텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명**12강 교차 검증과 그리드 서치**

사실 3개의 세트로 나누어서 모델을 훈련하는 것이 가장 바람직함: 훈련세트, 검증세트, 테스트세트

테스트세트: 서비스에 투입 시 성능예측 \*서비스에 투입 시에는 타깃이 없으므로 예측해야 함. 🡺 실전에 투입 시 어느정도 성능을 할 지 감을 얻는 것이 테스트 세트 \*적용연습을 할 실전용데이터라고 생각하면 됨

* 그러면 과대/과소적합을 테스트세트와 비교하는 건 시뮬레이션이 아닌 모델구성의 과정이므로 적합하지 않다. 🡪 검증세트를 통해 검증과정을 거친 후 최종적으로 테스트 세트에 적용 \*검증세트에서 매개변수 튜닝(하이퍼 파라미터 설정)을 진행

훈련세트: 모델 구축 / 검증세트: 모델 매개변수 튜닝(과적합이슈완화) / 테스트 세트: 실전 시뮬레이션 \*테스트 세트를 사용하기 전에 훈련세트와 검증세트를 병합하여 최종모델을 훈련한 후에 테스트세트에 적용. \*테스트세트의 성능에 따라 다시 모델 훈련하고 테스트 세트 점수확인하는 과정을 가지면 또 테스트 점수에 과대적합되는 현상이 일어나므로 유념해 두어야 함

* 교차검증 \*딥러닝에서 사용됨

= 훈련세트의텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명 양이 크지 않을 때 더욱 효율적으로 모델을 훈련시키는 방법

훈련세트를 세개의 폴드로 나눈 후 그 중 하나를 검증세트로 지정한다. 이러한 형식으로 검증세트를 3개 중에 돌아가면서 지정하여 3회 모델 훈련을 실시하고 검증된 점수를 평균화한다.

Ex) 모델훈련1: 훈련세트12 검증세트3

모델훈련2: 훈련세트13, 검증세트 2

모델훈련3: 훈련세트23, 검증세트 1

* *Coross\_validate()* : 훈련모델, 훈련세트(훈련데이터, 훈련타겟)을 넣으면 자동으로 5폴드로 나누어서 각각의 폴드를 돌아가며 검증세트로 지정하여 총 5차례 훈련 및 스코어 점수를 낸다.
  + fit\_time: 훈련하는 데 걸리는 시간
  + score\_time: 점수도출하는 데 나오는 시간
  + test\_score:5개의 검증점수

np.mean(scores[‘test\_score’]))를 통해 검증점수 평균화

* 딥러닝에선 검증세트와 훈련세트만을 사용 / 머신러닝에선 검증세트와 훈련세트와 더불어 교차검증을 사용 🡪 딥러닝에선 데이터가 풍부하기 때문에

\*tensorflow와 같은 딥러닝 library에는 교차검증보다 데이터를 읽어들이는 도구가 많음(양이 많음

\*머신러닝 library에는 교차검증이나 매개변수 탐색 도구가 더 많음

* 텍스트, 폰트, 스크린샷, 라인이(가) 표시된 사진

  자동 생성된 설명분할기(사이킷런에선 spliter)를 사용한 교차 검증

Cv 매개변수는 기본적으로 5개의 폴드를 사용하는데 cv=10으로 쓰면 폴드 개수 지정 가능

Cv는 기본적으로:

Kfold라는 split모델 사용(회귀일 경우)

StratifiedKfold(분류모델일 경우)\*폴드안의 클래스들이 골고루 나눠지도록

분류모델이나 회귀모델로 특정 지어지는 모델은 자동으로 사이킷런이 분별하여 적합한 매개변수 사용

Splitter 객체를 새로 만들어서 StratifiedKfold에 n\_splits(폴드 개수), shuffle(무작위로 섞기)를 넣어 cv매개변수를 만든 splitter로 지정 가능

\*스플리터는 교차검증을 해주는 기능이라고 생각하면됨

방법에는 3가지:

1. Cross\_validate() 매개변수로 cv = StratifiedKFold() 넣기 \*분류모델전용
2. Cross\_validate() 매개변수로 cv = KFold() 넣기 \*회귀모델전용
3. Splitter 객체를 생성하여 매개변수 cv = splitter를 사용해 직접 설정한 세부적인 조건으로 교차검증하기

