[SESSION 1] 파이썬 머신러닝 및 사이킷런

파이썬 머신러닝 완벽 가이드 P.1-P.129

1주차 발표자 : 박세은

목차

1. 파이썬 기반의 머신러닝과 생태계 이해

넘파이

- 넘파이 모듈 불러오기

  import numpy as np

- 넘파이 함수 및 메서드 정리

  ndarray 배열 생성

np.array( )	인자를 받아 ndarray로 변환
np.arange( )	0부터 (인자-1)까지 1차원 ndarray 생성
np.zeros()	0으로 shape만큼의 ndarray 생성
np.ones( )	1로 shape만큼의 ndarray 생성

넘파이

ndarray 속성 확인

ndarray.shape	Ndarray의 행, 열 출력
ndarray.ndim	Ndarray의 차원 출력
ndarray.dtype	Ndarray의 데이터 타입 출력
ndarray.reshape( )	입력받은 shape의 ndarray로 재생성

넘파이

넘파이 데이터 세트 선택하기 : indexing

1. 특정 데이터만 추출 : 원하는 위치 인덱스 값을 지정해 해당 위치 데이터 반환
2. 슬라이싱 : ‘:’을 이용해 연속된 인덱스상 ndarray 추출하는 방식

* 0:10은 0부터 9까지

넘파이

넘파이 데이터 세트 선택하기 : indexing

3. 팬시 인덱싱: 리스트나 ndarray로 인덱싱 집합을 지정하면 해당 위치의 인덱스에 해당하는 ndarray를 반환하는 방식

array2d[[0,1],2]

4. 불린 인덱싱 : 특정조건에 따라 true/falsee 값 인덱싱 집합을 기반으로 True에 해당되는 데이터의 ndarray 반환

Array1d[array1 > 5]

넘파이

행렬의 정렬

1. np.sort( )

- 넘파이에서 sort( )를 호출하는 방식
- 원 행렬을 그대로 유지한 채 원 행렬의 정렬된 행렬을 반환 (ndarry 원본 유지)

2. ndarray.sort( )

- 행렬 자체에서 sort( )를 호출하는 방식
- 원 행렬 자체를 정렬한 상태로 변환 (반환값 None/ ndarray 원본 대체)

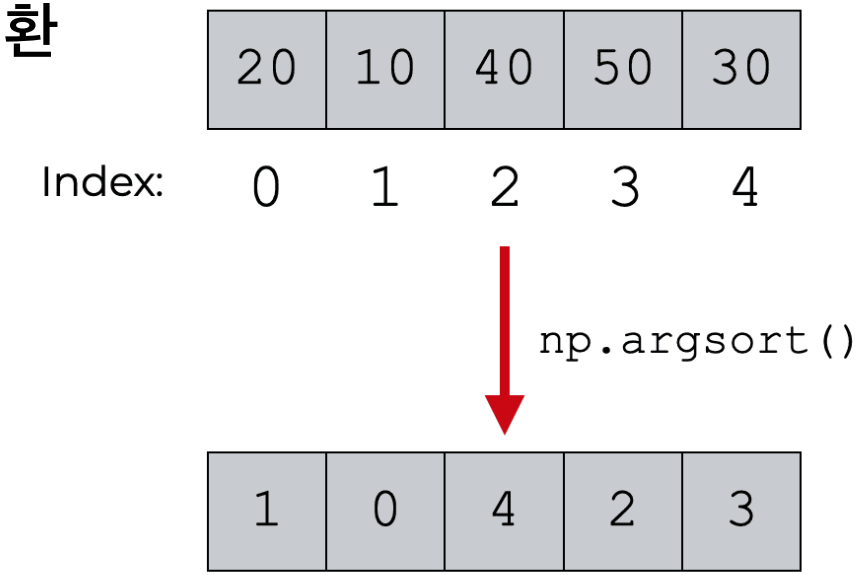
넘파이

행렬의 정렬

3. np.argsort( )

`np.argsort()` RETURNS THE INDEX VALUES
IN AN ORDER THAT WOULD SORT THE INPUT ARRAY

- 정렬 행렬의 원본 행렬 인덱스를 ndarray 형으로 변환



넘파이

선형대수 연산 : 행렬 내적과 전치 행렬 구하기

1. np.dot( )

- 행렬 내적(행렬 곱)
- 왼쪽 행렬의 열 개수와 오른쪽 행렬의 행 개수가 동일해야 함

판다스

기본 이해

- Pandas 핵심 개체 : DataFrame
- Series와 DataFrame

Series : 칼럼이 하나뿐인 데이터 구조체

DataFrame: 칼럼이 여러 개인 데이터 구조체 (여러 개의 Series)

