원자력 스터디 3조

Logloss

모델의 출력값과 정답의 오차를 정의하는 함수

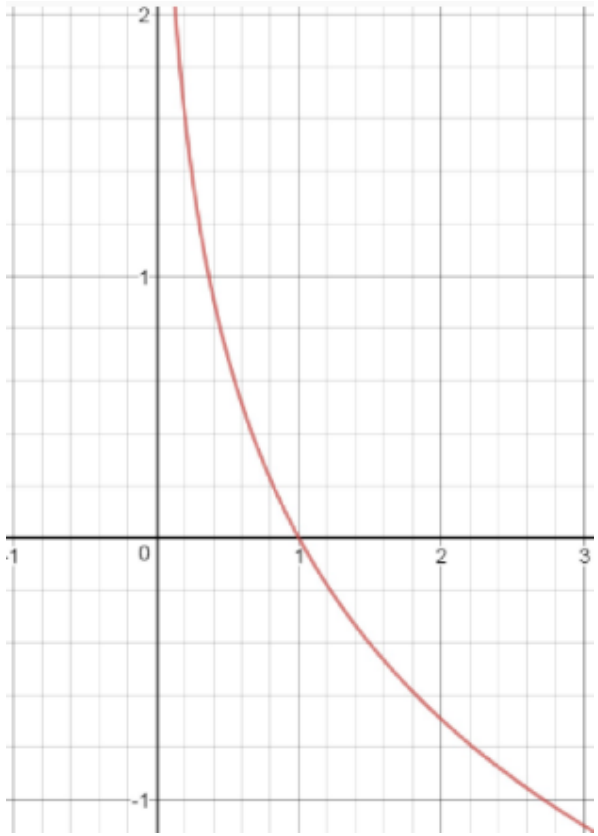
모델이 예측한 확률값을 반영

확률이 낮아질수록 logloss 값이 기하급수적으로 증가

-> 성능이 낮으면 더욱 큰 페널티를 부과

모델 전체의 로그로스

-> 제 답안에 해당하는 확률 값을 음의 로그를 취해 더하고 1/n 평균내기



Logloss

$$-\log P(yt|yp) = -(yt \log(yp) + (1 - yt) \log(1 - yp))$$

데이터 로드

```
import pandas as pd
import numpy as np

def data_loader_v2(file_name, folder='', train_label=None, event_time=10, nrows=60):
    file_id = int(file_name.split('.')[0]) # file id만 불러오기
    df = pd.read_csv(folder+file_name, index_col=0, nrows=nrows) # 파일 읽어오기
    df = df.replace('.*', 0, regex=True).fillna(0) # 모든 문자열과 NA값을 0으로 대체
    df = df.loc[event_time:] # event_time 이후의 row들만 가지고 오기
    df.index = np.repeat(file_id, len(df)) # row 인덱스를 file id로 덮어 씌우기
    if type(train_label) != type(None):
        label = train_label.loc[file_id]['label']
        df['label'] = np.repeat(label, len(df)) #train set일 경우 라벨 추가하기
    return df
```

```
def data_loader_all_v2(func, files, folder='', train_label=None, event_time=10, nrows=60):
    func_fixed = partial(func, folder=folder, train_label=train_label, event_time=event_time, nrows=nrows)
    if __name__ == '__main__':
        pool = Pool(processes=multiprocessing.cpu_count())
        df_list = list(pool.imap(func_fixed, files))
        pool.close()
        pool.join()
    combined_df = pd.concat(df_list)
    return combined_df
```

데이콘 baseline

데이터 전처리

데이터 전처리의 필요성

데이터에 Null값 등 결손값은 허용되지 않는다!

➔ Null값을 고정된 다른 값으로 변환해야 함

어떻게 결손값을 처리할 것인가?

1) 결손값이 단순 결손인 경우(결손값이 적은 경우!) : 피처의 평균값, 최빈값 등으로 간단히 대체

2) 결손값이 많은 경우 : 해당 피처는 드롭하는 것이 나을 수도 있음

만약, 피처의 중요도가 높다면 Null을 단순히 피처의 평균값으로 대체할 경우 예측 왜곡이 심할 수 있음

➔ **정밀한 대체 값을** 선정해야 함!

사이킷런의 머신러닝 알고리즘은 문자열 값을 입력 값으로 허용하지 않음

➔ 따라서 모든 문자열 값은 **인코딩 해서 숫자 형으로 변환**하기!

