**스키마 정리 \*세부코드는 나중에 하고 일단 주요 개념 및 과정**

* 데이터분석 개념:
  + 딥러닝 Ⅽ 머신러닝 Ⅽ 인공지능
  + 정형 데이터: 행과 열이 잘 구분되는 데이터(csv, db, 엑셀 등에 담을 수 있는 레코드)
    - 머신러닝에서 많이 사용됨
  + 비정형 데이터: 텍스트, 오디오, 이미지, 영상 등
    - 딥러닝에서 많이 사용됨
  + 차원:
    - 벡터: 벡터에 잇는 원소의 개수를 의미
    - 배열: 축의 개수를 dmlal
* 알아두면 좋은 파이썬 코드들:
  + 코드 맨 앞에 !을 붙이면 shell 명령 프롬프트에서 해당 코드 수행
    - !wget는 shell 프롬프트에서 파일을 불러오라고 명령하는 것
  + 3차원 배열의 이해
    - [[[1, 2, 3], [4, 5, 6]], 맨 위의 2차원배열: [[1,2,3],

[[7, 8, 9], [10, 11, 12]], [4,5,6]]

[[13, 14, 15], [16, 17, 18]]]

* (3,2,3)의 3차원 배열 = 3개의 2차원 배열(2개의 행과 3개의 열)
  + 여래개의 plot을 한번에 출력하기
    - Fig, axs = plt.subplots(행,열, figsize = (width,height))

Axs[그래프의 인덱스].플롯종류(x축, y축)

\*웬만한 그래프들과 imshow와 같은 이미지도 출력가능

* + - * 행과 열에 지정한 개수만큼 그래프의 인덱스를 부여하여 생성
      * For루프를 활용하여 axs[i, j].플롯종류()를 하면 i의 range와 j의 range에따라 플롯 출력
  + Np.celi(\_/\_)
    - 나머지가 있는 경우 몫에 1을 추가하여 출력
* 시각화
  + 산점도
    - 용도:
      * 샘플링(훈련 데이터와 테스트 데이터)이 잘 진행되었는 지 확인
      * 샘플의 분포 확인(특성이 2개 일 때)
    - 산점도 생성방법
    - 주요 데이터 포인트 형태 바꾸기
    - 산점도에서의 x축과 y축 스케일을 맞추는 방법 \*거리관련 모델링 시각화 시에 중요
* 데이터 전처리
  + .inf() / .describe() / .shape를 통해 데이터의 형태 및 특징 탐색
    - .shape : NumPy 배열 또는 다차원 배열의 차원을 나타내는 속성
  + Null값 대처법(경우에 따라 적합한 수단 사용):
    - 전체 데이터의 평균을 넣어 null 값 대체
    - 중간값 사용
    - 열 완전히 삭제
    - 빈도가 가장 높은 값으로 NA 대체 (분류 or 범주형 데이터일 경우)
  + 사이킷런에 적합한 데이터 형태로 변환(2차원 배열) ex) [[~~~]]
    - 넘파이로 데이터 준비:
      * Np.column\_stack(특성들)
    - 아니면 .reshape(행, 열) 함수 사용
    - 데이터프레임.to\_numpy()를 사용하여 데이터프레임을 넘파이 배열(2차원배열)로 변환할 수 있음
  + 훈련세트와 테스트 세트 구분

\*샘플링 편향이 일어나지 않게 조심 = 데이터가 골고루 섞인 채로 나눠지도록

* + - 사이킷런으로 데이터 나누기
      * From sklearn.model\_selection import train\_test\_split
    - **교차검증(cross\_validate):** 훈련세트, 검증세트, 테스트 세가지로 분류하는 분할기를 사용하여 데이터 나누기
      * 원리: 일차적으로 훈련세트와 테스트 세트를 나눈 후 cross\_validate 클래스를 사용하여 훈련세트를 지정한 개수의 폴드로 나눈다. 이때 폴드 중 하나는 검증세트로 나머지 폴드들을 훈련시키고 검증세트에 적용한다. 그렇게 폴드 개수만큼 도출되는 검증세트의 점수를 평균화하여 최종 검증 스코어를 도출. 이후에 테스트 세트에 적용연습
      * 매개변수들:
        + Cv = :을 통해 폴드의 개수지정 가능

**분할기를 사용한 교차검증:** 각 폴드에 대한 추가설정

사용방법:회귀일 경우 kfold라는 split 모델 사용 / 분류일 경우 StratifiedKfold 모델 사용

매개변수들:

