2. 사이킷런으로 시작하는 머신러닝

goblurry · 방금 전

데이터분석 머신러닝 파머완

파이썬 머신러닝 완벽가이드

목록 보기 2/2

📊 파이썬 머신러닝 완벽 가이드 (위키북스, 개정 2판) 교재를 바탕으로 학습한 내용을 정리한 포스트입니다.
https://github.com/goblurry/ML-Study 에서 예시 코드를 확인할 수 있습니다.

1. Scikit-learn 소개

사이킷런(scikit-learn): 파이썬 머신러닝 라이브러리 중 가장 많이 사용되는 라이브러리.
아나콘다 환경에서 다음과 같이 실행할 수 있다. 참고로 본 포스팅에서는 1.6.1 버전을 사용하였다.

```
conda install scikit-learn
import sklearn
```

2. 첫 번째 머신러닝 만들기: 붓꽃 품종 예측하기

붓꽃 데이터 세트로 붓꽃의 품종을 분류(Classification) - 이 데이터 세트는 꽃잎의 길이와 너비, 꽃받침의 너비와 길이 피처를 기반으로 꽃의 품종을 예측한다.

분류(Classification)

- 대표적인 **지도학습** 방법의 하나

```
 - Label 데이터(학습을 위한 다양한 피처와 분류 결정값)로 모델을 학습한 뒤, 별도의 테스트 데이터 세트에서 미지의 레이블 예측
- 정답이 주어진 데이터를 학습한 뒤 미지의 정답을 예측하는 방식
- 학습 데이터 세트: 학습을 위해 <u>주어진</u> 데이터 세트
- 테스트 데이터 세트: 학습 후 머신러닝 모델의 예측 성능을 평가하기 위해 주어진 데이터 세트
```

사이킷런의 모듈

사이킷런 패키지 내의 모듈명은 sklearn으로 시작하는 명명 규칙이 있다.

1. sklearn.datasets 내 모듈: 사이킷런에서 자체 제공하는 데이터셋을 생성하는 모듈의 모임
2. sklearn.tree 내 모듈: 트리 기반 머신러닝 알고리즘을 구현한 클래스의 모임
3. sklearn.model_selection: 학습 데이터와 검증 데이터, 예측 데이터로 데이터를 분리하거나, 최적의 하이퍼 파라미터로 평가하기 위한 모듈의 모임.
    - 하이퍼 파라미터: 머신러닝 알고리즘별로 최적의 학습을 위해 직접 입력하는 파라미터. 이를 통해 머신러닝 알고리즘의 성능을 튜닝한다.

교재에서는 붓꽃 데이터셋 생성에는 load_iris()를 사용, 머신러닝 알고리즘은 의사결정 트리로, 이를 구현한 DecisionTreeClassifier를 적용한다. 또, 데이터셋을 학습/테스트 데이터로 분리하는 데는 train_test_split() 함수를 사용한다.

```
# 모듈 임포트하기
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
```

사이킷런을 활용해 분류 예측하기

1. **데이터 세트 분리:** 학습 & 테스트 데이터로 분리
    - test_test_split 메서드를 활용하며, 이때 테스트 데이터의 비율은 입력 파라미터 중 'test_size=0.x'로 결정할 수 있다.
2. **모델 학습:** 학습 데이터를 기반으로 머신러닝 알고리즘을 적용해 모델 학습시키기
    - 의사결정트리 객체의 fit(학습용 피처 데이터 속성, 결정 값 데이터 세트) 메서드 호출해 학습 수행
3. **예측 수행:** 학습된 머신러닝 모델을 이용해 테스트 데이터의 분류를 예측하기
    - 의사결정트리 객체의 predict(테스트용 피처 데이터 세트) 호출하면 학습된 모델 기반에서 테스트 데이터 세트에 대한 예측값 반환
4. **평가:** 예측된 결괏값과 테스트 데이터의 실제 결괏값을 비교해 머신러닝 모델의 성능 평가
    - 정확도: `from sklearn.metrics import accuracy_score`
    - accuracy_score(실제 레이블 데이터 세트, 예측 레이블 데이터 세트)