* 텍스트, 스크린샷, 폰트이(가) 표시된 사진

  자동 생성된 설명그리드서치: 매개변수를 바꿔보고 교차검증을 해보는 과정을 반복해야 될 때 사용

\*매개변수들이 서로에게 영향을 줄 수 있기 때문에 순차적으로 최적의 값을 찾는 게 아니라 동시에 상응하는 결과값들을 찾아야 함

\*정보이득: 자식과 부모 노드간의 불신도 차이: 정보이득이 클수록 정보분할을 잘한 것 🡪 정보이득의 최솟값을 지정 \*최솟값 이하는 노드 분할금지

\*정보이득이 클수록 좋은 이유: 부모노드의 불확실성 보다 자식노드의 불확실성이 크게 감소했다는 것을 의미(감소밖에 없는 상황에서 차이가 크다는 것) 🡺 불확실성이 낮아지고 예측의 정확도가 올라감

* 가장 적합한 정보이득 최솟값을 모르므로 params라는 딕셔너리를 만들어 매개변수이름을 스트링으로 지정하고 그 값을 넣어보고 싶은 값들의 리스트로 지정.

🡪 매개변수에 해당 매개변수이름을 소환가능

총 25개의 모델이 만들어짐(5개의 매개변수 \* 디폴트 5개의 폴드 교차검증)

N\_jobs 은 qudfufgsfus을 가능하게 함 🡪 -1로 지정하면 가능한 모든 코어를 동시에 수행 🡪 25개의 모델을 순차적이 아닌 동시에 돌린다는 의미

Dt = gs.best\_estmiator\_는 5가지의 파라미터를 통해 돌린 평균화한 교차검증 결과값들 중 가장 정확한 결과를 찾아 최적의 파라미터를 구하고 이 파라미터를 토대로 훈련세트+검증세트에 훈련시킨다.

Gs.best\_params\_ : 선정된 최적의 매개변수 출력

Gs.cv\_results\_[‘mean\_test\_score’])를 하면 위에서 출력한 각 파라미터에 따라 평균화된 5가지의 테스트 스코어가 출력됨

\*매개변수를 더 탐색해보고 싶으면 params를 range나 넘파이 배열로 만들어도 괜찮다.

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

박스 안의 params: 9 \* 15 \*10 = 모델훈련 🡪 촘촘하게 찾아볼려면 꽤 많은 자원이 필요함

* 텍스트, 폰트, 스크린샷이(가) 표시된 사진

  자동 생성된 설명랜덤서치(확률 분포 서택)

Uniform과 randint는 균등분포(고르게 샘플링이 되어 있는 상태)에서 샘플링하는 클래스

Uniform은 실수

Randint는 정수

Rgen을 객체로 지정

Rgen.rvs(10)을 하면 rgen의 샘플 중에서 10개의 샘플을 뽑아 랜덤배치함

두번째는 rgen의 샘플 중에서 1000개의 값을 뽑아(중복 가능) 이들 중 고유한 값들 각각의 카운트 개수를 뽑는다.

Uniform()은 실수값 샘플링 \*랜덤하게 샘플링하는 것으로 테스트 🡪 일정간격으로 테스트를 하는 것이 아닌 더 촘촘하게 테스트를 하기 위해서 -->

* 텍스트, 스크린샷, 폰트, 문서이(가) 표시된 사진

  자동 생성된 설명랜덤서치

Uniform(0.0001, 001)은 그 사이의 값들의 균등분포를 의미

N\_iter = 100이 모델을 만들 개수고 params의 범위 내에서 샘플링을 100번하여 모델을 만든느 것

이러한 모델을 gs 객체에 넣은 후 훈련하여 베스트 파라미터와 제일 정확한 검증세트의 점수를 찾는다.

Np.max(gs.cv\_results\_[‘mean\_test\_score’]))

그리고 최종적으로 test\_input에 적용

그리드 서치를 하여 구간을 설정한 후 그 구간 내를 세부적으로 살펴볼 때 랜덤서치를 사용하는 느끼미 \* 구간 내에서도 세부적인 값이 셀수 없이 많기에 랜덤으로 골라서 개수를 한정해서 보는 것임

**13강 트리의 앙상블**

* 정형데이터: 행과열이 잘 구분되는 데이터(csv, db, 엑셀 등에 담을 수 있는 레코드) – 머신러닝에서 자주 사용
* 비정형 데이터: 텍스트 ,오디오, 이미지, 영상 등 – 딥러닝을 자주 사용함
* 비정형 데이터에서는 특성공학을 얻기가 까다로움 🡪 특성추출을 스스로 잘하는 ‘표현학습’ 알고리즘을 사용하기에 일반적으로 비정형 데이터에 적합하다고 함