N\_splits = : 폴드의 개수 지정

Shuffle = True: 폴드 안의 클래스들이 무작위로 썩여 들어가도록 함

* + - * 출력값들(딕셔너리들을 모아놓은 리스트 형식으로 출력됨):
        + Fit\_time: 훈련하는 데 걸리는 시간
        + Score\_time: 점수도출하는 데 나오는 시간
        + Test\_score: 검증세트들 각각의 검증점수
        + Np.mean(scores[‘test\_score’]\_\_ 검증점수 평균화
  + 타겟데이터 설정
    - 이진분류:
      * 0과 1로 타겟 지정
  + 표준 점수로 훈련데이터 변환하기(다양한 데이터의 기준을 맞추는 과정)
    - (훈련샘플 – 훈련샘플의 평균 ) / 훈련샘플의 표준편차 = z정수 = 표준점수
* 머신러닝
  + 관련용어:
    - Class: 분류가 되는 것들
    - Classification: 분류과정
    - Binary classification: 두가지로 분류하는 과정
  + 기법들:
    - 분류(Classifier)
      * 지도학습
        + **K-최근접이웃** : 거리에 따라 가장 가까운n개의 샘플 중 다수결의 클래스로 분류

Sklearn 라이브러리에서 클래스 불러오기

Kn을 클래스 객체로 지정

매개변수 지정

N\_neighbors = : 예측 기준이 되는 이웃 개수 지정

훈련데이터와 타겟데이터에 따라 훈련

테스트데이터와 타겟데이터에 따라 점수내기

새로운 데이터를 훈련된 모델을 통해 예측하기

* + - * + **결정트리(DecisionTreeRegressor)**

주의사항

하나의 스케일 내에서 분류되기 때문에 특성들의 스케일을 표준화할 필요가 없음

관련용어:

Root node: 맨위의 노드

Leaf node: 여러 과정을 거쳐 맨 마지막 단계에 도달했을 떄의 노드

부모노드: 두개의 노드 중 위의 노드

자식노드: 두개의 노드 중 밑의 노드

매개변수들:

Max\_features = : 지정한 숫자에 따라 일부 특성 무작위로 사용 가능 (default = None = 모든 특성 사용)

Plot\_tree() : 트리의 시각화

매개변수들

Max\_depth = : 루트노드 밑에 몇 개의 노드를 출력할 지 지정할 수 있음

가지치기: 순수노드가 될 때까지 분화하여 과대적합이 이루어지는 것을 max\_depth를 사용해 예방하는 것

Filled = True, 양성과 음성 그리고 그 정도에 따라 노드들을 색으로 칠함 (파란색일수록 양성)

Feature\_name = : 결정트리에 특성들이 입력된 순서에 따라 특성들의 이름이 담긴 리스트를 입력하면 결정트리 시각화에 그 특성이름값들이 같이 출력됨

출력값들:

특성 <= 숫자 : 특성에게 부여된 조건을 보여주고 이 조건에 따라 자식노드가 yes / no로 갈림

Gini: 지니불순도 = 1 – (음성클래스 비율2 + 양성클래스비율2)

부모 지니불순도 – 자식 지니불순도의 값이 가장 크게 나오는 방향으로 노드 진행

양성 및 음성 클래스가 정확하게 반으로 나눠질 때 지니가 0.5로 가장 큼

한쪽 클래스에 몰려 있으면 지니불순도가 0이 됨

= 순수노드 🡪 제한이(max\_depth) 없으면 leaf노드가 순수노드가 될 때까지 분화함.

Samples: 몇 개의 샘플이 해당 노드로 이동했는 지

Value: 음성클래스와 양성클래스의 개수

관련 기타함수들:

Print(df.feature\_importances\_) : 각 특성에 따른 중요도도 출력 가능

* + - * + **K-최근접 이웃의 다중분류(확률 게산하기) :** 타깃값과 특성이 여러 개 존재할 때 사용가능하며 각 타깃별 분류확률을 구할 수 있음

Kn.classes\_ : 타깃으로부터 추출한 학습값들을 알 수 있음

Kn.predict\_proba(test\_scaled)를 통해 테스트 데이터셋 내의 샘플들을 대상으로 각 타깃별 분류확률 도출

print(proba)를 통해 결과 출력

* + - * 비지도학습
    - 회귀(Regressor)
      * 지도학습
        + **K-최근접 이웃 알고리즘:** 거리에 따라 가장 가까운n개의 샘플들의 평균값으로 예측 값 도출

클래스 불러오기 / 객체 지정/ 훈련 / 스코어

매개변수(하이퍼 파라미터) 지정

N\_neighbors = : 예측 기준이 되는 이웃 개수 지정

모델평가의 종류(score)

