TITLE : 2_사이킷런으로 시작하는 머신러닝 DATE : 2022.08.27

1 사이킷런 소개와 특징

- 파이썬 머신러닝 라이브러리
- 특징
  - i) 쉬운 API
  - ii) 머신러닝을 위한 다양한 알고리즘과 개발을 위한 편리한 프레임워크/API 제공
  - iii) 활용도↑

2 머신러닝_분류

① 분류(classification)
- 지도학습 방법
  ↳ 학습을 위한 다양한 feature와 분류 결정값인 label("정답") 데이터로 모델을 학습한 뒤, 별도의 test data 세트에서 미지의 레이블 예측

② Sklearn 내의 모듈
- i) sklearn.datasets : 사이킷런에서 자체적으로 제공하는 데이터 세트를 생성
- ii) sklearn.tree : 트리 기반 ML 알고리즘을 구현한 클래스의 모임
- iii) sklearn.model_selection : 데이터 분리(학습/검증/예측 데이터) or 최적의 하이퍼파라미터로 평가하게 위한 모듈

③ train_test_split( ) : dataset을 학습 데이터와 테스트 데이터로 분리
↳ parameters
- feature data
- label data
- test_size : 전체 데이터 세트 중 test data의 비율
- random_state : 랜덤 seed값 지정

↳ 반환값 > 순서대로 학습용 feature 데이터, 테스트용 feature 데이터, 학습용 label 데이터 , 테스트용 label 데이터

④ model.fit(X_train, y_train) : 모델 학습시키기
분류 모델 객체

⑤ model.predict(X_test) : 테스트용 feature data에 대한 label값 예측

⑥ accuracy_score( ) ┌ 정확도(예측 결과가 실제 label과 얼마나 일치하는가) 계산
└ accuracy_score(실제, 예측 결과)

3 사이킷런의 기반 framework

① Estimator 이해 & fit( ), predict( ) 메서드
- i) Estimator
  ↳ Classifier(분류 알고리즘을 구현한 클래스) + Regressor(회귀 알고리즘을 구현한 클래스)
  ↳ 학습: fit( ), 예측: predict( )

② 주요 모듈
↳ p.94~95 표

③ 내장된 데이터셋
- i) 예제 데이터셋 API : datasets.load_데이터셋명( )
- ii) 표본 데이터 생성기
  - a. datasets.make_classifications( ) : 분류 데이터셋 생성
  - b. datasets.make_blobs( ) : 클러스터링을 위한 데이터셋 생성
- iii) key값
  - data : feature 데이터세트
  - target : 분류⇒label, 회귀⇒숫자 결과값 데이터
  - target_names : 개별 label의 이름
  - feature_names : feature의 이름
  - DESCR : 데이터 세트에 대한 설명 ⊕ 각 feature에 대한 요약

♥ TITLE : 2_사이킷런으로 시작하는 머신러닝 ♥ DATE : 2022.08.

4 Model Selection 모듈

- 학습 데이터와 테스트 데이터 세트 분리
- 교차 검증 분할·평가
- estimator의 하이퍼 파라미터를 튜닝

① train_test_split( )

- 학습/테스트 데이터 분리

★ 예측을 수행하는 데이터 세트는 학습을 수행한 학습용 데이터 세트가 아니라 전용 test 데이터셋이여야 함.

- **parameters**
  - i) feature datasets ┐ 필수
  - ii) label datasets ┘
  - *선택사항
  - iii) test_size : 전체 데이터 중 test data의 크기 지정 default: 0.25
  - iv) train_size : 〃 train data 〃
  - v) shuffle : 데이터 분리 전 미리 섞을지 결정 default: True
  - vi) random_state : 시드값 지정 → 데이터 변경 방지
- **반환값**
  - ↳ (X_train, X_test, y_train, y_test)
    - feature
    - label

② 교차 검증

- 오델의 과적합(overfitting)을 방지
  - ex 특정한 학습 데이터에만 과도하게 최적화되는 것
- 데이터 편중을 막기 위해 별도의 여러 세트로 구성된 학습 데이터 세트와 검증 데이터 세트에서 학습·평가 수행 → 파라미터 튜닝 등의 모델 최적화
  → 테스트 데이터로 최종 평가

i) k-fold 교차 검증

↳ k개의 데이터 폴드 세트를 만들어 k번만큼 각 fold 세트에 대해 학습-검증을 반복적으로 수행하는 방법

↳ 순서

1. 데이터 세트를 k등분
2. k번 반복하여 1개의 등분을 검증 데이터로, 나머지를 학습 데이터로 이용
   - ex 모든 등분이 한번씩은 검증 데이터로 사용되도록

↳ **parameters & 메서드**

1) kfold.split( ) ┐ 데이터를 k개의 fold datasets로 분할
   - feature data
   - ┘ 리턴값 > 학습용/검증용 데이터로 분할할 수 있는 인덱스

ii) Stratified k-fold

↳ 불균형한 분포도를 가진 레이블(결정 클래스) 데이터 집합을 위한 k-fold 방식

↳ 원본 데이터의 레이블 분포를 먼저 고려한 뒤, 이 분포와 동일하게 데이터 분배

↳ kfold.split(feature data, label data) ← 분포 고려

iii) cross_val_score( )

