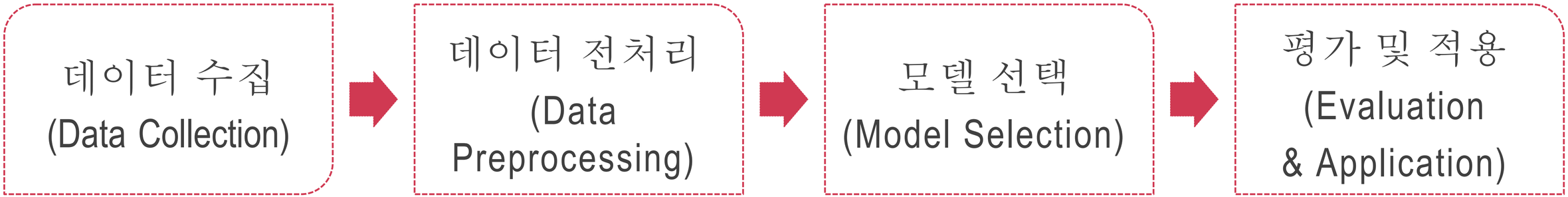
Machine Learning [2020.07.09 (목) ~ 2020.07.23 (목)]

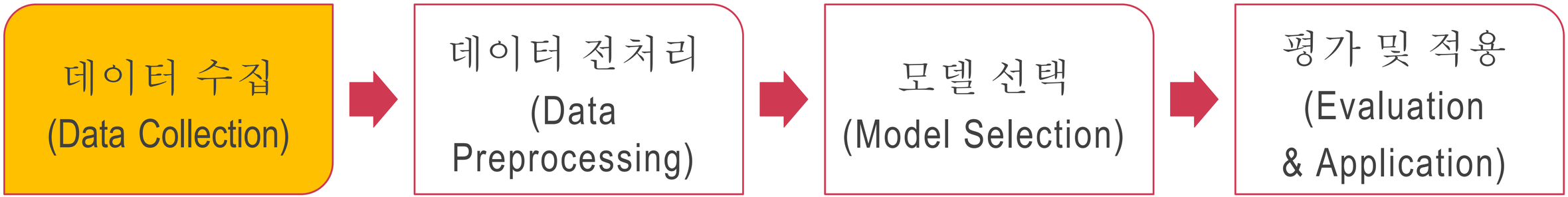
1	타이타닉 데이터로 머신러닝 데이터 분석 절차 소개
2	머신러닝 소개
3	머신러닝 지도학습_분류
4	머신러닝 지도학습_회귀
5	머신러닝 비지도학습_군집화
6	머신러닝 조별 실습 및 발표


TITANIC
사이킷런으로 수행하는 타이타닉 생존자 예측

빅데이터 분석 절차


데이터 수집
(Data Collection)
데이터 전처리
(Data
Preprocessing)
모델 선택
(Model Selection)
평가 및 적용
(Evaluation
& Application)

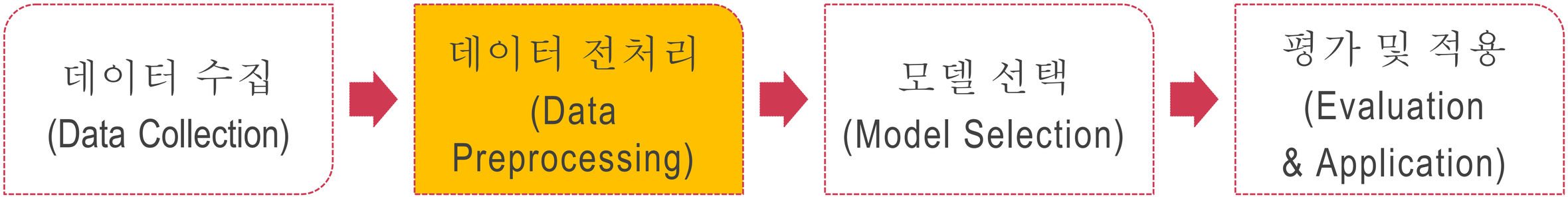
빅데이터 분석 절차


데이터 수집
(Data Collection)
데이터 전처리
(Data
Preprocessing)
모델 선택
(Model Selection)
평가 및 적용
(Evaluation
& Application)

타이타닉 탑승자 데이터 파일

https://www.kaggle.com/c/titanic/data

빅데이터 분석 절차


데이터 수집
(Data Collection)
데이터 전처리
(Data
Preprocessing)
모델 선택
(Model Selection)
평가 및 적용
(Evaluation
& Application)

데이터 확인

train - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말

```
PassengerId,Survived,Pclass,Name,Sex,Age,SibSp,Parch,Ticket,Fare,Cabin,Embarked
1,0,3,"Braund, Mr. Owen Harris",male,22,1,0,A/5 21171,7.25,,S
2,1,1,"Cumings, Mrs. John Bradley (Florence Briggs Thayer)",female,38,1,0,PC 17599,71.283
3,1,3,"Heikkinen, Miss. Laina",female,26,0,0,STON/O2. 3101282,7.925,,S
4,1,1,"Futrelle, Mrs. Jacques Heath (Lily May Peel)",female,35,1,0,113803,53.1,C123,S
5,0,3,"Allen, Mr. William Henry",male,35,0,0,373450,8.05,,S
6,0,3,"Moran, Mr. James",male,,0,0,330877,8.4583,,Q
7,0,1,"McCarthy, Mr. Timothy J",male,54,0,0,17463,51.8625,E46,S
8,0,3,"Palsson, Master. Gosta Leonard",male,2,3,1,349909,21.075,,S
9,1,3,"Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)",female,27,0,2,347742,11.1333,,S
```

DATA KUBWA

컬럼명 해석

변수명	정의
Passengerid	탑승자 데이터 일련번호
survived	생존 여부, 0=사망, 1=생존
pclass	티켓의 선실 등급, 1=일등석, 2=이등석, 3=삼등석
sex	탑승자 성별
name	탑승자 이름
Age	탑승자 나이
sibsp	같이 탑승한 형제자매 또는 배우자 인원수
parch	같이 탑승한 부모님 또는 어린이 인원수
ticket	티켓 번호
fare	요금
cabin	선실 번호
embarked	중간 정착 항구 C=Cherbourg, Q=Queenstown, S=Southampton

DATA KUBWA

학습 파일 불러오기

- import pandas as pd
- titanic_df = pd.read_csv('train.csv')
- titanic_df.head()

Column 명

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

Index

- titanic_df.shape
- (891, 12)

결손 데이터 확인

- titanic_df.isna().sum()