R2 = 결정계수 = 1 – (타깃-예측)2의 합 / (타깃 – 평균)2의 합

평균절대값 오차

평균제곱값 오차

**\***잠재적 오류: 데이터의 범위에서 크게 벗어날 경우 정확한 예측 불가

* + - * + **선형회귀(Linear Regression):** 훈련데이터를 대표하는y = a(기울기)x + b의 식을 통해 x의 값이 주어질 때 기울기와 절편을 도출하여 y값을 예측 \*여러 개의 특성이 있을 때에는 계수의 개수도 늘어남

클래스 불러오기 / 객체 지정 / 훈련 / 예측

기울기(계수)와 절편 출력

학습한 직선 그리기

\*잠재적오류: y절편이 음수가 나오는 경우가 있음

* + - * + **다항회귀:** 1차원 선형회귀 식에서 길이를 제곱한 배열을 추가한 것으로 y=ax2 + bx + c의 식으로 표현가능 \*선형회귀에 x를 제곱한 특성을 추가하는 것

클래스 불러오기 / 객체 지정 / 훈련 / 예측

기울기(계수)와 절편 출력

학습한 직선 그리기

* + - * + **다중회귀(multiple regression / multinomial regression):** 특성공학을 사용하여 새로운 특성추가 및 변경을 하여 선형회귀의 기능을 키우는 것이다. \*특성개수+1(타깃)로 차원평면이 그려진다.

다항특성 생성: 클래스 불러오기 / 객체 지정 / 훈련 / 반환기(Transformer)

다중회귀: 다항특성 토대로 선형회귀 훈련 / 점수

\*반환기: 특성이 무엇무엇이 나올 수 있는 지 파악하는 용도

\*fit-transform()을 사용하면 fit과 transform 매써드를 동시에 처리

PolynomialFeatures()의 매개변수

Degree = : 제곱항을 만들어 주는 기능 (default =2)

Include\_bias = :절편관련 특성의 유무를 True/False로 지정

Poly.get\_feature\_names() : 생성된 특성들 출력

\*잠재적 문제: degree를 늘릴수록 과대적합의 위험성이 커짐 🡪 규제 필요

* + - * + **릿지 회귀(L2 규제 활용 규제모델):** 선형회귀 모델에 (가중치)2의 벌칙을 사용

클래스 불러오기 / 객체 지정 / 훈련 / 점수

매개변수:

Alpha = : 알파의 크기에 따라 규제 강도가 강해짐

적절한 규제 강도 찾기:

로그 스케일로 0.01부터 100까지 10배씩 증가하는 alpha값들을 실험 이후 더욱 세밀하게 범위를 자바 탐색

시각화:

Np.log10(alpha\_list)를 x축, 점수를 y축으로 놓는다.

테스트세트의 점수가 가장 크면서 훈련세트의 점수와 가장 차이가 적은 지점을 관찰

* + - * + **라쏘 회귀(L1 규제 활용 규제모델):** 가중치의 절댓값을 벌칙으로 갖는 규제모델 \*릿지회귀와 같은 방식으로 최적의 규제강도 탐색

Np.sum(Lasso.coef ==0)을 하면 가중치를 0으로 만들어 사용하지 않게 되는 특성의 개수를 알 수 있음

* + - * + **로지스틱 회귀**: 선형회귀와 비슷한 특징을 가지되 z값을 추출하는 회귀알고리즘 이후 출력값을 0과 1사이로 변환하는 시그모이드함수/로지스틱함수 적용.

Z = ax무게 + bx길이 + cx대각선 + dx높이 + ex두께 + f

\*0.5보다 크면 양성 클래스 / 0.5보다 작으면 음성 클래스(사실상 분류 알고리즘)

이진분류도 가능(z값 도출 후 시그모이드 함수 적용

타깃관련 샘플들의 인덱스만 따로 추출한 후 표준화한 훈련데이터셋 인덱스로 입력하여 타깃관련 샘플 데이터만 추출

다중분류도 가능

이중분류를 여러 번 시행: 하나의 특성을 양성 나머지를 음성으로 두고 z값 도출모델을 적용하는 과정을 모든 특성에게 각각 반복

도출된 z값을 토대로 소프트맥스함수를 적용하여 확률 형태로 변환

로지스틱 회귀 계수 확인

Lr.decision\_function(훈련데이터) : z함수값 출력하기

Expit(z값들)를 통해 해당 z값들에 시그모이드함수(로지스틱함수) 적용가능

추가원리

OVR/OVA 활용(one verses rest / one verses all) : 클래스 하나를 양성으로 두고 나머지를 음성으로 두어 분류하고 이 과정을 모든 클래스에 따라 반복하여 클래스들을 서로 구분함

**소프트맥스 함수**: z값에 지수함수를 적용하여 z0부터 z6까지 계산한 후 각각을 sum(분모를 다 더 한것)으로 나눠 줌.

용도:

시그모이드 값이 출력이 되고 그 총합이 1이 됨

* + - * + **확률적 경사 하강법 :** 점진적인 학습방법으로 샘플을 하나하나 훈련시키고 모두 훈련시켰을 경우를 1 에포크라고 지칭.
        + **로지스틱 손실 함수(이진분류):** 나쁜 정도를 측정하는 함수로 미분가능한(구간이 없는) 데이터를 대상으로 모델의 정확도를 평가하는 데에 사용됨

사용방법:

SGDCLassifer

매개변수:

Loss = ‘log’ : 로지스틱 손실함수 지정

Max\_iter = : 에포크의 횟수

주의사항:

데이터를 표준점수로 스케일링한 후에 사용

최적의 에포크(매개변수) 찾는 법:

Partial\_fit() : 이전에 학습햇던 것을 유지하면서 추가적으로 학습하는 것 🡪 에포크를 늘려가는 과정으로도 고려가능

For loop 안에 partial\_fit을 사용하여 append기능으로 range(0부터 300까지)리스트를 만들고 이를 라인 그래프로 시각화

Test\_score이 가장 높으면서 train\_score과 점수 차이가 가장 적은 지점을 찾아 최적의 에포크로 선정

* + - * + **크로스엔트로피(crossentropy) 손실함수(다중분류):**

개념:

소포트맥스함수를 거쳐 도출한 확률값들에 로그를 취하고 각각 전체 타깃값들을 곱한다. 이때 해당 레이블의 타깃값만 1로 살고 나머지는 0으로 구성되는 원핫인코딩이 적용되어야함.

용도:

모델의 예측값이 실제 레이블 또는 확률분포와 얼마나 일치하는 지를 측정하여 모델을 훈련 🡪 손실함수 값이 작을수록 모델의 예측이 실제와 일치한다고 판단되며, 이 값을 최소화하는 방향으로 모델을 향상시키도록 학습됨

* + - * + **결정트리(DecisionTreeRegressor)**

주의사항

하나의 스케일 내에서 분류되기 때문에 특성들의 스케일을 표준화할 필요가 없음

관련용어:

Root node: 맨위의 노드

Leaf node: 여러 과정을 거쳐 맨 마지막 단계에 도달했을 떄의 노드

부모노드: 두개의 노드 중 위의 노드

자식노드: 두개의 노드 중 밑의 노드

매개변수들:

Max\_features = : 지정한 숫자에 따라 일부 특성 무작위로 사용 가능 (default = None = 모든 특성 사용)

Plot\_tree() : 트리의 시각화

매개변수들

Max\_depth = : 루트노드 밑에 몇 개의 노드를 출력할 지 지정할 수 있음

가지치기: 순수노드가 될 때까지 분화하여 과대적합이 이루어지는 것을 max\_depth를 사용해 예방하는 것

Filled = True, 양성과 음성 그리고 그 정도에 따라 노드들을 색으로 칠함 (파란색일수록 양성)

Feature\_name = : 결정트리에 특성들이 입력된 순서에 따라 특성들의 이름이 담긴 리스트를 입력하면 결정트리 시각화에 그 특성이름값들이 같이 출력됨

Min\_samples\_split: 샘플 개수의 최솟값을 지정하여 그 이하로는 결정트리를 정지시킴(가지치기의 일환)

출력값들:

특성 <= 숫자 : 특성에게 부여된 조건을 보여주고 이 조건에 따라 자식노드가 yes / no로 갈림

Gini: 지니불순도 = 1 – (음성클래스 비율2 + 양성클래스비율2)

부모 지니불순도 – 자식 지니불순도의 값이 가장 크게 나오는 방향으로 노드 진행

양성 및 음성 클래스가 정확하게 반으로 나눠질 때 지니가 0.5로 가장 큼

한쪽 클래스에 몰려 있으면 지니불순도가 0이 됨

= 순수노드 🡪 제한이(max\_depth) 없으면 leaf노드가 순수노드가 될 때까지 분화함.

Samples: 몇 개의 샘플이 해당 노드로 이동했는 지

Value: 음성클래스와 양성클래스의 개수

관련 기타함수들:

Print(df.feature\_importances\_) : 각 특성에 따른 중요도도 출력 가능

* + - * 비지도학습
        + **K-평균 알고리즘:** 군집에 따라 설정한 개수의 중심을 무작위로 배치하고 이 군집의 중심과 과일간의 거리(유사도)를 반복적으로 확인한다. 확인하는 과정에서 유사한 것들은 특정 중심에 거리가 좁혀지고 아닌 것들은 다른 중심에 모여들어 분류가 이루어진다

관련용어:

군집(클러스터): 데이터를 분류할 때 분류클래스(그룹)