3. 사이킷런의 기반 프레임워크 익히기

> 1. Estimator 이해 및 fit(), predict() 메서드
> 2. 사이킷런의 주요 모듈
> 3. 내장된 예제 데이터 세트

Estimator 이해 및 fit(), predict() 메서드

사이킷런이 제공하는 두 개의 핵심 메서드: fit(), predict()

- fit(): 머신러닝 모델 학습용
- predict(): 학습된 모델 예측용

이 두 개의 메서드만을 이용해, 사이킷런 클래스들은 간단하게 학습과 예측 결과를 반환한다.
사이킷런은 지도학습의 두 축인 **분류**와 **회귀**의 다양한 알고리즘을 구현하고 있는데, 분류 알고리즘을 구현한 클래스를 **Classifier**, 회귀 알고리즘을 구현한 클래스를 **Regressor**라고 지칭한다. 또, 이 둘을 합쳐서 **Estimator**라고 부른다.
이 Estimator 클래스는 fit(), predict() 메서드를 내부에서 구현한다.
평가/하이퍼파라미터 튜닝을 수행하는 함수 및 클래스에서는 이 Estimator를 인자로 받는다.

사이킷런의 주요 모듈

지금은 잘 모르지만 자주 쓰이는 핵심 모듈을 정리한 표. 실습하다 보면 하나하나 알게 될 것!

분류	모듈명	설명
예제 데이터	sklearn.datasets	사이킷런에 내장되어 예제로 제공하는 데이터 세트
피처 처리	sklearn.preprocessing	데이터 전처리에 필요한 다양한 가공 기능 제공
	sklearn.feature_selection	알고리즘에 영향 미치는 피처를 우선순위대로 셀렉션 작업 수행하는 기능 제공
	sklearn.feature_extraction	텍스트 데이터/이미지 데이터의 벡터화된 피처를 추출하는 데 사용
피처 처리&차원 축소	sklearn.decomposition	차원 축소 관련 알고리즘 지원하는 모듈
데이터 분리, 검증&파라미터 튜닝	sklearn.model_selection	교차 검증을 위한 학습용/데이터용 분리, 그리드 서치로 최적 파라미터 추출 등의 API 제공
평가	sklearn.metrics	분류, 회귀, 클러스터링, 페어와이즈에 대한 다양한 성능 측정 방법 제공
ML 알고리즘	skleaarn.ensemble	앙상블 알고리즘 제공: 랜덤 포레스트, 에이다 부스트, 그래디언트 부스팅 등 제공
	sklearn.linear_model	선형회귀, Ridge, Lasso, 로지스틱 회귀 등 회귀 관련 알고리즘 지원
	sklearn.naive_bayes	나이브 베이즈 알고리즘 제공: 가우시안, 다항 분포 NB 등
	sklearn.neigbors	최근접 이웃 알고리즘: K-NN
	sklearn.svm	서포트 벡터 머신 알고리즘 제공
	sklearn.tree	의사결정트리 알고리즘 제공
	sklearn.cluster	비지도 클러스터링 알고리즘 제공: K-평균, 계층형 등
유틸리티	sklearn.pipeline	피처 처리 등 변환과 ML 알고리즘 학습, 예측 등을 함께 묶어 실행할 수 있는 유틸리티 제공

> **머신러닝 모델을 구축하는 주요 프로세스:**
> 1. 피처 처리: 가공, 변경, 추출
> 2. ML 알고리즘 학습/예측 수행
> 3. 평가
> 이 단계를 반복적으로 수행하는 것.

내장된 데이터 세트

저번에 Pandas 실습 때는 캐글에서 데이터셋을 다운받아야 했다. 사이킷런에는 그런 과정 없이, 예제로 활용 가능한 데이터 세트가 내장되어 있다. 이 데이터는 datasets 모듈(상단의 표에도 있음)에 있는 여러 API를 호출해 만들 수 있다.
사이킷런에 내장된 데이터셋은 (1) 분류/회귀 연습용 예제 데이터셋 (2) 분류/클러스터링을 위해 표본 데이터로 생성될 수 있는 데이터셋 둘로 나뉜다.