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

2 가지의 결측치 처리 방법

결손 데이터 채우기 비교

DATA KUBWA

#1번 방식

#2번 방식

결손 데이터 채우기 비교

DATA KUBWA

#1번 방식

#2번 방식

결손데이터 확인

컬럼명	전체 행 개수	데이터 수	결측치 수
Age	891	714	177
Cabin	891	204	687
Embarked	891	889	2

DATA KUBWA

결손데이터 확인

컬럼명	전체 행 개수	데이터 수	결측치 수
Age	891	714	177
Cabin	891	204	687
Embarked	891	889	2

DATA KUBWA

1ST 방식 – 탑승항 비율 확인 후 가장 많은 데이터 입력

DATA KUBWA

- df_train['Embarked'].value_counts()
- S 644
- C 168
- Q 77
- Name: Embarked, dtype: int64

- df_train['Embarked'] = df_train['Embarked'].fillna('S')
- df_train['Embarked'].isnull().sum()
- 0

결손데이터 확인

컬럼명	전체 행 개수	데이터 수	결측치 수
Age	891	714	177
Cabin	891	204	687

DATA KUBWA

1ST 방식

DATA KUBWA

- # 전체 평균 나이 확인
- df_train['Age'].mean()
- 29.69911764705882

- # 성별에 따른 나이 평균값
- # 남자 나이 평균
- df_train[df_train['Sex'] == 'male'].Age.mean()
- 30.72664459161148

- # 여자 나이 평균
- df_train[df_train['Sex'] == 'female']['Age'].mean()
- 27.915708812260537

- # Pclass 범주확인
- df_train['Pclass'].value_counts()
- 3 491
- 1 216
- 2 184

- # Pclass에 따른 평균 나이 확인
- df_train[df_train['Pclass'] == 1]['Age'].mean()
- 38.23
- df_train[df_train['Pclass']==2]['Age'].median()
- 29.88
- df_train[df_train['Pclass']==3]['Age'].median()
- 25.14

1ST 방식

```
# Age 결측치 채우기 – 성별과 선실 등급 각각의 평균 나이를 구하고 해당 조건에 따라 입력한다.
med_m_1 = df_train[(df_train['Sex']=='male')   & (df_train['Pclass']==1)]['Age'].mean()
med_m_2 = df_train[(df_train['Sex']=='male')   & (df_train['Pclass']==2)]['Age'].mean()
med_m_3 = df_train[(df_train['Sex']=='male')   & (df_train['Pclass']==3)]['Age'].mean()
med_f_1 = df_train[(df_train['Sex']=='female') & (df_train['Pclass']==1)]['Age'].mean()
med_f_2 = df_train[(df_train['Sex']=='female') & (df_train['Pclass']==2)]['Age'].mean()
med_f_3 = df_train[(df_train['Sex']=='female') & (df_train['Pclass']==3)]['Age'].mean()

df_train.loc[(df_train['Age'].isnull())&(df_train['Sex']=='male')  &(df_train['Pclass']==1),'Age'] = med_m_1
df_train.loc[(df_train['Age'].isnull())&(df_train['Sex']=='male')  &(df_train['Pclass']==2),'Age'] = med_m_2
df_train.loc[(df_train['Age'].isnull())&(df_train['Sex']=='male')  &(df_train['Pclass']==3),'Age'] = med_m_3
df_train.loc[(df_train['Age'].isnull())&(df_train['Sex']=='female')&(df_train['Pclass']==1),'Age'] = med_f_1
df_train.loc[(df_train['Age'].isnull())&(df_train['Sex']=='female')&(df_train['Pclass']==2),'Age'] = med_f_2
df_train.loc[(df_train['Age'].isnull())&(df_train['Sex']=='female')&(df_train['Pclass']==3),'Age'] = med_f_3

df_train['Age'].isnull().sum()
0
```

결손데이터 확인

컬럼명	전체 행 개수	데이터 수	결측치 수
Cabin	891	204	687

1ST 방식

DATA KUBWA

- # Cabin 데이터 확인
- df_train['Cabin'].describe()

```
count            204
unique           147
top        B96 B98
freq               4
Name: Cabin, dtype: object
```

- # Cabin 15개 샘플 확인
- df_train['Cabin'].sample(15)

```
691     NaN
635     NaN
206     NaN
162     NaN
247     NaN
302     NaN
794     NaN
274     NaN
406     NaN
193      F2
657     NaN
158     NaN
616     NaN
835     E49
870     NaN
Name: Cabin, dtype: object
```

1ST 방식

DATA KUBWA

- # Cabin 첫번째 알파벳만 남기고
- titanic_df['Cabin'] = titanic_df['Cabin'].str[:1]

- # Cabin 최종적으로 삭제
- df_train.drop(['Cabin'], axis=1, inplace=True)
- df_train.isnull().sum()

```
PassengerId    0
Survived       0
Pclass         0
Name           0
Sex            0
Age            0
SibSp          0
Parch          0
Ticket         0
Fare           0
Embarked       0
dtype: int64
```

결손 데이터 채우기 비교

DATA KUBWA

#1번 방식

#2번 방식

2ND 방식

DATA KUBWA

```
# 타이타닉 데이터를 새로운 데이터프레임명으로 지정
titanic_df = pd.read_csv('train.csv')

# 나이는 전체 평균값으로 입력
titanic_df['Age'].fillna(titanic_df['Age'].mean(), inplace=True)
# Cabin은 'N'으로 입력
titanic_df['Cabin'].fillna('N', inplace=True)
# 탑승항도 'N'으로 입력
titanic_df['Embarked'].fillna('N', inplace=True)
# 최종 결측값 확인
titanic_df.isnull().sum())
```

```
데이터 세트 Null 값 개수
 PassengerId    0
Survived        0
Pclass          0
Name            0
Sex             0
Age             0
SibSp           0
Parch           0
Ticket          0
Fare            0
Cabin           0
Embarked        0
Name_len        0
Child_Adult     0
Age_cat         0
dtype: int64
```

2ND 방식

DATA KUBWA

- # Cabin의 데이터 분포 확인
- titanic_df['Cabin'].value_counts()

```
 Cabin 값 분포:
 N                687
C23 C25 C27        4
G6                 4
B96 B98            4
D                  3
                ...
A31                1
A20                1
D46                1
B41                1
C103               1
Name: Cabin, Length: 148, dtype: int64
```

- # Cabin 데이터 수정. 첫번째 문자열만 남긴다
- titanic_df['Cabin'] = titanic_df['Cabin'].str[:1]

결측치 처리에 방식에 따른 머신러닝 모델 별 결과값 비교

DATA KUBWA

# 1 방식 결측치 처리	
Machine learning Model	Accuracy
Logistic Regression	0.7910
Support Vector Machine	0.7612
Decision Tree	0.7612
Random Forest	0.7799
하이퍼 파라미터 튜닝 (매개변수 최적화)	0.7935

