

2장 사이킷런

01 사이킷런 소개와 특징

특징

- 가장 파이썬스러운 API를 제공해서 다른 머신러닝 패키지도 사이킷런 스타일의 API를 지향함
- 머신러닝을 위한 매우 다양한 알고리즘, 개발을 위한 편리한 프레임워크와 API를 제공함
- 오랜 시간 검증되어왔음
- 매우 많은 환경에서 사용되는 성숙한 라이브러리

02 iris 품종 예측하기

용어 정리

- Supervised Learning: 학습을 위한 다양한 피쳐와 분류 결정값인 Label 데이터(명확한 답O)로 모델을 학습하고, 별도의 테스트 데이터 세트에서 미지의 레이블을 예측하는 것
- sklearn.dataset: 사이킷런에서 자체적으로 제공하는 데이터 세트를 생성하는 모듈의 모임
- sklearn.tree: 트리 기반 ML 알고리즘을 구현한 클래스 모임
- 하이퍼 파라미터: 머신러닝 알고리즘 별로 최적의 학습을 위해 직접 입력하는 파라미터
- sklearn.model_selection: 학습 데이터, 검증 데이터, 예측 데이터로 데이터를 분리하거나 최적의 하이퍼 파라미터로 평가하기 위한 다양한 모듈의 모임
- 정확도: 예측 결과가 실제 레이블 값과 얼마나 정확하게 맞는지 평가하는 지표

train_test_split(iris_data, iris_label, test_size=0.2, random_state=11)

- 학습용 데이터로 학습된 모델이 얼마나 뛰어난 성능을 가지는지 평가하려면 테스트 데이터셋이 필요함

- 파라미터 역할(순서대로): 학습용 피쳐 데이터셋, 테스트용 피쳐 데이터셋, 학습용 레이블 데이터셋, 테스트용 레이블 데이터셋
- `random_state`을 지정하지 않으면 수행할 때마다 다른 학습/테스트 용 데이터를 만들 수 있음
⇒ 동일하게 하려면 고유의 값으로 고정해야함

객체 `DecisionTreeClassifier`

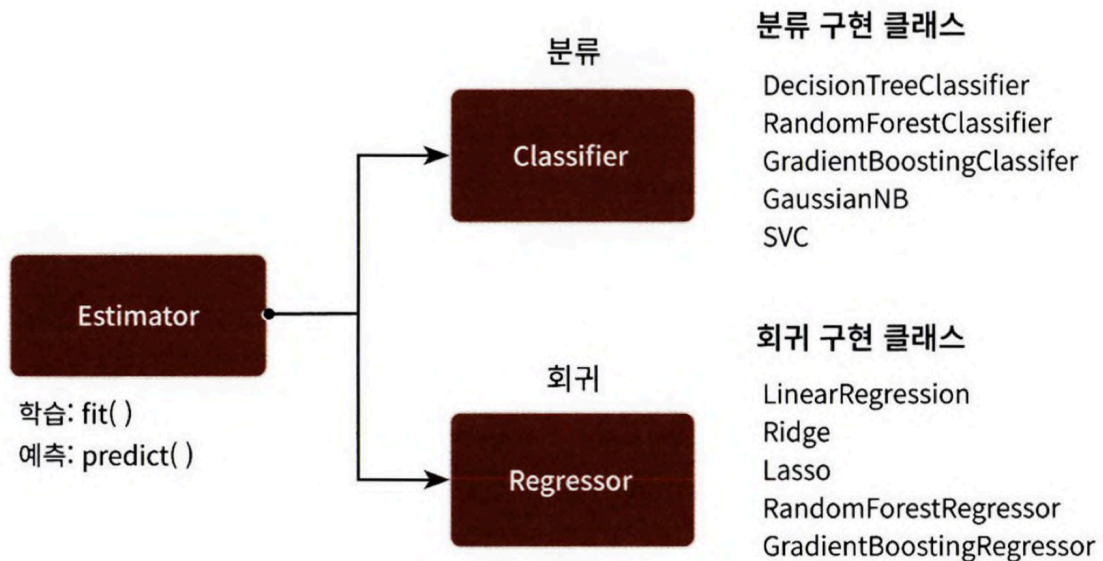
- `fit()` 메소드에 학습용 피쳐 데이터 속성과 결정값 데이터 셋을 입력해 호출하면 학습을 수행함
- 예측은 학습 데이터가 아닌 다른 데이터를 이용해야함! 주로 테스트 데이터셋 사용함..
- `predict()` 메소드에 테스트용 피쳐 데이터셋을 입력해 호출하면 학습된 모델 기반에서 테스트 데이터 셋에 대한 예측값을 반환하게 됨
- `accuracy_score()`: 정확도 측정하는 함수
파라미터(순서대로): 실제 레이블 데이터셋, 예측 레이블 데이터셋

