 만들 수 있도록 확장

- 분류와 회귀에 적용 가능

선형 모델의 비선형 특성

- 직선과 초평면은 유연하지 못해서 저차원 데이터 세트에서 선형 모델이 매우 제한적

- 선형 모델을 유연하게 만들기 위해 특성끼리 곱하거나 거듭제곱 하는 방식으로 새로운 특성을 추가하는 방법 사용



커널 기법

- 비선형 특성을 추가해서 선형 모델을 강력하게 만드는 경우 추가할 특성을 선택하는 문제와 많은 특성을 추가했을 때 연산 비용 문제 발생

- 커널 기법을 사용하면 수학적 기교를 통해 새로운 특성을 많이 만들지 않고도 고차원에서 분류기 학습 가능

- 데이터를 고차원에 매핑할 때 사용하는 방법
  - » 다항식 커널
  - » RBF(Radial Basis Function) 커널

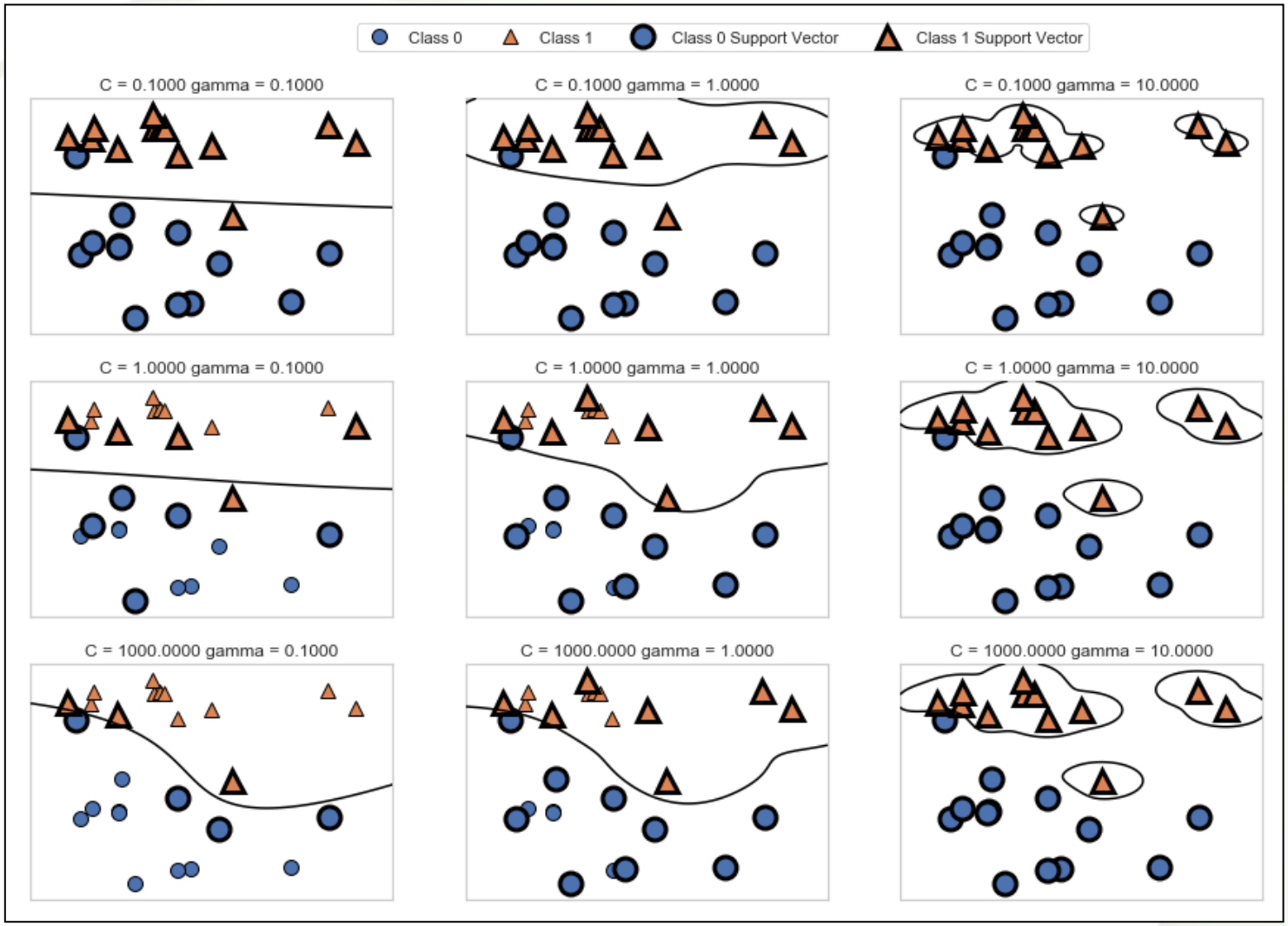
SVM 매개변수 튜닝

- 감마(gamma) 매개변수
  - » 가우시안 커널 폭의 역수
  - » 감마 매개변수는 하나의 훈련 샘플이 미치는 영향의 범위를 결정
    - › 작은 값은 넓은 영역을 의미하며 큰 값은 영향을 미치는 범위가 제한적
    - › 가우시안 커널의 반경이 클수록 훈련 샘플의 영향 범위도 커짐

- C 매개변수
  - » 선형 모델에서 사용한 것과 유사한 규제 매개변수
  - » 각 포인트의 중요도 제한
  - » C가 작으면 제약이 매우 큰 모델이 만들어지고 데이터 포인트의 영향 감소.

SVM 매개변수 튜닝

- 감마(gamma) 매개변수와 C 매개변수에 따른 분류 모델 비교



SVM 데이터 전처리

- 특성 스케일에 매우 민감해서 입력 특성의 범위를 비슷하게 만들어야 함
- 일반적으로 모든 특성 값을 0 ~ 1 범위에 맞추는 방법을 많이 사용

SVM 장점과 단점

- 데이터의 특성이 적어도 복잡한 결정 경계 도출 가능
- (특성이 적은) 저차원 및 (특성이 많은) 고차원 데이터 모두에 잘 동작