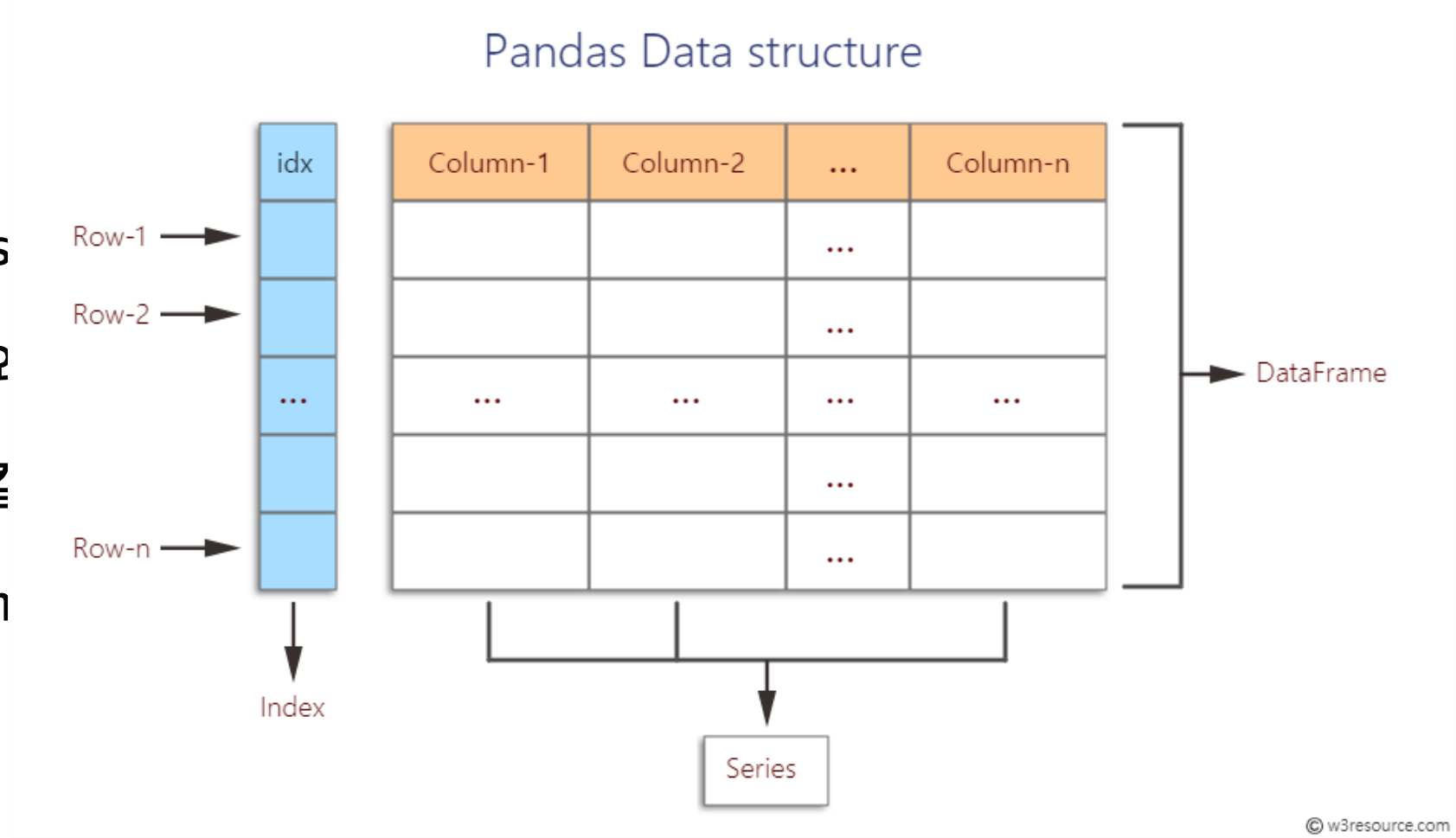
판다스

기본 이해

- Pandas
- Series와

Series : 칼

DataFran



판다스

판다스 시작 : 파일을 DataFrame으로 로딩, 기본 API

1. 판다스 모듈 임포트

Import pandas pd

2. DataFrame 불러오기

pd.read_csv( )	Csv 파일 포맷을 위한 API 디폴트 필드 구분 문자 : , Sep 인자를 활용 (sep='\t')
pd.read_table( )	디폴트 필드 구분문자 : \t
pd.read_fwf( )	고정 길이 기반 칼럼 포맷 대상 (잘 사용 XX)