1. 분류, 클러스터링을 위한 표본 데이터 생성기

API 명	설명
datasets.make_classifications( )	분류를 위한 데이터 세트를 만듭니다. 특히 높은 상관도, 불필요한 속성 등의 노이즈 효과를 위한 데이터를 무작위로 생성해 줍니다.
datasets.make_blobs( )	클러스터링을 위한 데이터 세트를 무작위로 생성해 줍니다. 군집 지정 개수에 따라 여러 가지 클러스터링을 위한 데이터 세트를 쉽게 만들어 줍니다.

2. 분류, 회귀 연습용

분류나 회귀 연습용 예제 데이터

API 명	설명
datasets.load_boston()	회귀 용도이며, 미국 보스턴의 집 피처들과 가격에 대한 데이터 세트
datasets.load_breast_cancer()	분류 용도이며, 위스콘신 유방암 피처들과 악성/음성 레이블 데이터 세트
datasets.load_diabetes()	회귀 용도이며, 당뇨 데이터 세트
datasets.load_digits()	분류 용도이며, 0에서 9까지 숫자의 이미지 픽셀 데이터 세트
datasets.load_iris()	분류 용도이며, 붓꽃에 대한 피처를 가진 데이터 세트

인터넷에서 내려받아 scikit_learn_data라는 서브 디렉터리에 저장한 후 불러와야 하는 데이터도 있다 - fetch 계열의 명령을 사용한다.

- fetch_covtype(): 회귀 분석용 토지 조사 자료
- fetch_20newsgroups(): 뉴스 그룹 텍스트 자료
- fetch_olivetti_faces(): 얼굴 이미지 자료
- fetch_lfw_people(): 얼굴 이미지 자료
- fetch_lfw_pairs(): 얼굴 이미지 자료
- fetch_rcv1(): 로이터 뉴스 말뭉치
- fetch_mldata(): ML 웹사이트에서 다운로드

이렇게 사이킷런에 내장된 분류, 회귀 연습용 데이터 세트는 일반적으로 **딕셔너리** 형태로 구성돼 있다.
Key: data, target, target_name, feature_names, DESCR로 보통 구성된다.

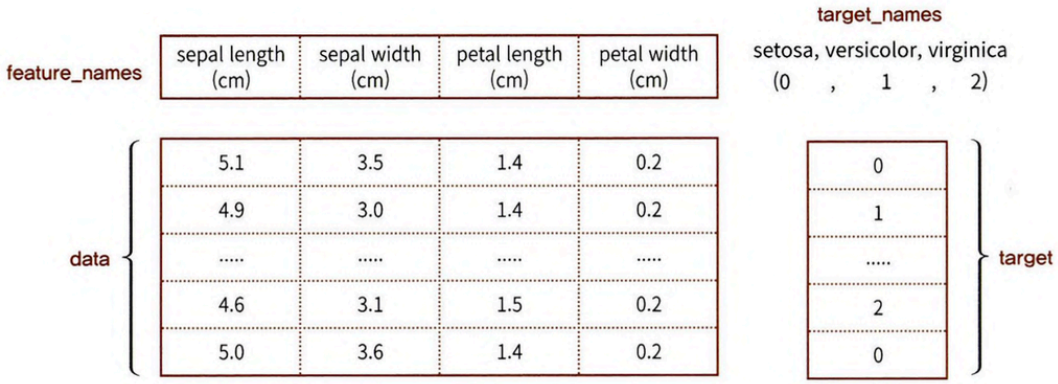
- data: 피처의 데이터 세트 (ndarray)
- target: 분류 시에는 label 값(정답), 회귀일 때는 숫자 결괏값 데이터 세트 (ndarray)
- target_names: 개별 label의 이름 (ndarray / python list)
- feature_names: 피처의 이름 (ndarray / python list)
- DESCR: 데이터 세트에 대한 설명과 각 피처의 설명 (string)

```
from sklearn.datasets import load_iris

iris_data = load_iris()
print(type(iris_data))
```

이렇게 붓꽃 데이터 세트를 생성하면, Bunch를 반환하는데, 이는 파이썬의 딕셔너리 자료형과 유사하다. 이 데이터 세트의 Key값을 확인해 볼 수 있는데, 주요 key 값으로는 **data, target, target_names, feature_names**가 있다. iris를 `iris_data` 변수에 load했으므로, iris_data.keys()를 호출해 이 데이터 세트에 어떤 키들이 있는지 확인할 수 있다.