#2 방식 결측치 처리	
Machine learning Model	Accuracy
Logistic Regression	0.7985
Support Vector Machine	0.7612
Decision Tree	0.7612
Random Forest	0.7799
하이퍼 파라미터 튜닝 (매개변수 최적화)	0.7912

탐색적 데이터 분석

DATA KUBWA

- # 컬럼별 상관관계 파악
- **df_train**.corr()

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
PassengerId	1.000000	-0.005007	-0.035144	0.039636	-0.057527	-0.001652	0.012658
Survived	-0.005007	1.000000	-0.338481	-0.067485	-0.035322	0.081629	0.257307
Pclass	-0.035144	-0.338481	1.000000	-0.407015	0.083081	0.018443	-0.549500
Age	0.039636	-0.067485	-0.407015	1.000000	-0.251313	-0.180705	0.118308
SibSp	-0.057527	-0.035322	0.083081	-0.251313	1.000000	0.414838	0.159651
Parch	-0.001652	0.081629	0.018443	-0.180705	0.414838	1.000000	0.216225
Fare	0.012658	0.257307	-0.549500	0.118308	0.159651	0.216225	1.000000

- **titanic_df**.corr()

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
PassengerId	1.000000	-0.005007	-0.035144	0.033207	-0.057527	-0.001652	0.012658
Survived	-0.005007	1.000000	-0.338481	-0.069809	-0.035322	0.081629	0.257307
Pclass	-0.035144	-0.338481	1.000000	-0.331339	0.083081	0.018443	-0.549500
Age	0.033207	-0.069809	-0.331339	1.000000	-0.232625	-0.179191	0.091566
SibSp	-0.057527	-0.035322	0.083081	-0.232625	1.000000	0.414838	0.159651
Parch	-0.001652	0.081629	0.018443	-0.179191	0.414838	1.000000	0.216225
Fare	0.012658	0.257307	-0.549500	0.091566	0.159651	0.216225	1.000000

탐색적 데이터 분석

DATA KUBWA

- df_train.head()

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked
0	1	0	3	Braund, Mr....	male	22.0	1	0	A/5 21171	7.2500	S
1	2	1	1	Cumings, Mr...	female	38.0	1	0	PC 17599	71.2833	C
2	3	1	3	Heikkinen, ...	female	26.0	0	0	STON/O2. 31...	7.9250	S
3	4	1	1	Futrelle, M...	female	35.0	1	0	113803	53.1000	S
4	5	0	3	Allen, Mr. ...	male	35.0	0	0	373450	8.0500	S

- titanic_df.head()

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr....	male	22.0	1	0	A/5 21171	7.2500	N	S
1	2	1	1	Cumings, Mr...	female	38.0	1	0	PC 17599	71.2833	C	C
2	3	1	3	Heikkinen, ...	female	26.0	0	0	STON/O2. 31...	7.9250	N	S
3	4	1	1	Futrelle, M...	female	35.0	1	0	113803	53.1000	C	S
4	5	0	3	Allen, Mr. ...	male	35.0	0	0	373450	8.0500	N	S

탐색적 데이터 분석

```
# 삭제: PassengerId, Name, Ticket
for i in [df_train, titanic_df]:
    i.drop(['PassengerId', 'Name', 'Ticket'], axis=1, inplace=True)
```

DATA ENCODING

Label Encoding

Food Name	Categorical #	Calories
Apple	1	95
Chicken	2	231
Broccoli	3	50

One Hot Encoding

Apple	Chicken	Broccoli	Calories
1	0	0	95
0	1	0	231
0	0	1	50

LABEL ENCODING

- from sklearn.preprocessing import LabelEncoder

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

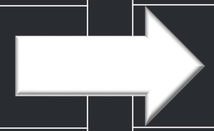
- # LabelEncoder를 객체로 생성한 후, fit()과 transform()으로 레이블 인코딩 수행
- encoder = LabelEncoder()
- encoder.fit(items)
- labels=encoder.transform(items)
- print('인코딩 변환값:', labels)
- 인코딩 변환값: [0 1 4 5 3 3 2 2]

- print('인코딩 클래스:', encoder.classes_)
- 인코딩 클래스: ['TV' '냉장고' '믹서' '선풍기' '전자레인지' '컴퓨터']

- print('디코딩 원본 값:', encoder.inverse_transform([4,5,2,0,1,1,3,3]))
- 디코딩 원본 값: ['전자레인지' '컴퓨터' '믹서' 'TV' '냉장고' '냉장고' '선풍기' '선풍기']

LABEL ENCODING

상품 분류	가격
TV	1,000,000
냉장고	1,500,000
전자렌지	200,000
컴퓨터	800,000
선풍기	100,000
선풍기	100,000
믹서	50,000
믹서	50,000



단점?

상품 분류	가격
0	1,000,000
1	1,500,000
4	200,000
5	800,000
3	100,000
3	100,000
2	50,000
2	50,000

ONE-HOT ENCODING

상품 분류
TV
냉장고
전자렌지
컴퓨터
선풍기
믹서

상품 분류 TV	상품 분류 냉장고	상품 분류 믹서	상품 분류 선풍기	상품 분류 전자렌지	상품 분류 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	1	0	0	0

ONE-HOT ENCODING

상품 분류	가격
TV	1,000,000
냉장고	1,500,000
전자렌지	200,000
컴퓨터	800,000
선풍기	100,000
선풍기	100,000
믹서	50,000
믹서	50,000

상품 분류	가격
0	1,000,000
1	1,500,000
4	200,000
5	800,000
3	100,000
3	100,000
2	50,000
2	50,000

상품 분류 TV	상품 분류 냉장고	상품 분류 믹서	상품 분류 선풍기	상품 분류 전자렌지	상품 분류 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	1	0	0	0

ONE-HOT ENCODING

- import pandas as pd

- df=pd.DataFrame({'item':['TV', '냉장고', '전자레인지', '컴퓨터', '선풍기', '선풍기', '믹서', '믹서']})
- pd.get_dummies(df)

	item_TV	item_냉장고	item_믹서	item_선풍기	item_전자레인지	item_컴퓨터
0	1	0	0	0	0	0
1	0	1	0	0	0	0
2	0	0	0	0	1	0
3	0	0	0	0	0	1
4	0	0	0	1	0	0
5	0	0	0	1	0	0
6	0	0	1	0	0	0
7	0	0	1	0	0	0

ONE-HOT ENCODING

- OHE연습 = df_train['Embarked']
- pd.get_dummies(OHE연습)