- 샘플이 많은 경우에는 비효율적
  - » 10,000개 정도는 모델이 잘 작동
  - » 100,000개 정도는 속도와 메모리에 문제 노출
- 데이터 전처리와 매개변수 튜닝에 신경을 많이 써야 함

분류 예측 불확실성

분류 예측의 불확실성 추정

- 어떤 테스트 데이터에 대해 분류기가 예측한 클래스에 대한 확실성 정도

- scikit-learn은 이를 위해 decision_function과 predict_proba 두 개의 함수 제공
  - » 대부분의 분류 클래스는 적어도 둘 중 하나 또는 둘 모두 제공

결정 함수 (decision_function)

- 이진 분류에서 반환 값의 크기는 (n_samples,)로 각 샘플이 하나의 실수 값을 반환

- 반환 값
  - » 반환된 값은 모델이 각 데이터 포인트에 대해 양성 클래스 1에 속한다고 믿는 정도
  - » 값의 범위는 데이터와 모델 파라미터에 따라 달라짐

예측 확률 (predict_proba)

- 이진 분류에서 반환 값의 크기는 (n_samples, 2)

- 반환 값
  - » 반환된 값(튜플)의 첫 번째 원소는 첫 번째 클래스의 예측 확률이고 두 번째 원소는 두 번째 클래스의 예측 확률
  - » 값의 범위는 0 ~ 1
  - » 두 원소 값의 합은 1
  - » 두 원소 중 50% 이상의 확신을 가진 값이 예측 값

- 일반적으로 복잡도가 낮은 모델은 예측에 불확실성이 더 높음

- 모델 보정 → 불확실성과 모델의 정확도가 동등하도록 조정

다중 분류의 불확실성

- decision_function과 predict_proba는 이진 분류 뿐만 아니라 다중 분류에도 사용 가능

- 다중 분류의 decision_function 결과 값 크기는 (n_samples, n_classes)
  - » 각 열은 각 클래스에 대한 확신 점수 → 수치가 크면 가능성이 높아짐
  - » 가장 값이 큰 열이 모델의 예측 값