이 예시에서 data, target, target_name, feature_names가 의미하는 바는 다음과 같다.



- feature_names: 꽃잎의 길이와 너비, 꽃받침의 너비와 길이 피처
- target_name: setosa, versicolor, virginica (붓꽃의 종류)
- target: 0, 1, 2. 위의 타겟들을 숫자로 표현한 것
- data: 각 feature의 데이터 세트로, 꽃잎의 길이와 너비, 꽃받침의 너비와 길이 값을 저장

```python
# load_iris()가 반환하는 객체의 키들이 가리키는 값들

# 1. feature_names
print('feature_names의 타입:', type(iris_data.feature_names))
print('feature_names의 shape:', len(iris_data.feature_names))
print(iris_data.feature_names)

# 2. target_names
print('\ntarget_names의 type:', type(iris_data.target_names))
print('target_names의 shape:', len(iris_data.target_names))
print(iris_data.target_names)

# 3. data
print('\ndata의 type:', type(iris_data.data))
print('data의 shape:', len(iris_data.data))
print(iris_data['data'])

# 4. target
print('\ntarget의 type:', type(iris_data.target))
print('target의 shape:', len(iris_data.target))
print(iris_data.target)
```

```
feature_names의 타입: <class 'list'>
feature_names의 shape: 4
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']

target_names의 type: <class 'numpy.ndarray'>
target_names의 shape: 3
['setosa' 'versicolor' 'virginica']

data의 type: <class 'numpy.ndarray'>
data의 shape: 150
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
```

```
target의 type: <class 'numpy.ndarray'>
target의 shape: 150
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

load_iris()가 반환하는 객체의 키들이 가리키는 값들을 확인할 수 있다.

4. Model Selection 모듈 소개

> 1. 학습/테스트 데이터 세트 분리: train_test_split()
> 2. 교차 검증
> 3. GridSearchCV: 교차 검증 + 최적 하이퍼 파라미터 튜닝

학습/테스트 데이터 세트 분리: train_test_split()

학습과 테스트에 동일한 데이터 세트를 활용하고 정확도를 출력하면 1.00이 출력된다. 학습을 수행한 데이터 세트가 아닌 **테스트 전용** 데이터 세트를 활용해야 한다. → **train_test_split()** 을 통해 원본 데이터 세트에서 학습/테스트용을 쉽게 분리할 수 있다. 이 메서드는 튜플 형태로 반환된다.

이 메서드는 sklearn.model_selection 모듈에서 로드할 수 있다. 첫 번째 파라미터로 **피처 데이터 세트**, 두 번째 파라미터로 **레이블 데이터 세트**를 입력받는다.
그리고 선택적으로 test_size(전체 데이터 중 테스트 데이터셋의 크기. 디폴트는 0.25), train_size(테스트 사이즈 파라미터를 입력하면 자동으로 결정되므로 잘 사용하지 않음), shuffle(데이터 분리 전에 미리 섞을지 여부를 결정. 데이터가 분산되면 더 효율적인 학습/테스트 데이터 세트 생성이 가능함. 디폴트 true), random_state(호출 시마다 동일한 데이터 세트를 생성하기 위해 주어지는 난수 값. 이를 지정하지 않으면 스플릿 메서드를 수행할 때마다 다른 데이터가 생성된다.)

테스트 데이터셋의 크기가 크지 않을 때에는 알고리즘의 예측 성능을 판단하는 게 적절하지 않다. 학습을 위한 데이터의 양을 일정 수준 이상으로 보장하는 것, 또 학습된 모델에 대해 다양한 데이터를 기반으로 예측 성능을 평가하는 것이 중요하다.

교차 검증

과적합(Overfitting) : 모델이 학습 데이터에만 과하게 최적화되어, 실제 예측을 다른 데이터로 수행할 경우 예측의 성능이 저하되는 현상 (데이터 편중)

고정된 학습/테스트 데이터로 평가하다 보면 이러한 경향이 생기게 되고, 이를 개선하기 위해 교차 검증을 수행해 더 다양한 학습과 평가를 수행한다.

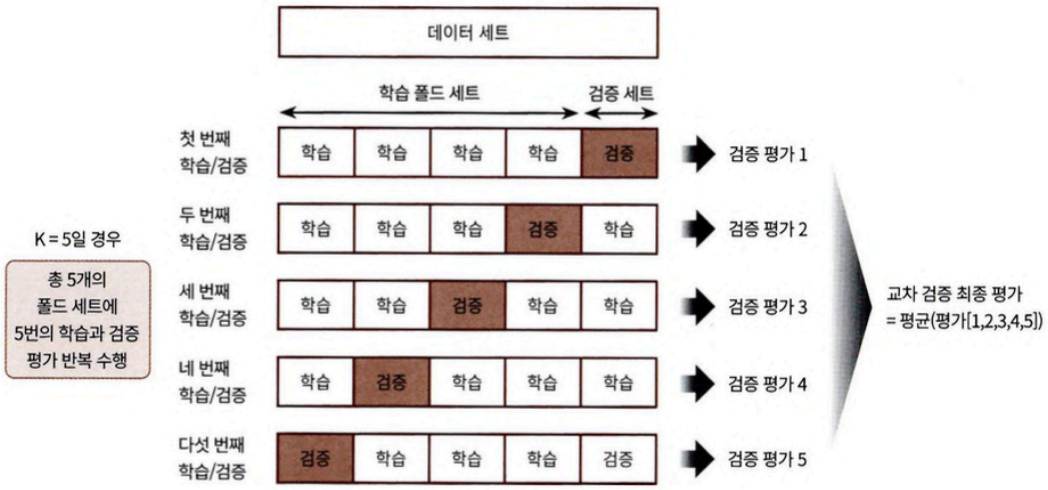
교차 검증 : 여러 세트로 구성된 학습 데이터 세트와 검증 데이터 세트에서 학습과 평가를 수행하는 것. 각 세트에서 수행한 평가 결과에 따라 하이퍼 파라미터 튜닝 등의 모델 최적화를 쉽게 할 수 있다.

ML 모델의 성능 평가: 교차 검증 기반 1차 평가 → 최종적으로 테스트 데이터 세트에 적용해 평가
ML에 사용되는 데이터 세트를 세분화해 학습, 검증, 테스트 데이터 세트로 나눌 수 있음. (테스트 데이터 세트 외에 별도의 **검증** 데이터 세트를 두어 학습된 모델을 다양하게 평가한다.)

① K 폴드 교차 검증
② Straitified K 폴드
③ cross_val_score()

K 폴드 교차 검증

가장 보편적인 교차 검증 기법. **K개의 데이터 폴드 세트를 만들어 K번만큼 각 폴드 세트에 학습 & 검증 평가를 반복 수행하는 방법** (데이터 폴드: 하나의 데이터를 여러 조각으로 나누는 것)



위 사진과 같이, K개로 나눈 데이터에서 학습 데이터 세트와 검증 데이터 세트를 K번 변경하면서 학습과 검증을 수행한다. K개의 예측 평가를 구했으면 이를 평균하여 평가 결과로 반영한다.