프로세스 정리

1. 데이터셋 분리: 학습 데이터, 테스트 데이터로 데이터를 분리
2. 모델 학습: 학습 데이터를 기반으로 ML 알고리즘을 적용해 모델을 학습시킴
3. 예측 수행: 학습된 ML 모델을 이용해 테스트 데이터의 분류 예측
4. 평가: 예측된 결과값과 테스트 데이터의 실제 결과값을 비교해 ML 모델 성능 평가

03 사이킷런 기반 프레임워크 익히기

Estimator 이해 및 `fit()`, `predict()` 메소드



- Classifier: classification 알고리즘을 구현한 클래스
- Regressor: Regression 알고리즘을 구현한 클래스

⇒ 합쳐서 Estimator 클래스! .. 내부에서 fit(), predict()를 구현하고 있음

- evaluation 함수, 하이퍼 파라미터 튜닝을 지원하는 클래스는 Estimator를 인자로 받아서 메소드들을 호출해서 평가하거나 하이퍼 파라미터 튜닝을 수행함
- dimension reduction, clustering, feature extraction등을 구현한 클래스 역시 fit(), transform()을 적용함
 - 여기서 fit()은 입력 데이터의 형태에 맞춰 데이터를 변환하기 위한 사전 구조를 맞추는 작업
 - 위 과정을 거치면, 이후 입력 데이터의 dimension reduction, clustering, feature extraction등의 실제 작업은 transform()으로 수행
 - 사이킷런은 fit()과 transform()을 하나로 결합한 fit_transform()도 제공함
 - fit_transform()은 fit()과 transform()을 따로 호출할 필요가 없다는 장점이 있지만, 사용에 약간의 주의해야함

사이킷런 주요 모듈

분류	모듈명	설명
예제 데이터	<code>sklearn.datasets</code>	사이킷런에 내장되어 예제로 제공하는 데이터 세트
피처 처리	<code>sklearn.preprocessing</code>	데이터 전처리에 필요한 다양한 가공 기능 제공(문자열을 숫자 형 코드 값으로 인코딩, 정규화, 스케일링 등)
	<code>sklearn.feature_selection</code>	알고리즘에 큰 영향을 미치는 피처를 우선순위로 선택션 작업을 수행하는 다양한 기능 제공
피처 처리	<code>sklearn.feature_extraction</code>	텍스트 데이터나 이미지 데이터의 벡터화된 피처를 추출하는데 사용됨. 예를 들어 텍스트 데이터에서 Count Vectorizer나 Tf-Idf Vectorizer 등을 생성하는 기능 제공. 텍스트 데이터의 피처 추출은 <code>sklearn.feature_extraction.text</code> 모듈에, 이미지 데이터의 피처 추출은 <code>sklearn.feature_extraction.image</code> 모듈에 지원 API가 있음.
피처 처리 & 차원 축소	<code>sklearn.decomposition</code>	차원 축소와 관련한 알고리즘을 지원하는 모듈임. PCA, NMF, Truncated SVD 등을 통해 차원 축소 기능을 수행할 수 있음
데이터 분리, 검증 & 파라미터 튜닝	<code>sklearn.model_selection</code>	교차 검증을 위한 학습용/테스트용 분리, 그리드 서치(Grid Search)로 최적 파라미터 추출 등의 API 제공
평가	<code>sklearn.metrics</code>	분류, 회귀, 클러스터링, 페어와이즈(Pairwise)에 대한 다양한 성능 측정 방법 제공 Accuracy, Precision, Recall, ROC-AUC, RMSE 등 제공
ML 알고리즘	<code>sklearn.ensemble</code>	앙상블 알고리즘 제공 랜덤 포레스트, 에이다 부스트, 그래디언트 부스팅 등을 제공
	<code>sklearn.linear_model</code>	주로 선형 회귀, 릿지(Ridge), 라쏘(Lasso) 및 로지스틱 회귀 등 회귀 관련 알고리즘을 지원. 또한 SGD(Stochastic Gradient Descent) 관련 알고리즘도 제공
	<code>sklearn.naive_bayes</code>	나이브 베이즈 알고리즘 제공. 가우시안 NB, 다항 분포 NB 등.
	<code>sklearn.neighbors</code>	최근접 이웃 알고리즘 제공. K-NN 등
	<code>sklearn.svm</code>	서포트 벡터 머신 알고리즘 제공
	<code>sklearn.tree</code>	의사 결정 트리 알고리즘 제공
	<code>sklearn.cluster</code>	비지도 클러스터링 알고리즘 제공 (K-평균, 계층형, DBSCAN 등)
유틸리티	<code>sklearn.pipeline</code>	피처 처리 등의 변환과 ML 알고리즘 학습, 예측 등을 함께 묶어서 실행할 수 있는 유틸리티 제공