판다스

판다스 시작 : 파일을 DataFrame으로 로딩, 기본 API

3. DataFrame 정보 확인하기

DataFrame.head( )	DataFrame 맨 앞에 있는 N개 row 반환 (default : 5)
DataFrame.shape)	DataFrame의 행과 열을 튜플 형태로 반환
DataFrame.info( )	총 데이터 건수와 칼럼별 데이터 타입, null 건수
DataFrame.describe( )	숫자형 칼럼에 대한 개략적인 데이터 분포도 (non-null 건수, mean, std, min, max, quantile)
DataFrame['열].value_counts( )	해당 칼럼값의 유형과 건수 확인

판다스

DataFrame의 칼럼 데이터 세트 생성 수정

1. [ ] 연산자 이용

- DataFrame[‘열‘] = 숫자
- DataFrame[‘열'] = 수식

판다스

DataFrame 데이터 삭제

1. DataFrame.drop( )

- Labels: drop할 칼럼이나 행 지정 ([ ] 안에 넣어서 입력)
- Axis : 특정 칼럼 또는 행 drop (axis=0: row, axis=1: column)
- inplace: drop한 데이터프레임을 원본으로 저장 (default: False)

  : inplace = True 일 경우 반환값 None/ 원본이 새로운 DF로 대체됨.

판다스

Index 객체

1. DataFrame.index

- DataFrame의 인덱스 객체 추출
- 출력값 : RangeIndex(start=0, stop=891, step=1)

2. DataFrame.index.values

- DataFrame의 인덱스 객체를 실제 값 array로 변환
- 출력값:

[[0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17

18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35

36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 … 889 890]]

판다스

Index 객체

1. DataFrame.reset_index( )

- DataFrame 인덱스를 새롭게 연속 숫자형 (0~)으로 할당 (고유 인덱스 잃어버림)
- 기존 인덱스 'index'라는 새로운 칼럼명으로 추가
- 주요 파라미터

  (i) drop : 기본 인덱스를 새로운 칼럼으로 추가하지 않고 삭제 (default = False)

  (ii) inplace: reset_index한 데이터프레임을 원본으로 저장(default = False)

판다스

데이터 셀렉션 및 필터링

1. [ ] 연산자

- DataFrame의 칼럼만 지정하는 ‘칼럼 지정 연산자‘
- 단일 칼럼 추출

```
titanic_df['Pclass']
```

- 여러 칼럼 추출 ([ ] 안에 해당 칼럼들이 포함된 [ ] 입력)

```
titanic_df[['Survived', 'Pclass']]
```

- 불린 인덱싱 가능

```
titanic_df[titanic_df['Pclass']==3]
```

판다스

데이터 셀렉션 및 필터링

2. DataFrame.iloc[ ]

- 위치 기반 인덱싱
- 행과 열 값으로 integer 혹은 integer형의 슬라이싱, 팬시 인덱싱만 가능
- 칼럼 명칭을 입력하면 오류 발생

```
Data_df.iloc[0, 0]
```

판다스

데이터 셀렉션 및 필터링

4. 불린 인덱싱

- [ ] 연산자와 DataFrame.loc[ ] 사용
- 복합 조건 결합해 사용 가능 (&, |, ~)
- [ ] 연산자의 경우 : DataFrame[[조건][열 이름들]]

```
titanic_df[titanic_df['Age'] > 60][['Name', 'Age']]
```

- .loc[ ]의 경우: DataFrame/loc[[조건], [열 이름들]]

```
titanic_df.loc[titanic_df['Age'] > 60, ['Name', 'Age']]
```

판다스

정렬, Aggregation 함수, GroupBy 적용

1. DataFrame.sort_values( )

- DataFrame과 Series 정렬
- 주요 파라미터

    (i) by : 정렬의 기준이 될 열(들)을 리스트형으로 입력

    (ii) ascending : 오름차순으로 정렬 여부 (default: True)

    (iii) inplace : 정렬한 데이터프레임을 원본으로 저장 (default : false)

```
titanic_sorted = titanic_df.sort_values(by=['Pclass', 'Name'], ascending=False)
```

판다스

정렬, Aggregation 함수, GroupBy 적용

2. Aggregation 함수

- 기본 연산을 도와주는 함수들의 집합 (min, max, sum, median, count)
- DataFrame 전체 혹은 특정 칼럼들에 해당 aggregation 적용
- 전체 칼럼들 : 데이터프레임 뒤에 함수 적용

  titanic_df.count()

- 특정 칼럼들 : 데이터프레임[[칼럼들]] 뒤에 함수 적용

  titanic_df[['Age', 'Fare']].mean( )

