지도학습의 대표적 2가지 방법

분류(Classification)

: 미리 정의되고 가능성 있는 여러 클래스 레이블 중 하나로 예측

두 개 클래스 레이블: 이진 분류(Binary Classification)

셋 이상: 다중 분류(Multiclass Classification)

회귀(Regression)

: 연속적인 숫자 또는 실수 예측

kNN(k-Nearest Neighbors algorithm)

kNN 분류(classification)

[sklearn.neigbors.KNeighborsClassifier()]

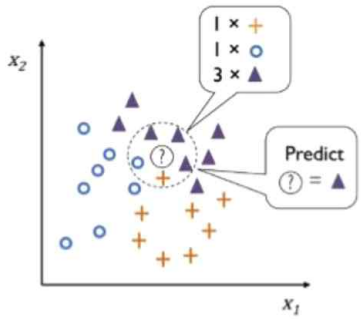
: 주변 k개의 자료의 클래스 중 가장 많은 클래스로 분류 예측

Training data 자체가 모형일 뿐

추정 방법과 모형 없음

= 데이터 분포 표현을 위한 파라미터 없음

매우 간단하지만 performance는 떨어지지 않음



게으른 학습(lazy learner)

= 사례중심학습(instance-based learning)

: 판별 함수 학습없이 훈련 데이터셋를 메모리에 저장

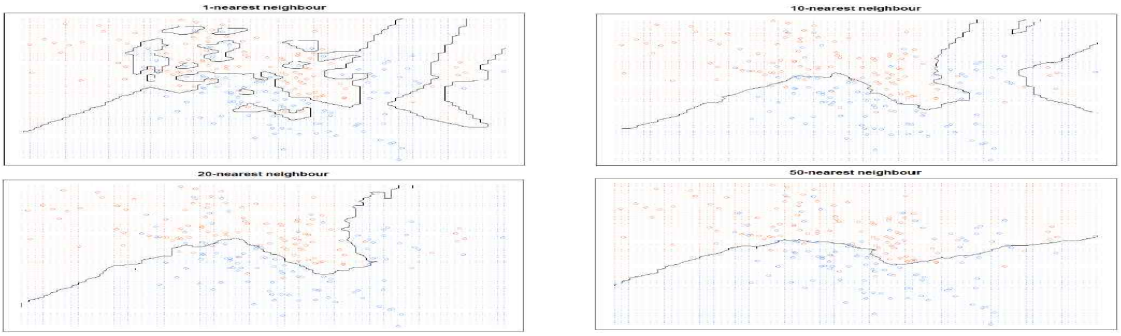
데이터 차원 증가하면 차원의 저주(curse of dimension) 문제 발생

차원 증가 → kNN 성능 저하

차원↑ → 공간의 크기(부피)↑↑ → 동일 개수 데이터 밀도↓↓

Minkowski 거리: $d(x_i, x_j) = \sqrt[p]{\sum_{k=1}^{d} |x_{ik} - x_{jk}|^p}$

k가 너무 작으면: 과대적합(overfitting) k가 너무 크면: 과소적합(underfitting)



분류 예측 방법

다수결 방식(majority voting)

: 이웃 클래스 가운데 빈도가 가장 많은 클래스로 예측

가중합 방식(weighted voting)

: 가까운 이웃의 정보에 높은 가중치 부여

장점

1. 학습 데이터에 있는 노이즈 영향을 크게 받지 않는다
2. 학습 데이터가 충분히 많다면 효과적이다
3. 마할라노비스 거리와 같이 데이터 분산 고려하는 경우 매우 강건(robust)하다

마할라노비스 거리(Mahalaobis distance)

: 평균과의 거리가 표준편차의 몇 배인지 나타내는 값

얼마나 일어나기 힘든 값인지 얼마나 이상한 값인지 수치화하는 방법

단점

1. 최적 이웃 개수(k)와 어떤 거리 척도(distance metric)가 분석에 적합한지 불분명해서 데이터 각각의 특성에 맞게 임의로 선정해야 한다

적절한 k는 데이터마다 다르므로 탐욕적인 방식(Grid Search)으로 탐색

2. 새로운 관측치와 각각의 학습데이터 사이의 거리를 전부 측정해야 하므로 계산 시간이 오래 걸린다

kNN의 복잡성을 줄이려는 시도: Locally Sensitive Hashing, Network based Indexer, Optimized product quantization

기계학습의 일반적인 실습 순서

1. 데이터셋 불러오기

seaborn 라이브러리

2. 데이터셋 카테고리 실수화

※DictVectorize클래스(One-hot Encoding) vs LabelEncoder클래스(범주형 라벨)