앙상블 모델: 여러가지도표, 스케치, 텍스트, 그림이(가) 표시된 사진

자동 생성된 설명 모델을 훈련한 후 그들의 예측결과를 평균 내거나 다수를 따라가는 등의 프로세스를 취하는 것 – 결정트리랑 가장 잘 맞물림

랜덤 포레스트: 가장 대표적인 앙상블 알고리즘 = 결정트리를 여러 개 랜덤으로 만들어서 결정트리로 이루어진 숲을 만드는 것.(무작위성을 주어 트리의 성능을 억제 – 과대적합 예방) \*classifier / regressor 둘다 존재

* 랜덤포레스트에서 사용되는 두가지 주요 무작위성 기법
* 텍스트, 폰트, 도표, 친필이(가) 표시된 사진

  자동 생성된 설명부트스트랩 샘플링 사용: 훈련세트에서 랜덤하게 샘플링 \*동일한 샘플이 중복해서 선택받을 수 있음 = 중복허용샘플링
* 부트스트랩 샘플의 데이터 개수는 훈련세트와 동일하게 유지
* 두 부트스트랩을 각각 훈련하면 결정트리 훈련에서 나오는 클래스별 확률 존재. 그 확률들을 전부 더한 후 트리 개수로 나눔 \*회귀모델일 경우 확률이 아닌 예측값이 나올 것임 🡪 여러 개의 예측 / 트리 개수
* 랜덤 특성 선택(Random Feature Selection): 특성집합에서 특성개수만큼 루트를 쓰여 그 수 만큼의 특성을 무작위로 선택하고, 이중에서 분할에 사용될 특성을 고름. 다른 노드에서 또 같은 방식으로 랜덤하게 선택 🡪 최종의 분할을 하지않고 최대한 다양한 결과가 나올 수 있게 무작위성을 높임
* 기본적으로 랜덤포레스트는 100개의 트리를 만듦

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명return\_train\_score = True를 통해 훈련평가를 가져올 수 있음

특성을 랜덤하게 고르다보니 결정트리에서 특성의 중요도가 낮은 특성도 높아질 수 밖에 없다.

\*부트스트랩샘플링 사용

트리가 사용하지 못한 남는샘플: OOB \*중복을 허용하기에 랜덤하게 남는 샘플들이 있을 거임

* 남는 샘플들을 사용해 검증세트의 역할을 하게 만들 수 있음 \*하지만 디폴트는 OOB로 검증을 하지 않고 매개변수로 oob\_score을 사용해야 검증이 진행된다.
* Oob\_score와 테스트 스코어를 비교해서 과대/과소적합에 대해 다시 한번 비교해볼 수 있다.
* 테스트스코어는 검증세트의 점수임(검증세트는 훈련과정에서 사용되지 않기 때문에 테스트세트랑 마찬가지의 역할을 할 수 있음
* Train\_score는 훈련한 데이터를 대상으로 테스트했을 때의 저수
* 두번째는 oob를 검증세트로 활용하여 점수를 도출한 것임 🡪 위의 test\_score와 차이가 크지 않다는 것을 통해 test 시 어느정도 일관된 결과가 나온 다는 것을 알 수 있음
* 엑스트라 트리

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명100개의 트리 사용, 특성의 개수의 제곱근 사용하여 샘플링. \*부트스트랩샘플링 사용하지 않고 전체 샘플 사용 🡪 노드분할 시 무작위로 선택한 특성의 임계값을 또 무작위로 설정하고 분할한다.. \*속도가 빠름

분류텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명: 로지스틱손실함수

회귀: 평균제곱오차 사용

* 위의 두 값들의 실수값이 낮아지도록 트리를 추가 \*각 트리는 예측값과 실제값의 차이를 매번 측정

결정트리 회귀모델 사용

\*learning\_Rate 매개변수를 이용하여 학습속도 제한. Max\_depth를 3으로 제한하여 얕은 트리를 사용. 얕은 트리 여러 개 들이 쌓여가는 그림. 이전 트리에서 발생한 예측 오차에 대한 정보를 활용하여 각 트리가 학습됨 🡪 이전 트리에서 놓친 패턴을 찾고 예측 성능 향상시킴