- Axis 설정에 따라 적용되는 방향 달라짐(axis =0: 행/ axis =1: 열)

판다스

정렬, Aggregation 함수, GroupBy 적용

3. DataFrame.groupby( )

- 입력 파라미터 by에 칼럼을 입력하면 대상 칼럼을 groupby
- 반환된 DataFrame Groupby 객체에 aggregation 함수를 호출
  -> groupby( ) 대상 칼럼을 제외한 모든 칼럼에 해당 aggregation 함수 적용
  -> 특정 칼럼들에만 적용: .groupby()[열].함수( ) / .groupby( )[[열들]].함수( )
- .agg( ) 을 이용해 여러 개의 aggregation 함수 적용

하나의 칼럼에 여러 개의 함수: .agg([함수들])

```
titanic_df.groupby('Pclass')['Age'].agg([max, min])
```

칼럼에 따라 서로 다른 함수 : .agg(dic())

```
agg_format = {'Age':'max', 'SibSp':'sum', 'Fare':'mean'}
titanic_df.groupby('Pclass').agg(agg_format)
```

판다스

결손 데이터 처리하기

1. DataFrame.isna( )

- 모든 칼럼 값이 NaN인지 아닌지 true/false로 반환
- 결손 데이터 개수를 확인하기 위해서는 DataFrame.isna( ).sum( )

2. DataFrame.fillna( )

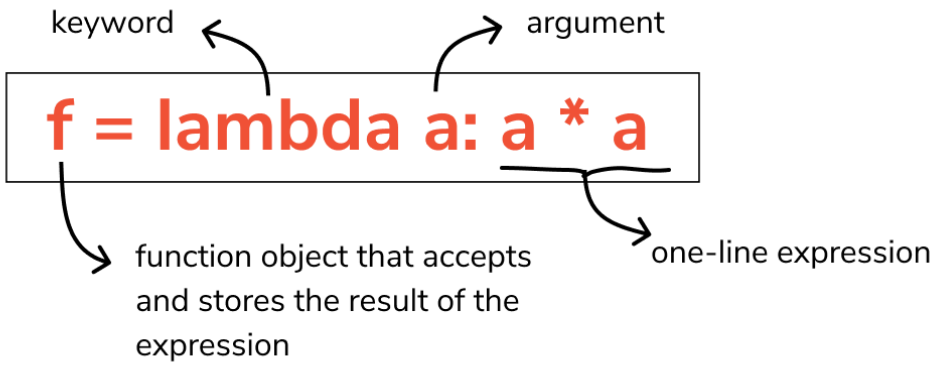
- 결손 데이터를 다른 값으로 대체 가능
- 특정 칼럼에 대해서만 적용 : DataFrame[열].filllna() / DataFrame[[열]].fillna( )
- Inplace = True 혹은 다른 이름으로 저장해야 filna( ) 실행 값 저장됨

판다스

Apply lambda 식으로 데이터 가공

1. Lambda

- 함수의 선언과 함수 내의 처리를 한 줄로 변환한 식



2. DataFrame.apply(lambda 식)

- 특정 칼럼에 대해서만 적용 : DataFrame[열].apply( ) / DataFrame[[열]].apply( )
- If else문 : list comprehension 사용
- 너무 길어질 경우 따로 함수 정의하는 것도 방법

2. 사이킷런으로 시작하는 머신러닝

첫 번째 머신러닝 만들어보기

프로세스 정리

1. 데이터 세트 분리: 데이터를 학습 데이터와 테스트 데이터로 분리

2. 모델 학습 : 학습 데이터 기반으로 ML 알고리즘 적용해 모델 학습

3. 예측 수행 : 학습된 ML 모델을 이용해 테스트 데이터 예측

4. 평가 : 예측된 결과값과 테스트 데이터의 실제 결과값 비교해 모델 성능 평가

사이킷런의 기반 프레임워크 익히기

Estimator 이해 및 fit( ), predict( ) 메서드

1. 지도 학습

- 분류(Classification)과 회귀(Regression)로 구성 -> Estimator 클래스
- Fit()과 predict( ) 내부 구현
- Evaluation 함수, 하이퍼파라미터 튜닝 클래스의 경우 이를 인자로 받음

사이킷런의 기반 프레임워크 익히기

Estimator 이해 및 fit( ), predict( ) 메서드

2. 비지도 학습

- 차원 축소, 클러스터링, 피처 추출(Feature Extraction) 등의 클래스
- Fit( )과 transform( ) 내부 구현 (하나로 결합한 fit_transform( ) 제공)
  - fit(): 입력 데이터의 형태에 맞춰 데이터를 변환하기 위한 사전 구조 맞춤
  - Transform(): 이후 입력 데이터의 차원 변환, 클러스터링 등 실제 작업 수행