	C	Q	S
0	0	0	1
1	1	0	0
2	0	0	1
3	0	0	1
4	0	0	1
...	...	...	...
886	0	0	1
887	0	0	1
888	0	0	1
889	1	0	0
890	0	1	0

891 rows × 3 columns

타이타닉 LABEL ENCODING

DATA KUBWA

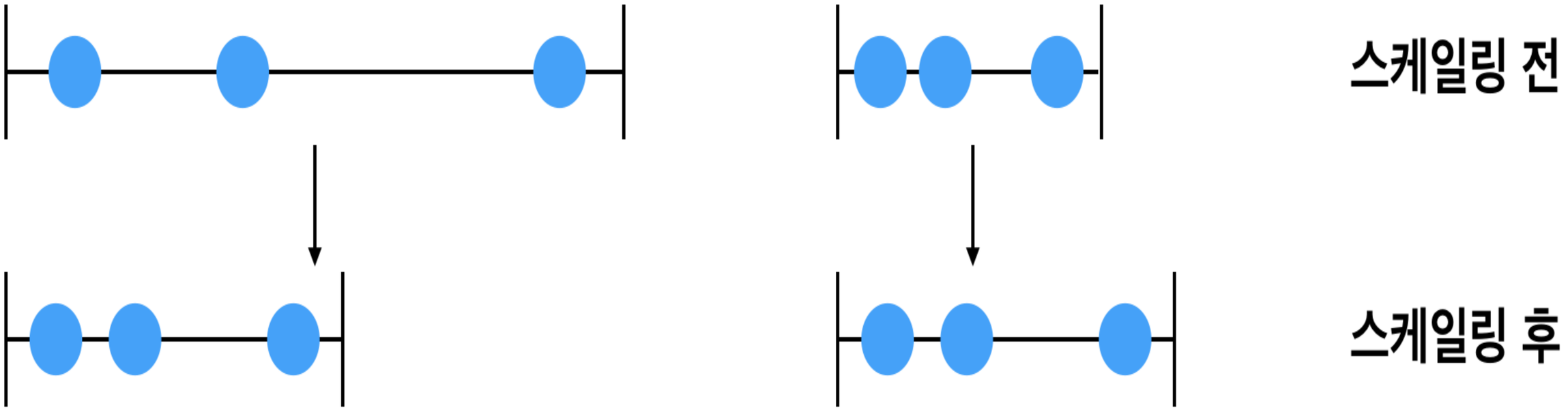
```
# 문자열 컬럼 사이킷런 레이블 인코딩
from sklearn.preprocessing import LabelEncoder
for i in ['Sex', 'Embarked']:
    encoder = LabelEncoder()
    encoder.fit(df_train[i])
    df_train[i]=encoder.transform(df_train[i])
df_train.head()
```

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	0	3	1	22.0	1	0	7.2500	2
1	1	1	0	38.0	1	0	71.2833	0
2	1	3	0	26.0	0	0	7.9250	2
3	1	1	0	35.0	1	0	53.1000	2
4	0	3	1	35.0	0	0	8.0500	2

```
for i in ['Sex', 'Embarked', 'Cabin']:
    encoder = LabelEncoder()
    encoder.fit(titanic_df[i])
    titanic_df[i]=encoder.transform(titanic_df[i])
titanic_df.head()
```

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Cabin	Embarked
0	0	3	1	22.0	1	0	7.2500	7	3
1	1	1	0	38.0	1	0	71.2833	2	0
2	1	3	0	26.0	0	0	7.9250	7	3
3	1	1	0	35.0	1	0	53.1000	2	3
4	0	3	1	35.0	0	0	8.0500	7	3

FEATURE SCALING



DATA
KUBWA

FEATURE SCALING

DATA KUBWA

표준화

$$X_{new} = \frac{X_i - X_{mean}}{\text{Standard Deviation}}$$

정규화

$$X_{new} = \frac{X_i - \min(X)}{\max(x) - \min(X)}$$

STANDARDSCALER

DATA KUBWA

- print('feature들의 평균값:\n', df_train.mean())
- print('\nfeature들의 분산값: \n', print(df_train.var())

```
feature들의 평균값
Survived     0.383838
Pclass       2.308642
Sex          0.647587
Age         29.112424
SibSp        0.523008
Parch        0.381594
Fare        32.204208
Embarked     1.536476
dtype: float64

feature들의 분산값
Survived       0.236772
Pclass         0.699015
Sex            0.228475
Age          177.007700
SibSp          1.216043
Parch          0.649728
Fare        2469.436846
Embarked       0.626477
dtype: float64
```

- print('feature들의 평균값:\n', titanic_df.mean())
- print('\nfeature들의 분산값:\n', titanic_df.var())

```
feature들의 평균값:
 Survived     0.383838
Pclass        2.308642
Sex           0.647587
Age          29.699118
SibSp         0.523008
Parch         0.381594
Fare         32.204208
Cabin         5.946128
Embarked      2.343434
dtype: float64

feature들의 분산값:
 Survived       0.236772
Pclass          0.699015
Sex             0.228475
Age           169.052400
SibSp           1.216043
Parch           0.649728
Fare         2469.436846
Cabin           4.253274
Embarked        1.362819
dtype: float64
```

STANDARDSCALER

DATA KUBWA

```
from sklearn.preprocessing import StandardScaler

# StandardScaler객체 생성
scaler = StandardScaler()

# StandardScaler로 데이터 세트 변환. fit()과 transform() 호출
scaler.fit(df_train)
First_scaled = scaler.transform(df_train)

# transform() 시 스케일 변환된 데이터 세트가 NumPy ndarray로 반환돼 이를 DataFrame로 변환
First_df_scaled = pd.DataFrame(data=First_scaled, columns=list(df_train.columns))
print('feature들의 평균값')
print(First_df_scaled.mean())
print('\nfeature들의 분산값')
print(First_df_scaled.var())
```

```
from sklearn.preprocessing import StandardScaler

# StandardScaler객체 생성
scaler = StandardScaler()

# StandardScaler로 데이터 세트 변환. fit()과 transform() 호출
scaler.fit(titanic_df)
Second_scaled = scaler.transform(titanic_df)

# transform() 시 스케일 변환된 데이터 세트가 NumPy ndarray로 반환돼 이를 DataFrame로 변환
Second_df_scaled = pd.DataFrame(data=Second_scaled, columns=list(titanic_df.columns))
print('feature들의 평균값')
print(Second_df_scaled.mean())
print('\nfeature들의 분산값')
print(Second_df_scaled.var())
```

STANDARDSCALER

```
feature들의 평균값
Survived   -2.287732e-16
Pclass     -2.031048e-16
Sex        -4.059603e-16
Age         1.804891e-16
SibSp       3.456519e-16
Parch       6.716164e-17
Fare       -4.373606e-17
Embarked    1.556306e-16
dtype: float64

feature들의 분산값
Survived    1.001124
Pclass      1.001124
Sex         1.001124
Age         1.001124
SibSp       1.001124
Parch       1.001124
Fare        1.001124
Embarked    1.001124
dtype: float64
```

```
feature들의 평균값
Survived   -2.287732e-16
Pclass     -2.031048e-16
Sex        -4.059603e-16
Age         2.562796e-16
SibSp       3.456519e-16
Parch       6.716164e-17
Fare       -4.373606e-17
Cabin      -2.683974e-16
Embarked    2.666529e-17
dtype: float64

feature들의 분산값
Survived    1.001124
Pclass      1.001124
Sex         1.001124
Age         1.001124
SibSp       1.001124
Parch       1.001124
Fare        1.001124
Cabin       1.001124
Embarked    1.001124
dtype: float64
```