```
from sklearn.model_selection import KFold
kfold = KFold(n_splits=K) // K에는 원하는 개수 입력
```

<검증 세트>

- 학습에 쓰이지 않은 데이터 조각
- 모델은 이 데이터로 예측 → 정답과 비교 → 성능 지표 계산 과정을 거침
- 검증한다: 학습된 모델이 보지 못한 데이터에서 얼마나 정답을 맞히는지 확인한다.

사이킷런에서는 이를 구현하기 위해 **KFold, StratifiedKFold** 클래스를 제공한다. 더 자세한 내용은 깃허브의 코드를 참고할 것.

Stratified K 폴드

불균형한 분포도를 가진 레이블 데이터 집합을 위한 K 폴드 방식. 분포도가 불균형하다는 것은 특정 레이블 값이 특이하게 많거나 적어서 값의 분포가 한쪽으로 치우치는 것을 의미한다.
레이블 데이터 집합이 원본 데이터 집합의 레이블 분포를 학습 및 테스트 세트에 제대로 분배하지 못하는 K 폴드 방식의 문제를 해결한다.
핵심 원리: 원본 데이터의 레이블 분포를 먼저 고려한 뒤, 이 분포와 동일하게 학습/검증 데이터 세트를 분배한다.
(참고: 회귀에서는 Stratified K 폴드가 지원되지 않는다.)

StratifiedKFold의 split()을 호출할 땐 레이블 데이터 세트도 반드시 넣어 주어야 한다. (KFold는 피처 데이터셋만 입력)