사이킷런의 기반 프레임워크 익히기

사이킷런의 주요 모듈

1. 피처 처리

Sklearn.preprocessing	데이터 전처리에 필요한 기능
Sklearn.feature_selection	알고리즘에 큰 영향을 미치는 feature를 우선순위대로 셀렉션 작업 수행
Sklearn.feature_extraction	텍스트 데이터와 이미지 데이터의 벡터화된 feature 추출

사이킷런의 기반 프레임워크 익히기

사이킷런의 주요 모듈

2. 피처 처리 & 차원 축소

Sklearn.decomposition	차원 축소 관련된 알고리즘	PCA, NMF, Truncated SVD

3. 데이터 분리, 검증 & 파라미터 튜닝

Sklearn.model_selection	교차 검증을 위한 학습/테스트 분리, 최적 파라미터 추출	Train_test_split, GridSearchCV

사이킷런의 기반 프레임워크 익히기

사이킷런의 주요 모듈

4. 평가

Sklearn.metrics	성능 측정 방법 제공	Accuracy, Precision, Recall, ROC-AUC, RMSE 등

사이킷런의 기반 프레임워크 익히기

사이킷런의 주요 모듈

5. ML 알고리즘

Sklearn.ensemble	앙상블 알고리즘 제공	RandomForest, Adaboost, Gradient boosting 등
Sklearn.linear_model	회귀 알고리즘 제공	선형회귀, 릿지, 라쏘, 로지스틱 등
Sklearn.naïve_bayes	나이브 베이즈 알고리즘 제공	가우시안 NB, 다항분포 NB 등

사이킷런의 기반 프레임워크 익히기

사이킷런의 주요 모듈

5. ML 알고리즘

Sklearn.neighbors	최근접 이웃 알고리즘 제공	KNN
Sklearn.svm	서포트 벡터 머신 알고리즘 제공	
Sklearn.tree	의사 결정 트리 알고리즘 제공	Decision tree 등
Sklearn.cluster	비지도 클러스터링 알고리즘 제공	K-평균, 계층형, DBSCAN

Model Selection 모듈 소개

학습/테스트 데이터 세트 분리 - train_test_split()

1. train_test_split( )

   From sklearn.model_selection import train_test_split

- 학습/테스트 데이터 세트 분리
- 반환값: train_X, test_X, train_y, test_y 튜플 형태
- 주요 파라미터

  test_size

  Shuffle

  Random_state

Model Selection 모듈 소개

교차 검증

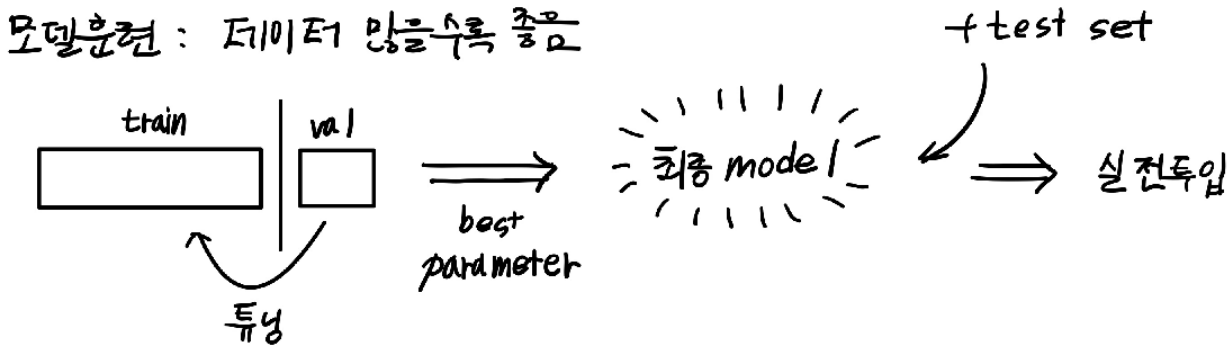
- 과적합(Overfitting) 방지
- 데이터 편중을 막기 위해 별도의 여러 세트를 구성된 학습/테스트 세트에서 학습과 평가 수행

Model Selection 모듈 소개

교차 검증

- 과적합(Overfit
- 데이터 편중을

평가 수행



너 학습과

트

학습

Model Selection 모듈 소개

교차 검증

1. K 폴드 교차 검증

- K개의 데이터 폴드 세트를 만들어 k번 만큼 각 폴드 세트에 학습과 검증 평가를 반복적으로 수행

  From sklearn.model_selection import Kfold

- N_splits 파라미터를 이용해 데이터 세트를 지정

  Kfold = kfold(n_splits = 5)