한번 더 상관관계

DATA KUBWA

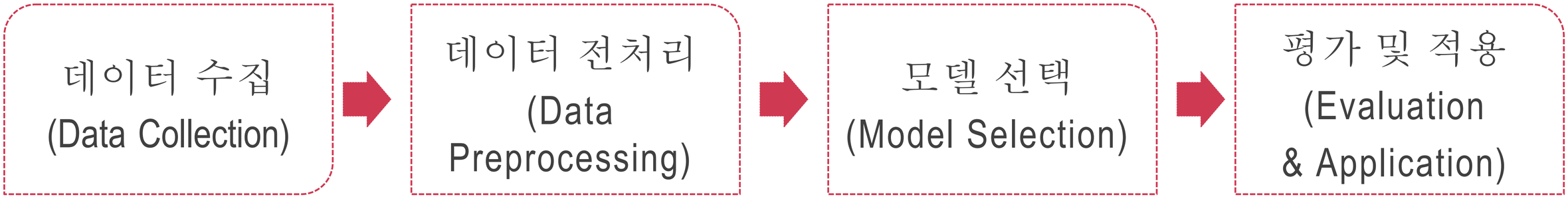
- df_train.corr()

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
Survived	1.000000	-0.338481	-0.543351	-0.067485	-0.035322	0.081629	0.257307	-0.167675
Pclass	-0.338481	1.000000	0.131900	-0.407015	0.083081	0.018443	-0.549500	0.162098
Sex	-0.543351	0.131900	1.000000	0.112851	-0.114631	-0.245489	-0.182333	0.108262
Age	-0.067485	-0.407015	0.112851	1.000000	-0.251313	-0.180705	0.118308	-0.011184
SibSp	-0.035322	0.083081	-0.114631	-0.251313	1.000000	0.414838	0.159651	0.068230
Parch	0.081629	0.018443	-0.245489	-0.180705	0.414838	1.000000	0.216225	0.039798
Fare	0.257307	-0.549500	-0.182333	0.118308	0.159651	0.216225	1.000000	-0.224719
Embarked	-0.167675	0.162098	0.108262	-0.011184	0.068230	0.039798	-0.224719	1.000000

- titanic_df.corr()

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Cabin	Embarked
Survived	1.000000	-0.338481	-0.543351	-0.069809	-0.035322	0.081629	0.257307	-0.295113	-0.174963
Pclass	-0.338481	1.000000	0.131900	-0.331339	0.083081	0.018443	-0.549500	0.742093	0.197493
Sex	-0.543351	0.131900	1.000000	0.084153	-0.114631	-0.245489	-0.182333	0.118635	0.106395
Age	-0.069809	-0.331339	0.084153	1.000000	-0.232625	-0.179191	0.091566	-0.249098	-0.034883
SibSp	-0.035322	0.083081	-0.114631	-0.232625	1.000000	0.414838	0.159651	0.041058	0.068043
Parch	0.081629	0.018443	-0.245489	-0.179191	0.414838	1.000000	0.216225	-0.031553	0.032517
Fare	0.257307	-0.549500	-0.182333	0.091566	0.159651	0.216225	1.000000	-0.525742	-0.246359
Cabin	-0.295113	0.742093	0.118635	-0.249098	0.041058	-0.031553	-0.525742	1.000000	0.227505
Embarked	-0.174963	0.197493	0.106395	-0.034883	0.068043	0.032517	-0.246359	0.227505	1.000000

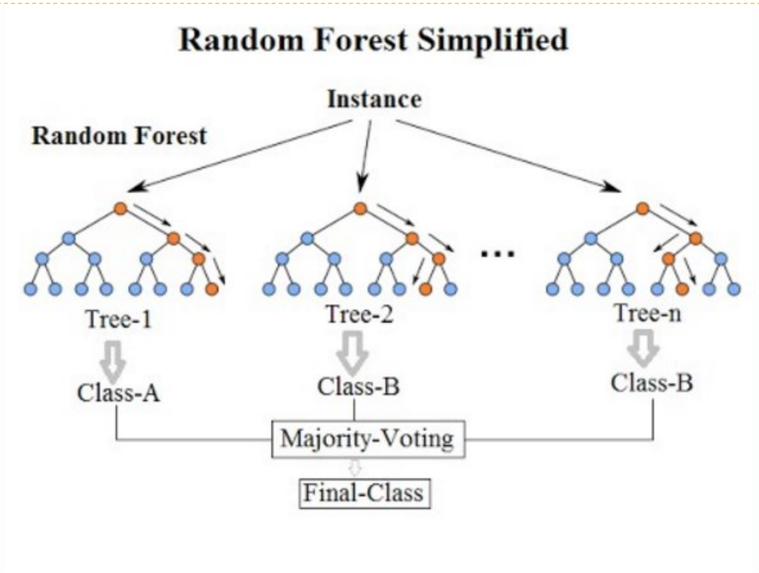
빅데이터 분석 절차

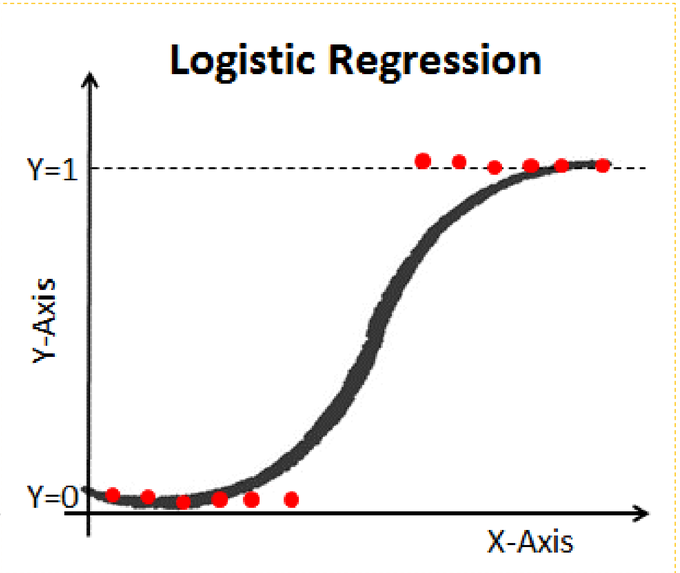

데이터 수집
(Data Collection)
데이터 전처리
(Data
Preprocessing)
모델 선택
(Model Selection)
평가 및 적용
(Evaluation
& Application)