```
for train_index, test_index in skfold.split(features, label):
    # split()으로 반환된 인덱스를 이용해 학습용, 검증용 테스트 데이터 추출
    X_train, X_test = features[train_index], features[test_index]
    y_train, y_test = label[train_index], label[test_index]
```

cross_val_score()

교차 검증을 더 편리하게 수행할 수 있게 해주는 API이다. 앞서 K 폴드로 데이터를 학습하고 예측하는 코드에서,

1. 폴드 세트 설정
2. for 루프에서 반복으로 학습 및 테스트 데이터의 인덱스 추출
3. 학습과 예측 수행, 예측 성능 반환을 반복

이 세 단계를 거쳤다. **cross_val_score()** 는 이 과정을 한 번에 수행하는 API이다.

```
# 이 API의 선언 형태
cross_val_score(estimator, X, y=None, scoring=None, cv=None, n_jobs=1, verbose=0, fit_para
```

에서 주요 파라미터는 estimator, X, y, scoring, cv이다.

- estimator: 사이킷의 분류 알고리즘 클래스 classifier 혹은 회귀 알고리즘 클래스 Regressor
- X: 피처 데이터셋
- y: 레이블 데이터셋
- scoring: 예측 성능 평가 지표
- cv: 교차 검증 폴드 수

이 메서드가 반환하는 값은 scoring 파라미터로 지정된 성능 지표 측정값의 배열 형태.
cross_val_score()는 classifier가 입력되면 Stratified K 폴드 방식으로 데이터 세트를 분할한다.
(Regressor: K 폴드 방식)

```
# 성능 지표: accuracy / 교차 검증 세트 3개
scores = cross_val_score(dt_clf, data, label, scoring='accuracy', cv=3)
```

이 메서드는 cv로 지정된 횟수만큼 평과 결괏값을 배열로 반환하고, 이를 평균해 평가 수치로 사용한다.
API 내부에서 estimator를 학습, 예측, 평가하기 때문에 간단하게 교차 검증을 수행할 수 있다! (비슷하게 cross_validate()라는 API가 있는데, 이는 여러 개의 평가 지표를 반환한다.)

GridSearchCV: 교차 검증과 최적 하이퍼 파라미터 튜닝

하이퍼 파라미터: 머신러닝 알고리즘을 구성하는 주요 요소로, 이 값을 조정해 알고리즘의 예측 성능을 개선한다. (결정트리에서의 max_depth와 같은 값)

사이킷런은 **GridSearchCV** API를 이용해 회귀나 분류 알고리즘에 사용되는 하이퍼 파라미터를 순차적으로 입력하면서 편리하게 최적의 파라미터를 도출할 수 있는 방안을 제공한다. 그 기반이 되는 것이 교차 검증이다.
데이터 세트를 교차검증을 위한 학습/테스트 세트로 자동 분할 한 뒤에 하이퍼 파라미터 그리드에 기술된 모든 파라미터를 순차적으로 적용해 최적의 파라미터를 찾는다.
즉, 모델 성능을 최대화하는 하이퍼파라미터를 자동으로 탐색하는 방법.

<GridSearchCV 클래스의 생성자로 포함되는 주요 파라미터>

- estimator: classifier/regressor/pipeline
- param_grid: key+ list 값 가지는 딕셔너리 주어짐. estimator 튜닝을 위해, 파라미터명과 여러 파라미터 값을 지정함
- scoring: 예측 성능을 측정할 평가 방법. 보통은 accuracyd나 같은 사이킷런의 성능 평가 지표 문자열. 별도의 성능 평가 지표 함수도 가능함.
- cv: 교차검증을 위해 분할되는 학습/테스트 데이터 세트의 개수
- refit: 디폴트는 True. 이 경우 최적의 하이퍼 파라미터를 찾은 뒤, 입력된 estimator 객체를 해당 하이퍼 파라미터로 재학습시킴.

데이터 전처리

머신러닝 알고리즘은 데이터에 기반하기 때문에 입력으로 어떤 데이터를 갖는지에 따라 결과도 달라질 수 있다.

<미리 처리해야 할 기본사항>
1. 결손값(Null, NaN) 허용하지 않음.
2. 문자열 값을 입력으로 가지지 않으므로 인코딩해 숫자형으로 변환해야 함.
- 텍스트형 피처: 피처 벡터화 등 기법으로 벡터화하거나 삭제
- 카테고리형 피처: 코드 값으로 표현

데이터 인코딩: 레이블 인코딩, 원-핫 인코딩

레이블 인코딩

- 카테고리 피처를 코드형 숫자 값으로 변환하는 것.
- 상품 구분이 과일, 야채, 고기로 되어있다면 이를 과일: 1, 야채: 2, 고기: 3처럼 숫자형 값으로 구분하는 것
- `LabelEncoder` 클래스로 구현 - 객체 생성 후, fit()과 transform() 호출해 레이블 인코딩을 수행한다.