Model Selection 모듈 소개

교차 검증

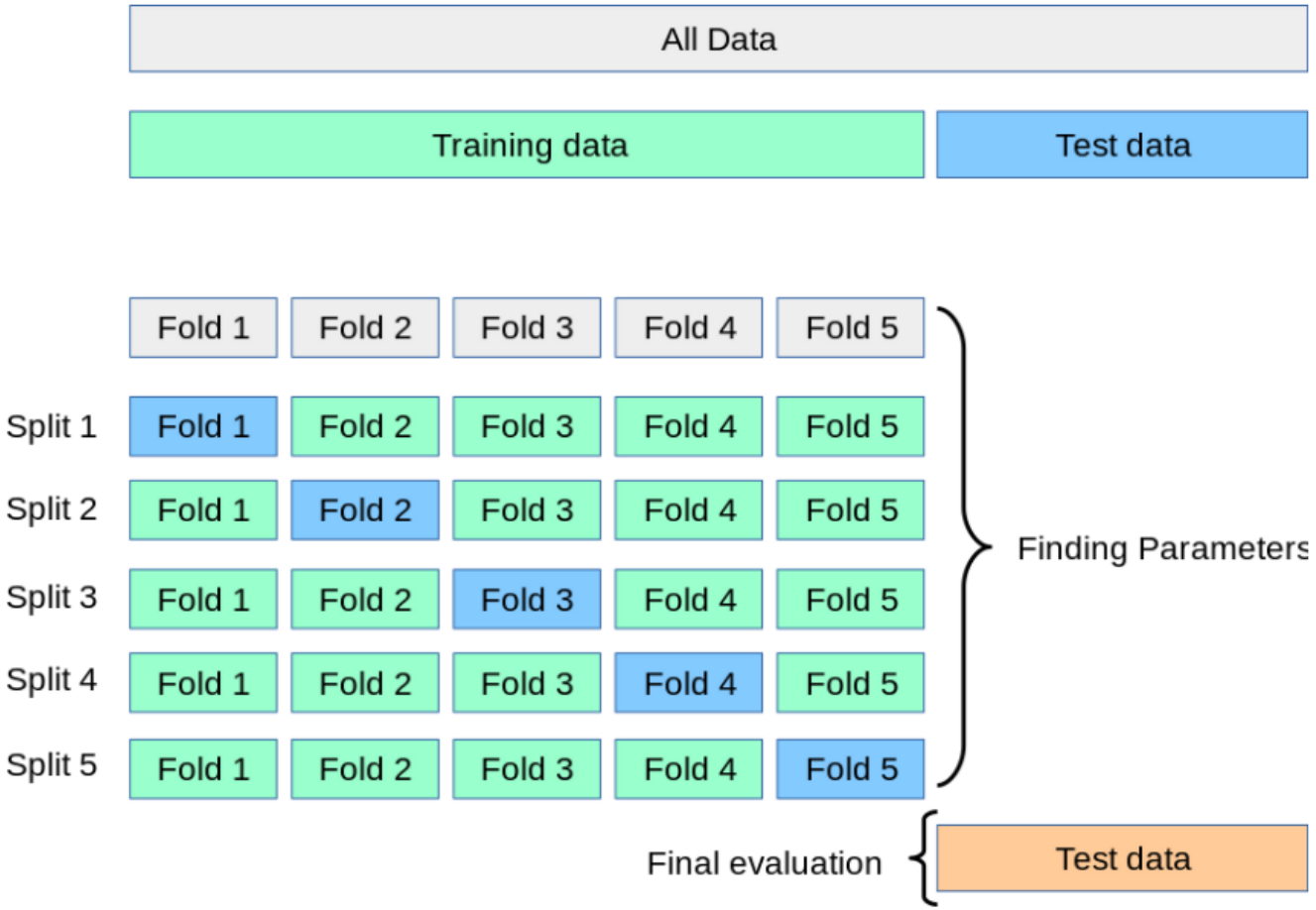
1. K 폴드 교차 검

- K개의 데이터 복적으로 수행

  From sklea

- N_splits 파

  Kfold = kfo



Model Selection 모듈 소개

교차 검증

2. Stratified K 폴드

- 이산값 형태의 레이블(Y)을 가진 데이터 한해 적용 - 분류, 로지스틱 회귀 등

  From sklearn.model_selection import StratifiedkFold

- 불균형한 (imbalanced) 분포도를 가진 Y 데이터 집합을 위한 k 폴드 방식
- 원본 데이터의 Y 분포를 먼저 고려한 뒤 이 분포와 동일하게 학습/검증 데이터 분배