군집화(클러스터링): 군집을 생성하고 이에 따라 분류하는 과정

센트러드: 클러스터의 중심

사용방법:

모델훈련

From sklearn.cluster import Kmeans

Km = Kmeans(n\_clusters=3, random\_state=42)

매개변수:

n\_clusters = 클러스터 중심(센트러드)의 개수(k의 개수)

n\_init = k-중심값을 이동하는 횟수

Km.fit(2차원 배열 데이터)

각 샘플이 어느 클러스터에 지정되었는 지 인덱스값으로 구분하여 출력

Print(km.labels\_)

클러스터 인덱스(labels\_)와 함께 각각에 포함된 샘플의 개수를 출력]

Print(np.unique(km.labels\_, return\_counts = True))

클러스터 결과 확인해보기

레이블 내의 샘플들을 출력해주는 함수 만들기

def draw\_img(arr, ratio = 1):

n = len(arr)

rows = int(np.ceil(n/10)) #나머지가 있는 경우 몫에 1추가

cols = n if rows < 2 else 10 # 샘플 10개 이하면 개수에 따라

fig, axs = plt.subplots(rows, cols,

figsize=(cols\*ratio, rows\*ratio), squeeze = False)

for i in range(rows):

for j in range(cols):

if i\*10 + j < n:

axs[i, j].imshow(arr[i\*10 + j], cmap = 'gray\_r')

axs[i, j].axis('off')

plt.show

draw\_img(img[km.labels\_==0])

매개변수:

arr: 출력할 이미지의 배열

ratio = 1: 가로세로 비율(1:1)

활용:

Draw\_img(img[km.labels\_=인덱스])

클러스터 중심값 시각화하기

Draw\_img(iom.cluster\_centres\_.reshape(-1, height, width), ratio = ) : 도출된 중심값 출력

새로운 이미지 데이터 예측하기

Print(km.transform(새로운 데이터)) : 앞서 km모델을 통해 도출된 중심값들까지의 거리를 출력함

Print(km.predict(새로운 데이터)) : 새로운 이미지를 도출된 클러스터에 따라 분류 후 그 인덱스를 출력

Draw\_img(새로운 데이터) : 데이터를 실제 이미지 형식으로 출력해서 클러스터에 옳게 분류 되었는지 확인

Print(km.n\_iter\_) : 새로운 데이터를 분류하는 데 k 클러스터링 모델이 반복 되었는 지(k 중심값이 몇 번 움직였는 지)

**최적의 k 찾기**: k값은 클러스터의 개수인데 사전에 타깃값이 몇 개 인지 알 수 없으므로 데이터의 특성에 따라 k값을 탐색하는 과정이 필요

방법:

for루프에 range를 담고 intertia라는 비어있는 리스트 변수에 inertia.append(km.intertia\_)를 사용해 k에 따른 intertia값을 축적하고 plt.plot(range(), intertia)를 통해 시각화한다.

Intertia: k평균에 얼마나 가까이 데이터가 위치해 있는가를 보여주는 값 – 조밀조밀하게 모여있으면 inertia가 낮으며 k값이 잘 설정된 것이고 반대로 k값이 크면 중심에서 벗어난 샘플들이 존재한다는 의미

엘보우 그림: 시각화했을 때 k값(x축)이 늘어나도 inertia(y축)이 더 이상 줄어들지 않는 위치를 최적의 k값으로 지정

주의사항:

구형의 클러스터를 만들 때에만 k평균클러스터가 효과적임

이미지가 아닌 다른 자료형태는 표준화와 같은 전처리를 거친 후 사용

* + - **그리드 서치 :** 머신러닝 기법들 내의 최적의 매개변수 값을 찾을 때 배개변수를 바꿔보면서 교차검증을 해보는 과정을 반복하게 해줌 \*매개변수들끼리도 영향을 줄 수 있기 때문에 순차적으로 여러 매개변수를 찾지말고 동시에 상응하는 결과값들을 찾아야 함
      * 방법:
        + 라이브러리에서 클래스 import
        + params라는 변수에 딕셔너리의 형태로 매개변수들과 그 range를 지정
        + GridSearchCV(알고리즘, params, n\_jobs)의 형태로 parmas 기입한 후 훈련