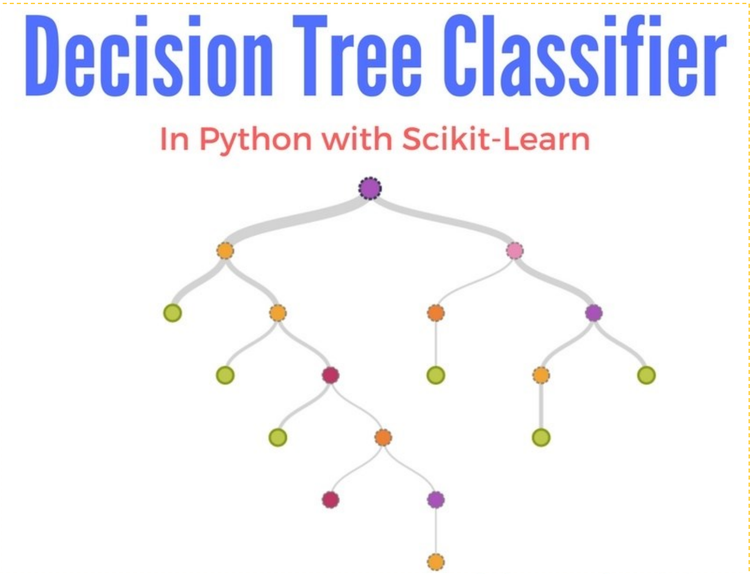
```
# 데이터 분리: 학습 데이터 + 테스트 데이터
y_titanic_df = df_train['Survived']
X_titanic_df = df_train.drop('Survived', axis=1)

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test =
train_test_split(X_titanic_df, y_titanic_df, test_size=0.2,
random_state=12)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(712, 7)
(179, 7)
(712,)
(179,)
```

MACHINE LEARNING (SCIKIT LEARN)







DATA KUBWA

머신러닝 모델 - 분류 (CLASSIFICATION) – SCIKIT-LEARN

- from sklearn.tree import DecisionTreeClassifier
- from sklearn.ensemble import RandomForestClassifier
- from sklearn.linear_model import LogisticRegression
- from sklearn.metrics import accuracy_score

- # 결정트리, Random forest, 로지스틱 회귀를 위한 사이킷런 Classifier 클래스 생성
- dt_clf = DecisionTreeClassifier(random_state=11)
- rf_clf = RandomForestClassifier(random_state=11)
- lr_clf = LogisticRegression()

머신러닝 모델 - 분류 (CLASSIFICATION) – SCIKIT-LEARN

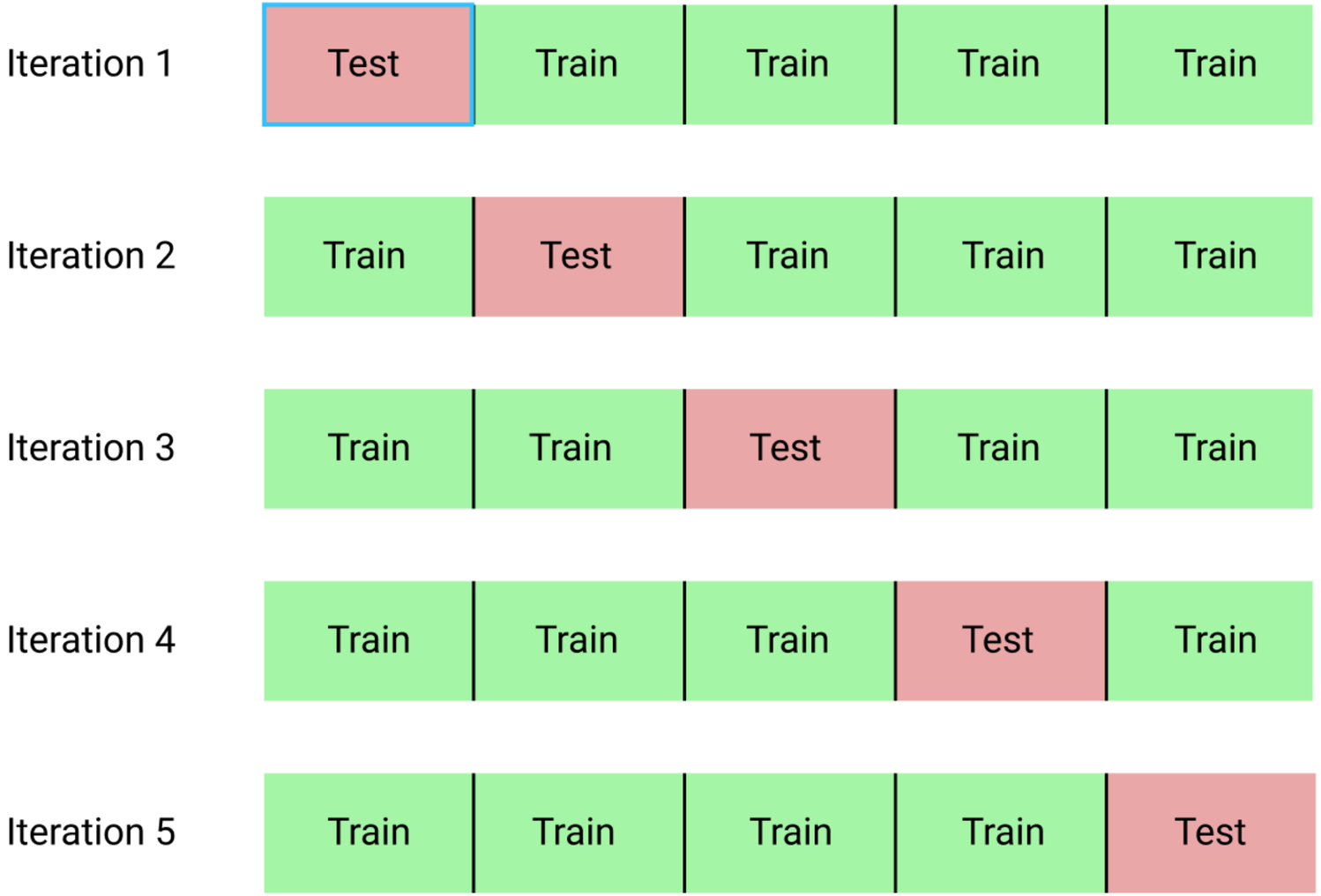
```
# DecisionTreeClassfier 학습/예측 평가
dt_clf.fit(X_train, y_train)
dt_pred = dt_clf.predict(X_test)
print('DecisionTreeClassifier 정확도: {0:.4f}'.format(accuracy_score(y_test, dt_pred)))

# RandomForestClassifier 학습/예측 평가
rf_clf.fit(X_train, y_train)
rf_pred = rf_clf.predict(X_test)
print('RandomForestClassifier 정확도:{0:.4f}'.format(accuracy_score(y_test, rf_pred)))

# LogisticRegression 학습/예측/평가
lr_clf.fit(X_train, y_train)
lr_pred = lr_clf.predict(X_test)
print('LogisticRegression 정확도: {0:.4f}'.format(accuracy_score(y_test, lr_pred)))
```

```
DecisionTreeClassifier 정확도: 0.7654
RandomForestClassifier 정확도:0.7430
LogisticRegression 정확도: 0.7654
```

Feature scaling은 분류 모델에서 사용하지 않는다.