내장된 예제 데이터셋

- 일반적으로 딕셔너리 형태로 되어있음
- 피처의 데이터 값을 반환받기 위해서는 내장 데이터 세트 API를 호출한 뒤에 그 Key값을 지정하면 됨
- data: feature의 데이터셋을 가리킴.. numpy 배열
- target: classification: label값, regression; 숫자 결과값 데이터셋 .. numpy 배열
- target_names: 개별 label 이름 .. numpy 배열, list
- feature_names: feature 이름 .. numpy 배열, list
- DESCR: 데이터셋에 대한 설명과 각 피처의 설명 .. String

feature_names				target_names		
sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	setosa, versicolor, virginica	(0 , 1 , 2)	
data	5.1	3.5	1.4	0	target	
	4.9	3.0	1.4	1		
		
	4.6	3.1	1.5	2		
	5.0	3.6	1.4	0		

04 Model Selection 모듈

학습/테스트 데이터 세트 분리 - train_test_split()

- 이미 학습을 수행한 학습용 데이터 세트가 아니라 전용 테스트 데이터 세트로 예측 수행해야함
- train_test_split()으로 이용하면 데이터셋을 학습 데이터셋, 테스트 데이터셋으로 분리 가능함
- 주요 파라미터
 - 학습용 피처 데이터셋, 테스트용 피처 데이터셋, 학습용 레이블 데이터셋, 테스트용 레이블 데이터셋
 - 먼저 sklearn.model_selection 모듈에서 train_test_split을 로드합니다. train_test_split()는 첫 번

째 파라미터로 피쳐 데이터 세트, 두 번째 파라미터로 레이블 데이터 세트를 입력받습니다. 그리고 선택적으로 다음 파라미터를 입력받습니다.

- `test.size` : 전체 데이터에서 테스트 데이터 세트 크기를 얼마로 샘플링할 것인가를 결정함. 디폴트는 0.25
- `train_size` : 전체 데이터에서 학습용 데이터 세트 크기를 얼마로 샘플링함. `test_size parameter`를 통상적으로 사용하기 때문에 `train_size`는 잘 사용되지 않음
- `shuffle` : 데이터를 분리하기 전에 데이터를 미리 섞을지를 결정함. 디폴트값은 `True`. 데이터를 분산시켜서 좀 더 효율적인 학습 및 테스트 데이터 세트를 만드는 데 사용함
- `random_state` : `random_state`는 호출할 때마다 동일한 학습/테스트용 데이터 세트를 생성하기 위해 주어지는 난수 값.
- `train_test_split()`는 호출 시 무작위로 데이터를 분리하므로 `random_state`를 지정하지 않으면 수행할 때마다 다른 학습/테스트 용 데이터를 생성함. `train_test_split()`의 반환값은 튜플 형태. 순차적으로 학습용 데이터의 피쳐 데이터 세트, 테스트용 데이터의 피쳐 데이터 세트, 학습용 데이터의 레이블 데이터 세트, 테스트용 데이터의 레이블 데이터 세트가 반환됨.