N\_estimatros-500은 트리의 개수를 더 높이는 것

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명255개의 구간으로 나뉘어 누락된 값은 1구간으로

훈련데이터를 256개의 구간으로 잘라서 특성변환 🡪 특성의 값의 범위가 제한되어 있기 때문에 속도가 빠름. 255 + 1으로 나누어 누락된 값은 마지막 1에 들어감 🡪 누락된 값을 처리할 필요가 없음

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명\*

행에 샘플이 있고 열에 특성이 있을 때 첫번째 특성의 열들에 대한 위치를 섞은 상태에서 성능을 평가한다. 그리고 나서 두번째 특성도 마찬가지로 과정반복.

이후 원래 데이터의 성능을 구하여 성능차이를 비교해본다. 그래서 각 특성의 위치를 섞었을 때와 일반적인 상황에서 성능을 확인했을 때 차이가 큰 경우는 그 특성이 성능에 크게 일조한다는 것을 의미 🡪 특성들 각각의 중요성을 도출해내는 알고리즘

N\_repeats = 10 : 특성마다 10번씩 섞는 것을 의미

result.importances\_mean : 일반적인 상황에서의 성능과 각 특성의 위치를 섞었을 때의 성능차이 🡪 클수록 중요한 특성이란 것 =정확도의 손실

훈련세트에 한 후 테스트 세트에 적용을 하면 어떤 특성이 실전에 투입되었을 때 더 민감한지 발견할 수 있음

Ex) 0.049는 실전에서 특히 큰 중요성이 없다는 것을 알 수 있음

최종 테스트셋에 적용했을떄(실전예행연습에서의) 점수 hgb.score()

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명XGBooST: gradient boosting 전용 library임 \*많이 사용됨

LightGBM: 🡪 histogram 기반의 boosting을 처음으로 지원해서 인기가 높아짐

XGBClassifer()에 tree\_method=’hist’로 하면 히스토그램 기반, cross\_validate적용 가능

LGBMClassifier도 cross\_validate 적용가능

앙상블 작업을 하고 싶으면 사이킷런의 그레이디언트 부스팅 알고리즘 사용가능(histogram기반을 전문적으로 하고 싶으면 위의 두 library 사용)

텍스트, 폰트, 스크린샷, 친필이(가) 표시된 사진

자동 생성된 설명

\*에이다부스트라는 알고리즘도 앙상블의 대표적인 종류로 존재함

**14강 흑백 이미지 분류 방법과 비지도 학습, 군집 알고리즘 이해하기**

* 텍스트, 스크린샷, 폰트, 도표이(가) 표시된 사진

  자동 생성된 설명비지도학습 활용예시: 과일 사진을 분류하는 모델을 만들어야 되는데 타데이터(어떤 과일사진을 습득할 것인가)에 대한 정보가 없음 = 타깃X, 특성데이터만 O

넘파이 배열로 사진들이 저장된 파일임.

!는 shell명령을 작성할 때 사용. Wget을 하면 shell프롬프트에서 파일을 불러오라는 명령을 내리는 것으로 이해

항상 입수한 데이터의 shape를 먼저 확인해보자.

#300은 배열에 들어 있는 샘플의 개수

#100 은 샘플들 각각의 height랑 width를 의미

첫번째텍스트, 스크린샷, 도표, 원이(가) 표시된 사진

자동 생성된 설명 값을 샘플이라고 보고 두번째랑 세번쨰 값을 2차원배열의 픽셀값들이라고 보면된다.

첫번째 행에 있는 100개의 원소를 출력한 것

= 샘플 1의 특성

Plt.imshow는 넘파이배열로 저장된 값을 2차원이미지로 출력해주는 함수, 츠메는 사용할 컬러의 종류를 지정해줌 – 흑백이미지이므로 흑백으로 출력한 것 \*디폴트는 단록색(miribeath(?))임

출력한 배열에서 숫자가 갑자기 커지는 부분은 사과꼭지임(하얗게 되어 있는 부분은 높고(to 255) 검은 부분은 낮음(to 0).

Gray\_r을 사용하면 색상이 반전해서 나오므로 인식하기에 편해짐

* 과일 데이터 분류 – 샘플마다 2차원 배열(이미지)를 1차원 배열로 펄쳐서 살펴볼 것임(분류의 편의를 위해)

샘플차원은텍스트, 폰트, 스크린샷, 디자인이(가) 표시된 사진

자동 생성된 설명 -1로 그대로 두고 두번째 차원과 세번쨰 차원 100\*100을 하나로 묶어서 1000으로 만든 것.