  Skfold = StrtifiedKFold(n_splits=3)

Model Selection 모듈 소개

교차 검증

2. Stratified K 폴드

교차 검증 과정

1. 폴드 세트 설정

2. for 루프에서 반복적으로 학습/검증용 인덱스 추출

3. 학습과 예측 수행해 예측 성능 반환

Model Selection 모듈 소개

교차 검증

3. cross_val_score( )

- 교차 검증 과정을 한꺼번에 수행해주는 API

  From sklearn.model_selection import cross_val_score, cross_validate

- 반환값 : 배열 형태의 지정된 성능 지표 측정값
- 주요 파라미터
  - Estimator: Classifer 또는 Regressor
  - X: 피처 데이터 세트(X에 해당하는 데이터)
  - y: 레이블 데이터 세트 (target 값)
  - Scoring: 예측 성능 평가 지표
  - Cv: 교차 검증 폴드 수 (default : 5)

Model Selection 모듈 소개

교차 검증

3. cross_validate( )

- 교차 검증 과정을 한꺼번에 수행해주는 API

  From sklearn.model_selection import cross_val_score, cross_validate

- 반환값 : dict 형태의 각 폴드별 test_score, fit_time 등 반환
- 주요 파라미터
  - Estimator: Classifer 또는 Regressor
  - X: 피처 데이터 세트(X에 해당하는 데이터)
  - y: 레이블 데이터 세트 (target 값)
  - Scoring: 예측 성능 평가 지표 - [‘accuracy’, ‘roc-auc’]처럼 리스트 형으로 여러 개 지정 가능
  - Cv: 교차 검증 폴드 수 (default : 3)
  - Return_train_score: 훈련 폴드에 대한 점수(train_score)와 score_time(default: True)

Model Selection 모듈 소개

GridSearchCV : 교차검증과 최적 하이퍼 파라미터 튜닝을 한 번에

1. 기본 정보

- 교차 검증을 기반으로 하이퍼 파라미터의 최적 값을 찾는 api
- 주요 파라미터
  - Estimator: Classifer 또는 Regressor, pipeline
  - Param_grid: (dict 형태) estimator 튜닝을 위해 파라미터명과 여러 파라미터 값들 지정
  - Scoring: 예측 성능 평가 지표
  - Cv: 교차 검증을 위해 분할되는 학습/테스트 세트의 개수
  - Refit : 가장 최적의 하이퍼 파라미터를 찾은 뒤 estimator 객체에 재학습 (default: True)