DATA KUBWA

교차 검증 KFOLD


Iteration 1
Test
Train
Train
Train
Train
Iteration 2
Train
Test
Train
Train
Train
Iteration 3
Train
Train
Test
Train
Train
Iteration 4
Train
Train
Train
Test
Train
Iteration 5
Train
Train
Train
Train
Test

교차 검증 KFOLD

```
from sklearn.model_selection import KFold

def exec_kfold(clf, folds=5):
    # 폴드 세트를 5개인 KFold객체를 생성, 폴드 수만큼 예측결과 저장을 위한 리스트 객체 생성.
    kfold = KFold(n_splits=folds)
    scores = []
    # KFold 교차 검증 수행.
    for iter_count, (train_index, test_index) in enumerate(kfold.split(X_titanic_df)):
        # X_titanic_df 데이터에서 교차 검증별로 학습과 검증 데이터를 가리키는 index 생성
        X_train, X_test = X_titanic_df.values[train_index], X_titanic_df.values[test_index]
        y_train, y_test = y_titanic_df.values[train_index], y_titanic_df.values[test_index]
        # Classifier 학습, 예측, 정확도 계산
        clf.fit(X_train, y_train)
        predictions = clf.predict(X_test)
        accuracy = accuracy_score(y_test, predictions)
        scores.append(accuracy)
        print("교차 검증 {0} 정확도: {1:.4f}".format(iter_count, accuracy))

    # 5개 fold에서의 평균 정확도 계산.
    mean_score = np.mean(scores)
    print("평균 정확도: {0:.4f}".format(mean_score))
```

교차 검증 KFOLD

DATA KUBWA

- # exec_kfold 호출
- exec_kfold(dt_clf, folds=5)

```
교차 검증  0 정확도:  0.7542
교차 검증  1 정확도:  0.7809
교차 검증  2 정확도:  0.7865
교차 검증  3 정확도:  0.7697
교차 검증  4 정확도:  0.8202
평균 정확도:  0.7823
```

- # exec_kfold 호출
- exec_kfold(rf_clf, folds=5)

```
교차 검증  0 정확도:  0.7989
교차 검증  1 정확도:  0.7640
교차 검증  2 정확도:  0.8202
교차 검증  3 정확도:  0.7921
교차 검증  4 정확도:  0.8315
평균 정확도:  0.8013
```

- # exec_kfold 호출
- exec_kfold(lr_clf, folds=5)

```
교차 검증  0 정확도:  0.7933
교차 검증  1 정확도:  0.7921
교차 검증  2 정확도:  0.7753
교차 검증  3 정확도:  0.7472
교차 검증  4 정확도:  0.8427
평균 정확도:  0.7901
```

교차 검증 STRATIFIEDKFOLD

DATA KUBWA

만약 1의 분포가 5% 내외라서
일부 검증 레이블 데이터 분포가 극단적이라면?

예) 검증 레이블 데이터 분포
0 200
1 0

```
import pandas as pd
df_train['Survived'].value_counts()
0    549
1    342
Name: Survived, dtype: int64

kfold = KFold(n_splits=5)
num = 0
for train_index, test_index in kfold.split(df_train):
    num+=1
    label_train = df_train['Survived'].iloc[train_index]
    label_test = df_train['Survived'].iloc[test_index]
    print('## 교차 검증: {0}'.format(num))
    print('학습 레이블 데이터 분포:\n', label_train.value_counts())
    print('검증 레이블 데이터 분포:\n', label_test.value_counts())
```

```
## 교차 검증: 2
학습 레이블 데이터 분포:
 0    450
1    263
Name: Survived, dtype: int64
검증 레이블 데이터 분포:
 0    99
1    79
Name: Survived, dtype: int64
## 교차 검증: 3
학습 레이블 데이터 분포:
 0    440
1    273
Name: Survived, dtype: int64
검증 레이블 데이터 분포:
 0    106
1    72
Name: Survived, dtype: int64
## 교차 검증: 5
학습 레이블 데이터 분포:
 0    434
1    279
Name: Survived, dtype: int64
검증 레이블 데이터 분포:
 0    115
1    63
Name: Survived, dtype: int64
```

교차 검증 STRATIFIEDKFOLD

DATA KUBWA

```
from sklearn.model_selection import StratifiedKFold

skf = StratifiedKFold(n_splits=5)
num = 0

for train_index, test_index in skf.split(df_train, df_train['Survived']):
    num += 1
    label_train = df_train['Survived'].iloc[train_index]
    label_test = df_train['Survived'].iloc[test_index]
    print('## 교차 검증: {0}'.format(num))
    print('학습 레이블 데이터 분포:\n', label_train.value_counts())
    print('검증 레이블 데이터 분포:\n', label_test.value_counts())
```

```
## 교차 검증: 1
학습 레이블 데이터 분포:
 0    439
1    273
Name: Survived, dtype: int64
검증 레이블 데이터 분포:
 0    110
1     69
Name: Survived, dtype: int64
## 교차 검증: 2
학습 레이블 데이터 분포:
 0    439
1    273
Name: Survived, dtype: int64
검증 레이블 데이터 분포:
 0    110
1     69
Name: Survived, dtype: int64
검증 레이블 데이터 분포:
 0    110
1     68
Name: Survived, dtype: int64
## 교차 검증: 4
학습 레이블 데이터 분포:
 0    439
1    274
Name: Survived, dtype: int64
검증 레이블 데이터 분포:
 0    110
1     68
Name: Survived, dtype: int64
## 교차 검증: 5
학습 레이블 데이터 분포:
 0    440
1    274
Name: Survived, dtype: int64
검증 레이블 데이터 분포:
 0    109
1     68
Name: Survived, dtype: int64
```