N\_jobs는 병렬처리(동시에 처리)할 모델들의 개수

* + - * + Dt라는 변수에 gs.best\_estimator\_ 값 저장: 그리드서치를 통해 얻은 평균화한 검증점수들 중에서 가장 정확한 결과를 찾고 당시 매개변수들의 설정이 포함된 모델을 dt라는 객체에 저장하는 것이다.
        + Gs.best\_params\_를 통해 선정된 최적의 매개변수값들을 알 수 있음
        + Print.(gs.cv\_results\_[‘mean\_test\_score’])를 통해 각 매개변수의 조합에 따라 도출된 평균화된 검증점수들의 값을 알 수 있음
      * 한계: 매개변수의 조합 개수가 상당히 많으므로 각 경우에 따라 교차검증을 사용하여 여러 모델을 만드는 것은 많은 자원이 필요함
    - **랜덤서치(확률 분포 선택):** 매개변수들을 설정할 때 해당 각 매개변수의 range를 전부 실행하는 것이 아니라 특정 range 안에서 랜덤한 숫자를 뽑도록 설정을 한다. 그리고 나서 만들어지는 모델의 개수를 제한하여(n\_iter 매개변수) 자원의 사용을 줄이면서도 최적의 매개변수 조합을 탐색
      * 개념관련 기초 함수
        + rgen = Randint(0,10) : 0이상 10미만으로 된 정수들을 대상으로 하는 난수 생성기를 만드는 함수
        + Rgen.rvs(10) : rgen의 난수생성기에서 무작위로 10개의 숫자를 뽑아 랜덤배치함
        + Np.unique(rgen.rvs(1000), return\_counts=True) : rgen에서 무작위로 1000개의 숫자를 뽑고 이들 중 고유한 값들과 함께 그 값들에 따른 count를 같이 출력.
        + Ugen = uniform(0,1): 0이상 1미만으로 된 실수들을 대상으로 하는 난수 생성기를 만드는 함수.
      * 매개변수:
        + Params: 클래스 사용 이전에 매개변수들과 그 값들의 range를 설정해 놓은 딕셔너리들의 리스트를 담은 변수
        + N\_iter = : params의 범위 내에서 총 몇번의 샘플링을 할 것인지 지정
        + n\_jobs = : 병렬 처리 개수( -1은 전체 병렬처리)
      * 관련함수들:
        + gs.best\_estimator\_: 가장 최적의 매개변수들을 포함한 최종모델을 새로운 객체에 저장하고 이를 테스트 세트에 적용
        + print(Np.max(gs.\_cv\_results\_[‘mean\_test\_score’])) : 최종적으로 성능평가가 가장 높은 결과값을 출력
    - **앙상블 모델:** 여러가지 모델을 훈련한 후 그들의 예측결과를 평균 내거나 다수를 따라가는 등의 프로세스를 취하는 것 \*결정트리랑 결합성이 가장 좋음
      * **랜덤포레스트(분류/회귀):** 가장 대표적 앙상블 알고리즘으로 결정트리를 여러 개 랜덤으로 만들어 결정트리로 이루어진 숲을 만듦(무작위성 🡪 과대적합 예방) \*기본적으로 100개의 결정트리 생성
        + 랜덤포레스의 두가지 주요 무작위성 기법

**부트스트랩 샘플링** : 훈련세트에서 중복을 허용하여 랜덤하게 샘플링하여 훈련세트와 동일한 개수의 샘플 묶음(= 부트스트랩 샘플) 생성. 각 부트스트랩을 대상으로 결정트리 훈련 후 각 트리의 클래스별 확률들을 전부 더한 후 트리의 개수로 나누어 평균값을 구함 \*회귀모델일 경우엔 예측값을 트리의 개수로 나눔

**랜덤 특성 선택(Random Feature Selection):** 특성집합에서 특성개수만큼 루트를 씌어 그 수 만큼의 특성을 무작위로 선택하고, 이중에서 분할에 사용될 특성을 고름. 다른 노드에서 또 같은 방식으로 랜덤하게 선택 \*최종분할을 하지 않고 최대한 다양한 결과가 나올 수 있도록 무작위성을 높임

* + - * + 랜덤포레스트 훈련:

From sklearn.ensemble import RnadomForestClassifer : 해당 라이브러리에서 클래스를 불러오면 부트랩샘플링을 사용하여 랜덤포레스트로 훈련

매개변수:

Return\_train\_scoreo = True : 교차검증과 함께 사용할 때 교차검증의 매개변수로 해당 코드를 입력하면 훈련평가를 가져올 수 있다.