* 열에는 샘플이 있고 행에는 픽셀이 있는 것
* 각 과일들 샘플의 평균을 내어 이 평균값을 과일 간에 비교 🡪 과일을 구분하는 특성 찾기

축을텍스트, 스크린샷, 도표, 폰트이(가) 표시된 사진

자동 생성된 설명 지정해서 평균을 낼 때 axis=1은 가각의 열을 따라서 평균을 내므로 100개의 평균값이 나오는 것임 \*axis=0으로 하면 행에 따라서 평균을 내므로 1000개의 평균값이 나올 것임

히스토그램:

X축(구간): 픽셀의 평균값

y값은 빈도

바나나가 낮은 픽셀평균값을 지니고 있다는 것을 알 수 있음. 파인애플과 사과는 픽셀평균이 비슷하다는 것을 알 수 있음.

* 샘플별로 평균값을 내는 게 아닌 픽셀별로 평균값 내기(axis=0 사용)

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명10000개의 픽셀에 대한 평균값 계산이 됨

x축에는 픽셀값, y축에는 평균값

plt.subplots은 부분그래프를 그려서 여러 개의 그래프를 한 이미지에 그릴 수 있게 함

1,3은 열과 행을 지정한 것으로 이것을 통해 그래프의 개수 지정가능. 한 개의 행에 3개의 그래프를 놓는다는 의미(배치)

Axs[0]은 각 그래프의 위치에 따른 인덱스

Range(10000)를 통해 x축의 좌표 정의하고, np.mean(apple, axis=0)을 통해 각 과일 종류의 픽셀 평균값으로 지정. 🡪 사과는 아래쪽이 빈도가 짙게 나온 것을 알 수 있으며 바나나는 높게, 파인애플은 전반적으로 고르다는 것을 알 수 있음. 🡪 과일종류에 따라 나타나는 픽셀의 형태와 분포에 차이가 있음.

* 위의 결과를 100\*100의 원본 이미지처럼 형태를 바꿔서 그려보는 것도 가능

다시텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명 reshape를 통해 100\*100으로 그리기

Plt.subplots로 3개의 히스토그램 그리고 gray\_r을 통해 색깔반전

사과: 흐릿하지만 사과의 형태를 잘 보임

파인애플: 파인애플도 형태가 잘 보임

바나나: 로테이션이 많이 바뀌기에 다양한 형태를 보임

분류방법: 각 히스토그램에서 보이는 평균과 새로 입수되는 데이터 간의 차이를 비교해보고 차이값이 가장 작은 카테고리로 새로운 그림을 분류하는 것이다.

* 사과 평균이미지를 통해 300개의 사과 데이터 중에 사과만 잘 골라지는 지 확인

텍스트, 폰트, 친필, 스크린샷이(가) 표시된 사진

자동 생성된 설명Abs\_diff 는 300개의 과일에서 사과의 평균을 뺀 것의 절댓값을 찾는 것이고 각 샘플별로 뺀 값이 가장 작은 값을 구하는 것. Axis 1과 axis 2에 대해서 모두 평균을 내는 것(2번째 차원과 3번쨰 차원에 대해서) \*0번쨰 차원은 샘플을 구분해주는 것이므로 평균 내면 안됨

* Abs\_mean이 가장 작은 것들이 애플의 평균이미지와 가장 가까운 것일 것이고 이들을 사과라고 분류할 수 있음

\*abs\_mean이 가장 작은 값들의 위치를 찾기 위해 np.argsort을 쓰는 것임.(작은 것부터 큰것까지 줄 세우며 그 중 시작부터 100개를 선택할 수 있음)

Subplot을 만들어서 추출한 100개의 샘플들을 그려낼 것임

For 루프를 두번 써서 10개의 열을 그리며 행에 따라 채워나가는 식으로 fruits[apple\_index[i\*10 + j]] \*0부터 시작해서 j가 9까지 채우면 i가 10부터 시작해서 자릿수를 늘려가는 형식

* 이미지에 있는 픽셀의 평균값과 가장 가까운 이미지를 찾으면 분류가능 = 군집(clustering)

하지만 이것은 타깃을 이미 인지하고 있다는 의미이기에 완벽한 비지도학습을 수행한 것은 아님