3. 데이터 분할

학습 데이터와 테스트 데이터로 나누기

4. (옵션) 입력 데이터의 표준화
5. 모형 추정 혹은 사례중심학습
6. 결과 분석

혼합행렬(confusion matrix)로 확인

데이터셋 불러오기

Seaborn 라이브러리

파이썬에서 데이터 시각화 담당하는 모듈

유익한 통계 그래픽을 그리기 위한 고급 인터페이스 제공

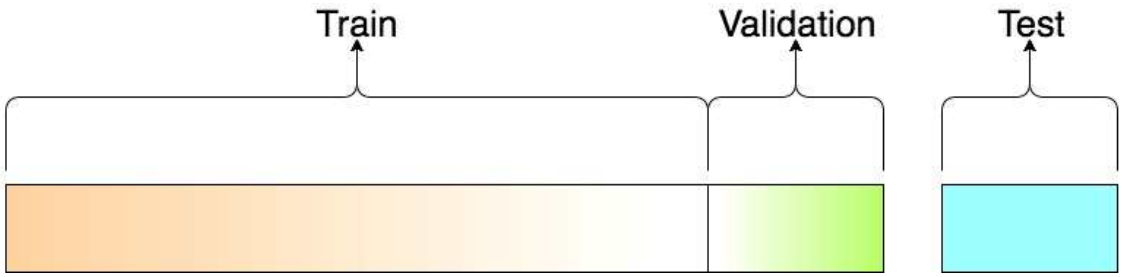
변수와 파라미터 조정으로 그래프 표현

유명한 데이터셋 대부분 지원

데이터 분할

학습 데이터(train)와 테스트 데이터(test)가 서로 겹치지 않게 나누기

목적: 학습 데이터로 학습시키고 학습에 사용되지 않은 테스트 데이터에 적용해서 학습 결과의 일반화(generalization)가 가능한지 알아본다



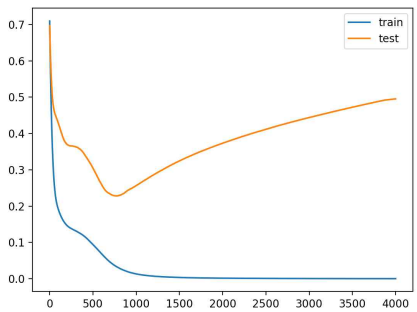
sklearn.model_selection.train_test_split()

test_size: 테스트셋의 구성비율(train_size와 반대) default = 0.25

shuffle: split 이전에 섞을건지 default = True

stratify: target이면 각각의 클래스 비율을 train/validation에 유지
한쪽으로 쏠리는 것 방지 dafault = None

random_state: 랜덤 시드값



모델이 과적합 되었다면 validation 셋으로 검증할 때 예측율이나 오차율이 떨어진다
이런 현상이 나타나면 학습 종료한다.

결과 분석

분류 성능 평가 방법

★ 1. sklearn.metrics.confusion_matrix(y_true, y_pred)

★ 2. sklearn.metrics.accuracy_score(y_ture, y_pred)

3. sklearn.metrics.precision_score(y_true, y_pred)

4. sklearn.metrics.recall_score(y_true, y_pred)

5. sklearn.metrics.fbeta_score(y_true, y_pred)

6. sklearn.metrics.f1_score(y_true, y_pred)

7. sklearn.metrics.roc_curve(y_true, y_pred)

8. sklearn.metrics.auc(x, y)

혼합행렬(confusion matrix)

:타겟의 원래 클래스와 모델이 예측한 클래스가 얼마나 일치하는지(개수) 나타낸 표

	예측 클래스 0	예측 클래스 1	예측 클래스 2
정답 클래스 0	샘플의 수	샘플의 수	샘플의 수
정답 클래스 1	샘플의 수	샘플의 수	샘플의 수
정답 클래스 2	샘플의 수	샘플의 수	샘플의 수

	양성으로 예측	음성으로 예측
실제 양성	True Positive	False Negative
실제 음성	False Positive	True Negative

정확도(accuracy)

: 전체 샘플 중 맞게 예측한 샘플의 비율

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

정밀도(precision)

: 양성 클래스로 예측한 샘플 중 실제로 양성 클래스에 속하는 샘플의 비율

$$precision = \frac{TP}{TP + FP}$$

제현율(recall)

: 실제 양성 클래스에 속하는 샘플 중 양성 클래스로 예측한 샘플의 비율

$$recall = \frac{TP}{TP + FN}$$

(옵션) 입력데이터의 표준화

표준화: 자료의 단위에 의해 영향받지 않도록 하는 과정

sklearn.preprocessiong.StandardScaler()

테스트 데이터의 표준화: 학습 데이터에서 구한 특성 변수의 평균과 표준편차 이용

※테스트 데이터는 입력데이터의 표준화(평균, 표준편차)에 사용하지 않는다

표준화는 일반적으로 머신러닝 성능을 향상시키지만

데이터의 분포(통계적 특성)이 깨지면 머신러닝의 학습 저하를 가져온다

∴ 표준화 여부는 테스트 데이터의 정밀도를 점검하여 결정해야 한다.

kNN 회귀(regression)

sklearn.neighbors.KNeighborsRegressor()

k개의 관측치 (x_i, y_j)에서 $\bar{y}$를 계산하여 적합치로 사용

y의 예측치: k개의 특성 변수 x에 대응하는 y의 평균을 구한다

$$\bar{y} = \frac{1}{k}\sum_{i=1}^{k} y_i$$ (y_i는 가장 가까운 k개의 학습 데이터)

회귀 예측 방법

단순 회귀

: 가까운 이웃들의 단순한 평균을 구한다

가중 회귀(weighted regression)

: 각 이웃이 얼마나 가까이 있는지에 따라 가중 평균(wighted average)을 구해서 거리가 가까울수록 데이터가 더 유사할 것으로 보고 가중치를 부여한다

x	y	distance
A	5.0	3.2
B	6.8	11.5
C	9.0	1.1

단순평균: $\frac{3.2+11.5+1.1}{3} = 6.93$

가중 평균: $\frac{\frac{5.0}{3.2}+\frac{6.8}{11.5}+\frac{9.0}{1.1}}{\frac{1}{3.2}+\frac{1}{11.5}+\frac{1}{1.1}} = 7.9$