Oob: 트리가 사용하지 못한 남는 샘플 \*중복을 허용하고 선택하는 횟수가 제한되어 있기 때문에 끝내 선택받지 못하는 샘플들이 존재

Oob\_score = True : oob를 대상으로 훈련을 하여 검증점수를 도출해냄

* + - * **엑스트라 트리 :** 랜덤 특성 선택을 기반으로 100개의 트리를 생성하여 랜덤포레스트를 구성함 \*노드 분할 시 무작위로 선택한 특성의 임계값을 또 무작위로 설정하고 분할
        + 장점: 속도가 빠름
      * **그레이디언트 부스팅:** 분류일 때는 로지스틱손실함수를 기반으로, 회귀일 때는 평균제곱오차를 기준으로 각 트리가 매번 예측값과 실제값의 차이를 측정하는 모델 \*데이터 포인트를 정렬하여 최적 분할을 찾는다 🡪 특성에 대해 가능한 모든 분할 지점을 탐색하여 최적의(손실값이 가장 적은) 분할 선택
        + 매개변수들:

Learning\_rate = : 학습속도 제한

N\_estimators = : 트리의 개수 지정

Max\_depth = : 디폴트로 3을 사용하며 다수의 얕은 트리를 활용하는 것이 그레이디언트 부스팅의 특징임 🡪 이전 트리에서 발생한 예측 오차에 대한 정보를 활용하여 각 트리 학습 = 예측 성능 향상

* + - * + **특성 중요도(Permutation Importance):** 특정 특성의 위치를 바꿨을 때의 모델과 본래의 모델 성능을 비교하여 이 과정을 모든 특성에게 반복하여 각 과정에서의 성능차이를 구한다. 성능차이가 크다는 것은 그만큼 특성이 성능에 크게 일조한다는 의미

용도: 특성 중요도에 따라 히스토그램 기반 그레이디언트 부스팅 보완가능 \*현 강의에서 그 방법까진 알려주지 않음

방법:

Sklearn.inspection 라이브러리에서 Permutation\_importance 클래스 불러오기

Result = Permutation\_importance() : 객체지정

테스트 세트에도 적용해보면 실전에서도 어느 특성이 큰 중요성을 지니는 지 알 수 있음

매개변수들:

N\_repeats = : 특성마다 몇 번씩 성능비교과정을 진행할 것인 지 지정

출력값들:

Print( Result.importances\_mean): 각 특성별 성능차이 출력(순서는 특성의 입력 순서에 따름) 🡪 클수록 중요한 특성이란 것 = 정확도의 손실

* + - **히스토그램 기반 그레이디언트 부스팅:** 훈련 데이터를 256개의 구간으로 잘라서 분할 진행 \*256번째 구간은 누락된 값들이 포함됨
      * 특징들:
        + 분할 지점 탐색과정이 생략되므로 자원 절약
        + 그레이디언트 부스팅과 다르게 특정 특성(손실값이 작아 분할이 잦은)에 중심을 두는 경향이 없어 특성중요도가 존재하지 않음
    - **XGBoost:** 그레이디언트 부스팅 전용 라이브러리
      * 매개변수:
        + Tree\_methods = ‘hist’ : 히스토그램기반 그레이디언트 부스팅을 가능하게 함
    - **LightGBM:** 히스토그램 기반의 그레이디언트 부스팅 전용 라이브러리
  + **흑백 이미지 분류하기:**
    - 기초 개념:
      * 이미지의 배열은 3차원 배열로 .shape를 사용 했을 때 벡터에서의 첫번째 값은 샘플의 개수, 2번째 값/3번째 값은 샘플들 각각의height/width를 의미
        + 2차원 배열 내의 값들은 픽셀 값으로 구성되어 있음
    - 절차:
      * 데이터 전처리
        + 샘플 한 개 확인해보기(배열 + 흑백사진):

print(\_\_\_[0,0,:])

plt.imshow(fruits[0], cmap=’gray’) \*색깔반전은 ‘gray\_r’ 사용하기

plt.show

* + - * + 다루기에 용이한 차원으로 샘플 변경하기

2차원배열의 이미지를 1차원 배열로 펼치기:

.reshape(-1, height\*width)

* + - * 고유특징에 따라 그래프로 시각화하여 비교해 보기
        + 클래스에 따른 샘플 각각의 전체픽셀평균값 분포도 비교하기

Plt.hist(np.mean(샘플카테고리데이터, axis=1), alpha = 0.8)

Axis = 1 : 열에 따라 평균을 냄

Alpha = 0.8 : 투명도

* + - * + 클래스에 따른 전체 샘플의 픽셀 하나하나(위치에 대응되는 픽셀의) 평균값 비교하기

Bar(range(height\*width), np.mean(샘플카테고리데이터, axis=0))

* + - * + 클래에스에 따른 전체샘플의 평균이미지 출력하여 비교하기

다시 2차원 배열의 묶음으로 reshape:

Np.mean(샘플카테고리데이터, axis=0).reshape(height,width)

각각의 이미지카테고리별 데이터를 fig, axs를 통해 동시출력

* + - * 위의 클래스별 평균이미지 데이터를 통해 새로운 이미지들을 예측분류하는 법
        + 새로운 이미지 데이터와 카테고리별 평균이미지 데이터의 차이(절댓값)를 구한 후 차이가 가장 작은 카테고리로 분류