교차 검증

- **Overfitting**: 모델이 학습 데이터에만 과도하게 최적화 되어, 실제 예측을 다른 데이터로 수행할 경우에 예측 성능이 과도하게 떨어지는 것
⇒ 해결방법: 교차 검증
- 교차 검증: 데이터 편종을 막기 위해 별도의 여러 세트로 구성된 학습 데이터셋과 검증 데이터셋에서 학습과 평가를 수행하는 것
- 각 세트에서 수행한 평가 결과에 따라 하이퍼 파라미터 튜닝 등 모델 최적화를 쉽게 할 수 있음
- 데이터셋을 세분화해서 학습, 검증, 테스트 데이터셋으로 나눌 수 있음
- 검증 데이터 세트: 최종 평가 이전에 학습된 모델을 다양하게 평가하는 데 사용됨



K 폴드 교차 검증

- K개의 데이터 폴드 셋을 만들어서 K번 만큼 각 폴드 셋에 학습과 검증 평가를 반복적으로 수행하는 작업
- 과정



- 예측 평가를 구했으면 평균내서 K 폴드 평가 결과로 반영함
- 사이킷런에서 KFold, StratifiedKFold 클래스를 제공
- KFold 객체는 split()을 호출하면 학습용/검증용 데이터로 분할할 수 있는 인덱스를 반환함. 교차 검증 수행 시마다 학습과 검증을 반복해 예측 정확도를 측정함
- 실제로 학습용/검증용 데이터 추출은 반환된 인덱스를 기반으로 개발 코드에서 직접 수행해야함.

Stratified K 폴드

- imbalanced 분포도를 가진 label 데이터 집합을 위한 K 폴드 방식
- imbalanced 분포도를 가진 label: 특정 label 값이 특이하게 많거나 매우 적어서 값의 분포가 한쪽으로 치우치는 것
- K 폴드가 레이블 데이터 집합이 원본 데이터 집합의 레이블 분포를 학습 및 테스트 세트에 제대로 분배하지 못하는 문제를 해결해
- 프로세스: 원본 데이터의 레이블 분포를 먼저 고려한 뒤 이 분포와 동일하게 학습과 검증 데이터 세트를 분배함
- Stratified K 폴드는 원본 데이터의 레이블 분포도 특성을 반영한 학습 및 검증 데이터셋을 만들 수 있음
=> 왜곡된 레이블 데이터셋서는 반드시 Stratified K 폴드를 이용해 교차검증해야함
- 회귀의 결정값은 연속된 숫자값 ⇒ Stratified K 폴드가 지원X

GridSearchCV - 교차 검증과 최적 하이퍼 파라미터 튜닝을 한 번에

- 교차 검증을 기반으로 이 하이퍼 파라미터의 최적 값을 찾게 해줌
- 데이터 세트를 cross-validation을 위한 학습/테스트 세트로 자동으로 분할한 뒤에 하이퍼 파라미터 그리드에 기술된 모든 파라미터를 순차적으로 적용해 최적의 파라미터를 찾을 수 있게 해줌
- 사용자가 튜닝하고자 하는 여러 종류의 하이퍼 파라미터를 다양하게 테스트하면서 최적의 파라미터를 편리하게 찾게 해주지만 동시에 순차적으로 파라미터를 테스트하므로 수행시간이 상대적으로 오래 걸리는 것에 유념해야 합니다.
- 학습 데이터를 GridSearchCV# 이용해 최적 하이퍼 파라미터 튜닝을 수행한 뒤에 별도의 테스트 세트에서 이를 평가하는 것이 일반적인 머신러닝 모델 적용 방법입니다.
- 주요 파라미터
 - estimator : classifier, regressor, pipeline
 - param_grid : key + 리스트 값을 가지는 딕셔너리가 주어짐. estimator의 튜닝을 위해 파라미터명과 사용될 여러 파라미터 값을 지정함
 - scoring : 예측 성능을 측정할 평가 방법을 지정함. 보통은 사이킷런의 성능 평가 지표를 지정하는 문자열 (예: 정확도의 경우 'accuracy') 로 지정하나 별도의 성