교차 검증 STRATIFIEDKFOLD

```
skfold = StratifiedKFold(n_splits=5)
n_iter = 0
cv_accuracy = []

# StratifiedKFold의 split() 호출 시 반드시 레이블 데이터 세트도 추가 입력 필요
for train_index, test_index, in skfold.split(X_titanic_df, y_titanic_df):
    # split()으로 반환된 인덱스를 이용해 학습용, 검증용 테스트 데이터 추출
    X_train, X_test = X_titanic_df.values[train_index], X_titanic_df.values[test_index]
    y_train, y_test = y_titanic_df.values[train_index], y_titanic_df.values[test_index]
    # 학습 및 예측
    dt_clf.fit(X_train, y_train)
    pred = dt_clf.predict(X_test)

    # 반복 시마다 정확도 측정
    n_iter += 1
    accuracy = np.round(accuracy_score(y_test, pred), 4)
    train_size = X_train.shape[0]
    test_size = X_test.shape[0]
    print('\n#{0} 교차 검증 정확도 :{1}, 학습 데이터 크기: {2}, 검증 데이터 크기: {3}'
          .format(n_iter, accuracy, train_size, test_size))
    cv_accuracy.append(accuracy)

# 교차 검증별 정확도 및 평균 정확도 계산
print('\n## 교차 검증별 정확도:', np.round(cv_accuracy, 4))
print('## 평균 검증 정확도:', np.mean(cv_accuracy))
```

```
#1 교차 검증 정확도 :0.7374, 학습 데이터 크기: 712, 검증 데이터 크기: 179

#2 교차 검증 정확도 :0.7765, 학습 데이터 크기: 712, 검증 데이터 크기: 179

#3 교차 검증 정확도 :0.8146, 학습 데이터 크기: 713, 검증 데이터 크기: 178

#4 교차 검증 정확도 :0.7697, 학습 데이터 크기: 713, 검증 데이터 크기: 178

#5 교차 검증 정확도 :0.7966, 학습 데이터 크기: 714, 검증 데이터 크기: 177

## 교차 검증별 정확도: [0.7374 0.7765 0.8146 0.7697 0.7966]
## 평균 검증 정확도: 0.77896
```

교차 검증 CROSS_VAL_SCORE

DATA KUBWA

```
from sklearn.model_selection import cross_val_score


for i in [dt_clf, rf_clf, lr_clf]:
    scores = cross_val_score(i, X_titanic_df, y_titanic_df, cv=5)
    for iter_count, accuracy in enumerate(scores):
        print("교차 검증 {0} 정확도: {1:.4f}".format(iter_count, accuracy))
    print("평균 정확도: {0:.4f}".format(np.mean(scores)))
```

```
교차 검증 0 정확도: 0.7374
교차 검증 1 정확도: 0.7765
교차 검증 2 정확도: 0.8146
교차 검증 3 정확도: 0.7697
교차 검증 4 정확도: 0.7966
평균 정확도: 0.7790
교차 검증 0 정확도: 0.7598
교차 검증 1 정확도: 0.7598
교차 검증 2 정확도: 0.8427
교차 검증 3 정확도: 0.7865
교차 검증 4 정확도: 0.8249
평균 정확도: 0.7947
교차 검증 0 정확도: 0.7989
교차 검증 1 정확도: 0.8156
교차 검증 2 정확도: 0.7809
교차 검증 3 정확도: 0.7640
교차 검증 4 정확도: 0.8192
평균 정확도: 0.7957
```

교차 검증 (1ST)

머신러닝 모델	dt_clf			rf_clf			lr_clf		
교차 검증 모델	KFold	S_KF	c_v_s	Kfold	S_KF	c_v_s	KFold	S_KF	c_v_s
교차 검증 0 정확도:	0.7821	0.7374	0.7374	0.7709	0.7598	0.7598	0.7933	0.7989	0.7989
교차 검증 1 정확도:	0.7753	0.7765	0.7765	0.7978	0.7598	0.7598	0.8090	0.8156	0.8156
교차 검증 2 정확도:	0.8034	0.8146	0.8146	0.8539	0.8427	0.8427	0.7753	0.7809	0.7809
교차 검증 3 정확도:	0.7528	0.7697	0.7697	0.7865	0.7865	0.7865	0.7472	0.7640	0.7640
교차 검증 4 정확도:	0.7921	0.7966	0.7966	0.8202	0.8249	0.8249	0.8371	0.8192	0.8192
평균 정확도:	0.7811	0.7790	0.7790	0.8059	0.7947	0.7947	0.7924	0.7957	0.7879
머신러닝 정확도	0.7654			0.7430			0.7654		

DATA KUBWA

최적 하이퍼 파라미터

```
from sklearn.model_selection import GridSearchCV

parameters1 = {'max_depth':[1,2,3,4,5,6,7,8,9,10],
               'min_samples_split':[2,3,4,5,6,7,8,9,10], 'min_samples_leaf':[1,2,3,4,5,6,7,8,9,10]}

grid_dclf = GridSearchCV(dt_clf, param_grid=parameters, scoring='accuracy', cv=5)
grid_dclf.fit(X_train, y_train)

print('GridSearchCV 최적 하이퍼 파라미터 :', grid_dclf.best_params_)
print('GridSearchCV 최고 정확도: {0:.4f}'.format(grid_dclf.best_score_))
best_dclf = grid_dclf. best_estimator_

# GridSEarchCV의 최적 하이퍼 파라미터로 학습된 Estimator로 예측 및 평가 수행
dpredictions = best_dclf.predict(X_test)
accuracy = accuracy_score(y_test, dpredictions)
print('테스트 세트에서의 DecisionTreeClassifier 정확도 : {0:.4f}'.format(accuracy))
```

```
GridSearchCV 최적 하이퍼 파라미터 : {'max_depth': 6, 'min_samples_leaf': 1, 'min_samples_split': 8}
GridSearchCV 최고 정확도: 0.8118
테스트 세트에서의 DecisionTreeClassifier 정확도 : 0.8492
```

최적 하이퍼 파라미터

	Decision Tree	Random Forest	Logistic Regression
GridSearchCV 최고 정확도	0.8137	0.8375	0.8067
테스트 세트에서의 Classifier 정확도	0.8475	0.8305	0.8475

- dt_clf.get_params().keys()
- dict_keys(['class_weight', 'criterion', 'max_depth', 'max_features', 'max_leaf_nodes', 'min_impurity_decrease', 'min_impurity_split', 'min_samples_leaf', 'min_samples_split', 'min_weight_fraction_leaf', 'presort', 'random_state', 'splitter'])
- rf_clf.get_params().keys()
- dict_keys(['bootstrap', 'class_weight', 'criterion', 'max_depth', 'max_features', 'max_leaf_nodes', 'min_impurity_decrease', 'min_impurity_split', 'min_samples_leaf', 'min_samples_split', 'min_weight_fraction_leaf', 'n_estimators', 'n_jobs', 'oob_score', 'random_state', 'verbose', 'warm_start'])
- lr_clf.get_params().keys()
- dict_keys(['C', 'class_weight', 'dual', 'fit_intercept', 'intercept_scaling', 'l1_ratio', 'max_iter', 'multi_class', 'n_jobs', 'penalty', 'random_state', 'solver', 'tol', 'verbose', 'warm_start'])