각 샘플마다 평균이미지와의 차이(절댓값) 구하기

Abs.diff = np.abs(새로운이미지데이터 – 클래스별평균이미지데이터)

각 픽셀마다의 차이값을 height랑 width에 따라 평균화하기

Abs\_mean = np.mean(abs\_diff, axis(1,2)

\*axis(1,2)는 각각 height, width

차이가 작은 것들 순으로 정렬하여 인덱스로 저장하기

인덱스변수 = np.argsort(abs\_mean):[:abs\_mean이 크게 달라지는 지점]

Fig, axs = plot.subplots()로 시각화

* + **주성분 분석(차원축소 알고리즘을 이용한)**
    - 필수개념:
      * 주성분(PCA): 훈련데이터가 산점도로 놓여져 있을 때 어느 한 방향으로 편향되어 퍼져 있는 경우 가장 많이 퍼져 있는 쪽으로 진행하며 원점을 지나는 벡터

= 데이터를 표현하는 데에 주된 성분 \*PCA 알고리즘 = 찾는 과정

* + - * 주성분 차원도: 원본공간차원\*개수가 아닌 주성분 각각의 차원
      * Ex) 직선의 방정식 같은 경우의 산점도엔 특성2개(x,y) 중 1개를 찾아 축소시키는 것
      * 주성분을 여러 개 찾는 경우: 원점에서 시작하며 주성분 벡터에 수직인 방향으로 두번째 주성분을 찾음 = 첫번쨰 주성분과 독립적으로 퍼져 있는 성분
    - 사용방법:
      * 불러오기:
        + From sklearn.decomposition import PCA
      * 객체 지정 및 축소시킬 주성분의 개수 지정하기:
        + pca = PCA(n\_componenets=50)

n\_components를 0과 1사이로 지정하면하면 설명된 분산이 해당 수치에 도달할 때까지의 주성분 개수 지정 ex) 0.5 = 설명된 분산50%

* + - * 훈련시키기 \*이미지의 경우 2차원배열로 변환 후 사용
      * 주성분 축소 결과 확인하기
        + shape확인

Print(pca.components\_.shape)

* + - * + 이미지로 출력하여 시각적으로 확인

Draw\_img(pca.components\_.reshape(-1,height,width)

* + - * 원본데이터를 주성분만을 사용하여 재구성
        + Img\_pca = pca.transform(img)

Print(img\_pca.shape)

* + - * 주성분만을 사용하여 재구성한 데이터를 기반으로 다시 원본 데이터의 성분개수만큼 늘려 이미지 결과 출력
        + Img\_inverse = pca.inverse\_transform(img\_pca)

Print(fruits\_inverse.shape)

Fruits\_reconstruct = fruits\_inverse.reshape(-1, height, width)

\* 축소시키면서 손실되고 남은 데이터를 기반으로 복원하기 때문에 실제 원본 이미지 데이터의 특성을 어느정도 유지하되 차이는 있음

* + - **설명된 분산**: 원본 데이터와 주성분을 토대로 원본데이터의 크기로 재구성한 데이터 간의 분산(차이)를 비교하는 것
      * 사용방법:
        + 설명된 분산값 출력:

Print(np.sum(pca.explained\_variance\_ratio\_)) : 각 배열이 얼마만큼 분산을 잘 설명하는 지 알려주며 이를 총산하여 재구성된 데이터가 본래 데이터의 몇 퍼센테지 만큼 보존하고 있는 지를 출력

* + - * + 주성분의 개수가 늘어남에 따라 설명되는 분산의 양을 시각화

Plt.plot(pca.expalined\_variance\_ratio\_)

* + - 용도:
      * 주성분 축소는 일종의 변환기로 주성분 변환 후 분류기나 회귀분석과 같은 다른 모델에 연결하여 사용 가능
        + Pca.transform()을 사용해 주성분으로만 이루어진 데이터로 축소 후 분류기(ex) 군집)나 회귀모델을 포함한 교차검증 적용
        + 군집과 연결하여 KMeans() 모델 사용 후 추가과정

주성분 분석을 통해 특성을 축소하면 이를 시각화하였을 때 특성에 따른 데이터들의 분포 양상이 조금 더 명확하게 보여진다.

For label in range(0,3):

Data = img\_pca[km.labels\_ == label]

Plt.scatter(data[:,0], data[:,1]) #모든 샘플의 0=height, 1=width

Plt.legend([군집이름 리스트로 작성])

Plt.show()

* + - * 주성분으로 축소를 하면 자원을 크게 아낄 수 있음 = fit\_time 축소
  + 과대적합 / 과소적합 확인
    - 과소적합: 훈련세트 점수 < 테스트세트 점수
    - 과대적합: 훈련세트 점수 > 테스트세트 점수