```
# 레이블 인코딩
from sklearn.preprocessing import LabelEncoder

items=['TV', '냉장고', '전자레인지', '컴퓨터', '선풍기', '선풍기', '믹서', '믹서']

# LabelEncoder를 객체로 생성한 뒤, fit()과 transform()으로 레이블 인코딩 수행
encoder = LabelEncoder()
encoder.fit(items)
labels = encoder.transform(items)
print('인코딩 변환값:', labels)
```

```
인코딩 변환값: [0 1 4 5 3 3 2 2]
```

```
# 위 예제는 데이터 크기가 작아서 문자열이 어떤 숫자로 인코딩됐는지 알 수 있지만 데이터가 많은 경우에는 classes_ 속성값을 이용한다.
# 인덱스 0부터 순서대로 출력
print('인코딩 클래스:', encoder.classes_)

# transform()으로 디코딩
print('디코딩 원본값:', encoder.inverse_transform([4,5,2,0,1,1,3,3]))
```

```
인코딩 클래스: ['TV' '냉장고' '믹서' '선풍기' '전자레인지' '컴퓨터']
디코딩 원본값: ['전자레인지' '컴퓨터' '믹서' 'TV' '냉장고' '냉장고' '선풍기' '선풍기']
```

하지만 간혹 레이블 인코딩을 적용할 경우 몇몇 머신러닝 알고리즘에서 예측 성능이 떨어지는 경우가 있다.
(숫자의 크고 작음에 대한 특성이 적용되어 큰 수에 가중치가 부여될 가능성이 있음)
이런 경우 때문에 선형회귀와 같은 ML 알고리즘에는 적용해서는 안 된다. 이러한 문제를 해결하는 인코딩 방식이 원-핫 인코딩이다.

원-핫 인코딩(One-Hot Encoding)

피처 값의 유형에 따라 새로운 피처를 추가해, 고유 값에 해당하는 컬럼에만 1을 표시하고 나머지에는 0을 표시하는 방식

원본 데이터

상품 분류
TV
냉장고
전자레인지
컴퓨터
선풍기
선풍기
믹서
믹서

원-핫 인코딩

상품분류_ TV	상품분류_ 냉장고	상품분류_ 믹서	상품분류_ 선풍기	상품분류_ 전자레인지	상품분류_ 컴퓨터
1	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	1	0
0	0	0	0	0	1
0	0	0	1	0	0
0	0	0	1	0	0
0	0	1	0	0	0
0	0	1	0	0	0

왼쪽처럼 행 형태로 되어 있는 피처의 고유 값을 열의 형태로 차원 변환한 뒤, 고유 값에 해당하는 컬럼에만 1을 표시하는 것이다.

이는 사이킷런에서 **OneHotEncoder** 클래스로 변환이 가능하다.

- 2차원 데이터가 입력값으로 필요
- OneHotEncoder를 이용해 변환한 값이 희소 행렬 형태이므로 이를 다시 toarray() 메서드를 이용해 **밀집 행렬**로 변환해야 한다.

피처 스케일링과 정규화

서로 다른 변수의 값 범위를 일정한 수준으로 맞추는 작업을 피처 스케일링(feature scaling) 이라고 한다.

표준화 (Standardization)

- 각 피처가 평균 0, 분산 1이 되도록 변환하는 방식.
- 데이터의 분포를 정규 분포(가우시안 분포)와 유사하게 만들어주는 효과가 있다.
- 서포트 벡터 머신(SVM), 선형 회귀(Linear Regression), 로지스틱 회귀(Logistic Regression) 같은 알고리즘에서 중요한 역할을 한다.