Model Selection 모듈 소개

GridSearchCV : 교차검증과 최적 하이퍼 파라미터 튜닝을 한 번에

2. 과정

1. 파라미터 조합들 중 하나를 선택해 모델 생성

2. 입력받은 cv 만큼 교차검증해 성능 평가

3. 모든 파라미터 조합들에 대해 이 과정을 반복

4. 가장 성능이 좋은 파라미터 조합을 채택

Model Selection 모듈 소개

GridSearchCV : 교

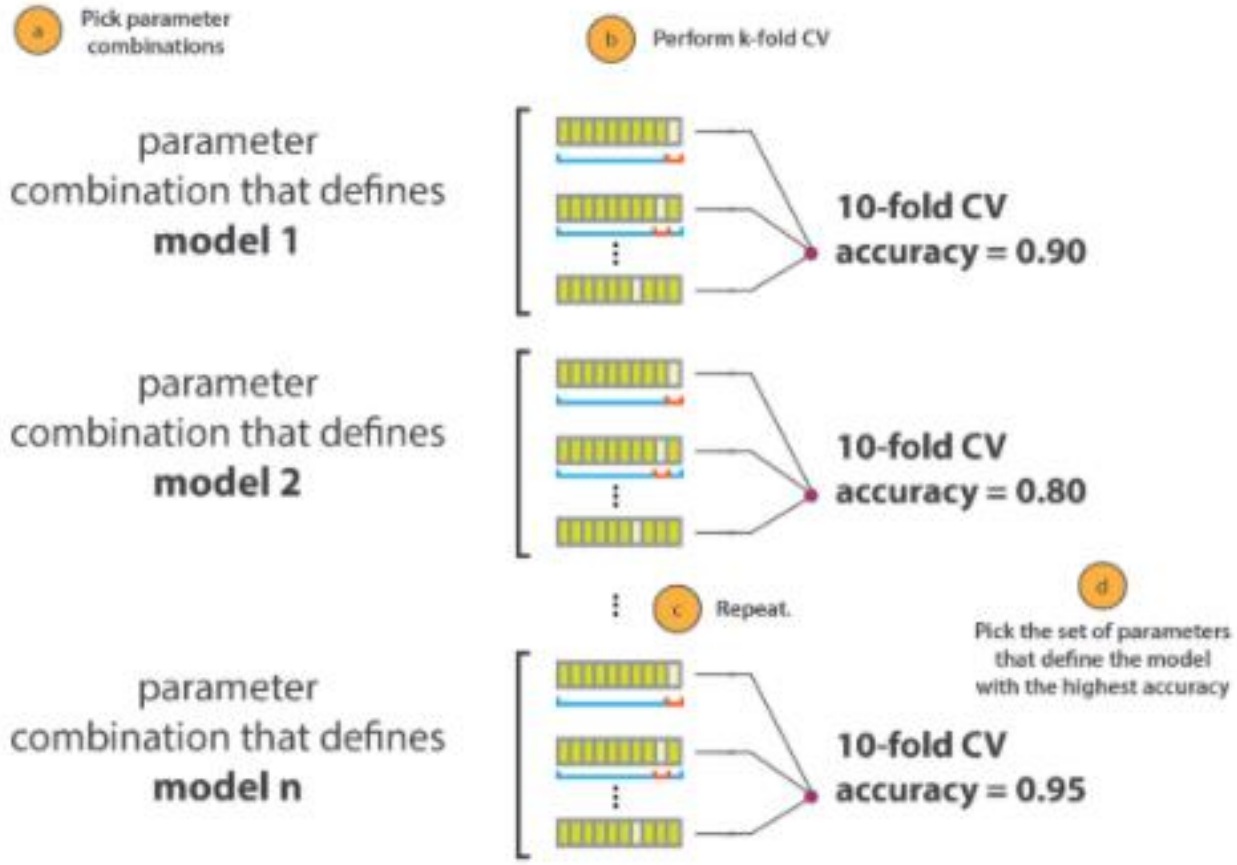
2. 과정

1. 파라미

2. 입력빋

3. 모든 교

4. 가장 성



Model Selection 모듈 소개

GridSearchCV : 교차검증과 최적 하이퍼 파라미터 튜닝을 한 번에

3. attributes

cv_results_	(dict 형태) 파라미터별 score	grid.cv_results_
best_estimators_	(estimator) 파라미터가 최적화된 모델	grid.best_estimators_
best_score	(float) 가장 좋은 모델의 성능 평균값	grid.best_score_
best_params_	(dict) 최적의 파라미터	grid.best_params_

데이터 전처리

결손값 (Null 값) 처리

1. 피처들 중 Null 값이 적은 경우

   해당 피처의 평균값 등으로 대체

   이때 피처의 중요도에 따라 대체값 선정에 유의

2. 피처들 중 Null 값 많은 경우

   해당 피처를 drop하는 게 일반적

데이터 전처리

데이터 인코딩

1. LabelEncoder

- 문자열 값을 숫자형 카테고리 값으로 변환

```
From sklearn.preprocessing import LabelEncoder
```

- 일괄적인 숫자값으로 변환되면서 특정 ML 알고리즘에서는 예측 성능 저하 문제 발생
- 트리 계열의 ML 알고리즘에서 사용 추천

데이터 전처리

데이터 인코딩

2. One-Hot Encoding

- 행 형태의 피처 고유 값을 열 형태로 차원 변환

```
from sklearn.preprocessing import OneHotEncoder
```

- 고유 값에 해당하는 칼럼에만 1, 나머지는 0 표시

데이터 전처리

데이터 인코딩

2. pd.get_dummies( )

- 판다스에서 제공하는 인코딩 방법
- 데이터 프레임을 인수로 받아야 함

데이터 전처리

피처 스케일링

1. 표준화

- 각 피처 값을 가우시안 정규 분포(평균=0, 분산 =1)를 가진 값으로 변환
- StandardScaler 이용
- 특히 SVM, 선형회귀 , 로지스틱 회귀 적용하기 전 꼭 수행

데이터 전처리

피처 스케일링

2. 정규화

- 서로 다른 피처의 크기를 0과 1 사이 값으로 통일 (음수가 있을경우 -1과 1 사이)
- MinMaxScaler 이용
- 데이터 분포가 가우시안 분포가 아닐 경우 적용

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

프로세스 정리

데이터 전처리

Null 값 처리, scaling => sklearn.preprocessing 모듈

train, test data 분리

=> sklearn.model_selection 모듈

모델 학습 (train data 이용)

데이터 특성별 모델 선택

교차 검증 => sklearn.model_selection 모듈

모델 성능 평가 (test data 이용)

수고하셨습니다