능 평가 지표 함수도 지정할 수 있음

- cv : 교차 검증을 위해 분할되는 학습/테스트 세트의 개수를 지정함
- refit : 디폴트가 True이며 True로 생성 시 가장 최적의 하이퍼 파라미터를 찾은 뒤 입력된 estimator 객체를 해당 하이퍼 파라미터로 재학습시킴

05 데이터 전처리

NaN, Null 값은 허용되지 않음

- Null 값이 얼마 안됨 ⇒ feature의 평균값 등으로 대체
- Null 값이 대부분 ⇒ drop the feature

사이킷런은 문자열 값을 입력값으로 허용하지 않음

- 모든 문자열 값은 인코딩돼서 숫자형으로 변환해야함
- 문자열 피처: 카테고리형 피처, 텍스트형 피처
- 텍스트형 피처 - 피처 벡터화 등의 기법으로 벡터화하거나 불필요하다고 판단하면 삭제

데이터 인코딩

- label encoding: 카테고리 피처(문자열 값) → 코드형 숫자(숫자형)
- One Hot encoding
 - 피처 값의 유형에 따라 새로운 피처를 추가해 고유 값에 해당하는 칼럼 → 1, 나머지 → 0
 - 행 형태로 돼 있는 피처 고유값 → 열 형태 (차원 변환), 고유 값에 해당하는 칼럼에 만 1표시, 나머지 칼럼에는 0 표시
 - 판다스에는 get_dummies()가 같은 역할을 함

피처 스케일링과 정규화

- 피처 스케일링: 서로 다른 변수의 값 범위를 일정한 수준으로 맞추는 작업
 - 표준화: 가우시안 정규 분포로 변환하는 것

$$x_{i_new} = \frac{x_i - \text{mean}(x)}{\text{stdev}(x)}$$

- 정규화: 서로 다른 피처의 크기를 통일하기 위해 크기를 변환해주는 것..

$$x_{i_new} = \frac{x_i - \min(x)}{\max(x) - \min(x)}$$

사이킷런에서는 개별 벡터의 크기를 맞추기 위한 변환해주는 것을 의미함!

$$x_{i_new} = \frac{x_i}{\sqrt{x_i^2 + y_i^2 + z_i^2}}$$

StandardScaler

- 표준화를 쉽게 지원하기 위한 클래스
- 사이킷런에서 구현한 RBF 커널을 이용하는 Support Vector Machine, Linear Regression, Logistic Regression은 데이터가 가우시안 분포를 가지고 있다고 가정하고 구현됨
→ 사전에 표준화를 적용하는 것은 예측 성능 향상에 중요한 요소가 될 수 있습니다.

MinMaxScaler

- 데이터값을 0과 1 사이의 범위 값으로 변환
- 음수 값이 있으면 -1 에서 1값으로 변환
- 가우시안 분포가 아닐때 적용

학습 데이터와 테스트 데이터의 스케일링 변환 시 유의점

- Scaler 객체를 이용해 데이터의 스케일링 변환 시 fit(), transform(), fit_transform() 메소드를 이용함
- fit(): 데이터 변환을 위한 기준 정보 설정
- transform(): 설정된 정보를 이용해 데이터 변환

- `fit_transform()`: `fit()`, `transform()` 한번에 적용, 테스트 데이터에서는 절대 사용하면 안됨
- 학습 데이터로 `fit()`이 적용된 스케일링 기준 정보를 그대로 테스트 데이터에 적용해야 함
- 테스트 데이터로 다시 새로운 스케일링 기준 정보를 만들게 되면, 학습 데이터와 테스트 데이터의 스케일링 기준 정보가 서로 달라지기 때문에 올바른 예측 결과를 도출하지 못할 수도 있음
- 유의점 요약
 1. 가능하다면 전체 데이터의 스케일링 변환을 적용한 뒤 학습과 테스트 데이터로 분리
 2. 1이 여의치 않다면 테스트 데이터 변환 시에는 `fit()`이나 `fit_transform()`을 적용하지 않고 학습 데이터로 이미 `fit()`된 `Scaler` 객체를 이용해 `transform()`의 변환