- 표준화를 통해 학습 데이터의 스케일 차이가 줄어들어, 가중치 계산 시 특정 피처가 과도하게 영향력을 갖는 문제를 방지할 수 있다.

$$x_i\_new = \frac{x_i - mean(x)}{stdev(x)}$$

정규화 (Normalization)

- 서로 다른 크기를 가진 데이터를 0-1 범위로 맞추는 방식.
- 피처의 최소값을 0, 최대값을 1로 변환하여 스케일을 통일한다.
- 모든 데이터가 동일한 스케일에서 비교될 수 있도록 해주며, 거리 기반 알고리즘(KNN, K-means 등)에서 효과적이다.
- 값의 분포가 일정하지 않더라도, 상대적인 크기를 일정하게 만드는 데 활용된다.

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

사이킷런에서 제공하는 Normalizer 모듈

- 개별 벡터의 크기를 맞추기 위한 변환: 개별 벡터를 모든 피처 벡터의 크기로 나눔.
- 3개의 피처 x, y, z가 있다면 새로운 데이터 x_new는 원래 값에서 세 개의 피처의 i번째 피처값에 해당하는 크기를 합한 값으로 나눔.

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

StandardScaler

- **표준화 전용 클래스**: 각 피처의 평균을 0, 표준편차를 1로 맞춘다.
- 데이터가 가우시안 분포를 가진다고 가정하는 알고리즘에서 효과적이다.
- 결과적으로 학습 데이터가 **단위 분산**을 가지게 되어, 모델 훈련 시 수렴 속도를 높이고 안정성을 확보할 수 있다.

MinMaxScaler

- **정규화 전용 클래스**: 데이터 값을 0과 1 사이의 범위로 변환한다.
- 데이터 분포의 형태를 그대로 유지하면서도 스케일만 조정하기 때문에 직관적이다.
- 단, 이상치가 존재할 경우 최소/최대값이 크게 왜곡되어 전체 변환 결과가 영향을 받을 수 있다.

학습 데이터와 테스트 데이터의 스케일링 변환 시 유의점

1. 학습 데이터와 테스트 데이터를 분리한 후, 반드시 **학습 데이터에 대해서만** fit() 하여 스케일 기준(평균, 표준편차, 최소/최대값 등)을 구해야 한다.
2. 테스트 데이터에는 transform()만 적용해야 한다.
3. 테스트 데이터까지 fit() 해버리면 실제 상황에서는 얻을 수 없는 정보가 포함되어 데이터 누수 문제가 발생할 수 있다.

머신러닝 프로젝트는 단순히 알고리즘을 선택하고 적용하는 것으로 끝나지 않는다. 실제로는 데이터 전처리와 준비 과정이 모델 성능에 큰 영향을 미친다.
결측값 처리, 인코딩, 스케일링과 정규화 같은 작업을 통해 데이터를 정리해야만, 알고리즘이 안정적으로 작동하고 의미 있는 결과를 도출할 수 있다.
또한 학습 데이터와 테스트 데이터를 올바르게 분리하고, 교차 검증을 수행하여 모델의 일반화 성능을 평가하는 절차가 필수적이다. 단순히 하나의 고정된 데이터셋으로 평가할 경우 과적합이나 편향된 결과가 발생할 수 있기 때문에, 다양한 검증 방식을 통해 모델을 반복적으로 점검하고 개선해야 한다.
이러한 일련의 과정은 단순히 기계적으로 거치는 절차가 아니라, 데이터의 특성과 알고리즘의 성격을 이해하고 조율하는 과정이라고 할 수 있다. 데이터 품질을 확보하고, 적절한 전처리 방법을 선택하며, 교차 검증을 통해 모델을 평가하고 개선하는 과정을 반복해야만 신뢰할 수 있는 머신러닝 모델을 구축할 수 있다.

goblurry

이전 포스트
1. 파이썬 기반의 머신러닝과 생태계 ...

0개의 댓글

댓글을 작성하세요

댓글 작성