↳ 교차 검증을 좀 더 편리하게 수행할 수 있는 API

↳ **parameters**

1) estimator : 사이킷런의 분류 알고리즘 클래스(분류/회귀) → Stratified k-fold 적용 됨!
2) X : feature data
3) y : label data default > None
4) scoring : 예측 성능 평가 지표 〃 ex. accuracy : 정확도
5) cv : 교차 검증 fold 수 〃

↳ 반환값 > scoring 파라미터로 지정된 성능 지표 측정값을 배열 형태로

↳ 내부적으로 Stratified k-fold를 이용

↳ 단 하나의 평가 지표만 지정할 수 o <cf> cross_validate( )

- ↳ 여러 개의 평가 지표로 반환
- ↳ 학습 데이터에 대한 성능 평가·수행시간

set a record

♥ TITLE : 2_사이킷런으로 시작하는 머신러닝 ♥ DATE : 2022.08.28

④ GridSearchCV

- 하이퍼 파라미터를 순차적으로 입력하면서 편리하게 최적의 파라미터를 도출
- 교차 검증 기반
- 수행시간이 오래 걸린다는 단점이 O
- 전체 테스트 수행 횟수: (CV의 수)X(측정하고자 하는 파라미터의 수)
- parameters>
  - i) estimator: classifier, regression, pipeline
  - ii) param_grid: key+list 값을 가지는 딕셔너리 입력
    - 튜닝을 할 파라미터의 이름·값 지정
  - iii) scoring: 예측 성능을 평가할 방법 지정
  - iv) cv: 분할 fold 수 지정
  - v) refit: True(디폴트 값)이면 가장 최적의 하이퍼 파라미터를 찾은 뒤 입력된 estimator 객체를 해당 하이퍼 파라미터로 재학습
- cv_results_: GridSearchCV의 결과 세트
  - 딕셔너리 형태로 key값과 리스트 형태의 value값을 가짐

  * 주요 key
    - i) params: 각 수행 시 적용된 개별 하이퍼 파라미터값
    - ii) rank_test_score: 하이퍼 파라미터별 성능의 순위, 1이 가장 good
    - iii) mean_test_score: 개별 하이퍼 파라미터별로 CV의 folding test set에 대해 총 수행한 평가의 평균값

- best_params_: fit() 결과 최고 성능을 낸 하이퍼 파라미터 조합이 반환됨.
- best_score_: 〃 의 평가 결과

5 데이터 전처리

- 결손값(NaN, Null) 처리
  - 평균값 등의 고정값으로 대체
  - 해당 feature 제거
- 문자열 값 처리: 인코딩을 통해 숫자형으로 변경

① 데이터 인코딩

i) 라벨 인코딩

- 카테고리 피처 → 코드형 숫자
- 예시> TV: 1, 냉장고: 2, 세탁기: 3 ...
- LabelEncoder 클래스로 구현 ⇒ fit( ), transform( )
- encoder.classes_: 속성값 확인
- encoder.inverse_transform( ): 디코딩(숫자 → 문자열)
- 단점> 몇몇 ML 알고리즘(ex. 선형 회귀)에서 숫자의 크고 작음에 대해 잘못된 예측 결과가 발생될 수 O

ii) One-Hot Encoding

- feature의 유형에 따라 새로운 피처를 추가해 고유 값에 해당하는 칼럼에만 1을 표시하고 나머지 칼럼에는 0 표시
  ⇒ 행 형태로 되어 있는 feature의 고유값 → 열 형태(차원 변환)
- OneHotEncoder 클래스로 구현 *주의)
  1. 적용 전 모든 문자열 값이 숫자형으로 변환되어
  2. 입력값으로 2차원 데이터가 필요
- Pd.get_dummies( ): 문자열 값을 바로 인코딩 할 수 O

♥ TITLE : 2_사이킷런으로 시작하는 머신러닝 ♥ DATE : 2022.08.28

② 피처 스케일링 & 정규화

- 피처 스케일링: 서로 다른 변수의 값 범위를 일정한 수준으로 맞추는 작업

i) 표준화
- 데이터의 feature 각각이 평균이 0이고 분산이 1인 가우시안 정규분포를 가진 값으로 변환
- $x_{i\_new} = \frac{x_i - E(x)}{Sd(x)}$

ii) 정규화
- 서로 다른 feature의 크기를 통일하기 위해 크기를 변환 ⇒ 단위 통일
- 사이킷런에서의 의미 > 개별 벡터의 크기를 맞추기 위해 변환
- $x_{i\_new} = \frac{x_i}{\sqrt{x_i^2 + y_i^2 + z_i^2 + \dots}}$

③ StandardScaler
- 표준화를 쉽게 지원하기 위한 클래스
- SVM, 선형회귀, 로지스틱 회귀에서 표준화가 중요
- scaler.fit( ) → scaler.transform( ) ⇒ 표준화 결과가 np.ndarray로!

④ MinMaxScaler
- 데이터 값을 0과 1 사이 범위의 값으로 변환
- scaler.fit( ) → scaler.transform( ) ⇒ 결과: np.ndarray

⑤ 스케일링 시 유의점
↳ 학습 데이터로 fit( )이 적용된 스케일링 기준 정보를 그대로 테스트 데이터에 적용해야 함.
⇒ 데이터 스케일링 기준을 동일하게 적용!
↳ 테스트 데이터에서 다시 fit( )을 적용하면 X ⇒ 이미 fit( )이 적용된 Scaler 객체를 이용해 transform( )을 적용해야 함.
↳ 가능하다면 (전체) 데이터의 스케일링을 적용한 뒤 학습과 테스트 데이터로 분리하기