데이터 전처리

모사데이터에서 컬럼의 모든 값이 동일하고 변하지 않는 컬럼 찾기 => 799개 DROP

In [ ]:

```
for i in tqdm_notebook(range(1548)):
    if i in [30, 828, 1548]:
        continue

    if i < 828:
        data = pd.read_csv(train_folder + str(i) + '.csv')
    else:
        data = pd.read_csv(test_folder + str(i) + '.csv')

    if i == 0:
        result_set = set(get_uni_dict(data).items())
    else:
        tmp_set = set(get_uni_dict(data).items())
        result_set = result_set.intersection(tmp_set)

cols_with_no_valid_info = [i[0] for i in result_set]
print(len(cols_with_no_valid_info))
print(cols_with_no_valid_info)
```

우승자 코드 5등팀 자료 중

STRING, NAN 값을 포함한 컬럼 DROP

```
train_nan = []
train_str = []
for i in tqdm_notebook(range(828)):
    data = pd.read_csv(train_folder + str(i) + '.csv')

    for col in col_index:
        col_data = data[col]
        if col_data.isna().any():
            train_nan.append(col)
        if pd.Series([type(i)==str for i in list(col_data.values)]).any():
            train_str.append(col)

test_nan = []
test_str = []
for i in tqdm_notebook(range(828, 1548)):
    data = pd.read_csv(test_folder + str(i) + '.csv')

    for col in col_index:
        col_data = data[col]
        if col_data.isna().any():
            test_nan.append(col)
        if pd.Series([type(i)==str for i in list(col_data.values)]).any():
            test_str.append(col)
```

우승자 코드 5등팀 자료 중

데이터 전처리

결론

데이터 결손값을 1)평균 대체, 2)최빈값 대체 3) 제거 로 성능 비교

불필요한 데이터 제거

데이터 인코딩

데이터 인코딩 방식 : 레이블 인코딩과 원-핫인코딩

레이블 인코딩

카테고리 피처를 코드형 숫자 값으로 변환하는 것

즉, 간단하게 문자열 값을 숫자형 카테고리 값으로 변환

사이킷런의 LabelEncoder 클래스로 구현

숫자 값의 크기에 대한 특성 작용 ->
선형회귀알고리즘에는 적용하지 말 것

```
#2020-03-24
#jih020202@gmail.com
from sklearn.preprocessing import LabelEncoder
```

```
items=['월요일','일요일', '금요일','금요일','목요일', '화요일', '수요일', '토요일' ]
```

```
#LabelEncoder 객체 생성 후 레이블 인코딩

encoder = LabelEncoder()
encoder.fit(items)
labels = encoder.transform(items)
print('인코딩 : ',labels )
```

```
인코딩 :  [3 4 0 0 1 6 2 5]
```

```
#어떻게 인코딩 되었는지 속성값 확인 가능

print('인코딩 클래스 : ', encoder.classes_)
```

```
인코딩 클래스 :  ['금요일' '목요일' '수요일' '월요일' '일요일' '토요일' '화요일']
```

```
#디코딩

print('디코딩 : ', encoder.inverse_transform([0,1,2,3,4,5,6]))
```

```
디코딩 :  ['금요일' '목요일' '수요일' '월요일' '일요일' '토요일' '화요일']
```

데이터 인코딩

데이터 인코딩 방식 : 레이블 인코딩과 원-핫인코딩

원-핫 인코딩

피처 값의 유형에 따라 새로운 피처를 추가해 고유 값에 해당하는 칼럼에만 1을 표시하고 나머지 칼럼에는 0을 표시하는 방식

사이킷런의 OneHotEncoder 클래스로 구현

➔ 이때 변환 전 모든 문자열 값이 숫자형 값으로 변환되어야 함 + 입력 값으로 2차원 데이터 필요

```
#2020-03-24
#jih0202020@gmail.com
from sklearn.preprocessing import OneHotEncoder
import numpy as np
```

```
items=['월요일','일요일', '금요일','금요일','목요일', '화요일', '수요일', '토요일' ]
```

```
#숫자 형태로 변환
encoder = LabelEncoder()
encoder.fit(items)
labels = encoder.transform(items)

#2차원으로 변환
labels = labels.reshape(-1,1)
```

```
#원핫인코딩 적용
encoder = OneHotEncoder()
encoder.fit(labels)
oh_labels=encoder.transform(labels)
print('인코딩 데이터 : ₩n', oh_labels.toarray())
print('데이터 차원 :₩n', oh_labels.shape)
```

```
인코딩 데이터 :
 [[0. 0. 0. 1. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 1. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1. 0.]]
데이터 차원 :
 (8, 7)
```

데이터 인코딩

원-핫 인코딩

판다스의 get_dummies()로 구현 가능

➔ 문자열 카테고리 값을 숫자 형으로 변환할 필요 없음

In [27]:
```
#2020-03-24
#jih0202020@gmail.com
import pandas as pd
```

In [28]:
```
df = pd.DataFrame({'item' : ['월요일','일요일', '금요일','금요일','목요일', '화요일',
```

In [29]:
```
pd.get_dummies(df)
```

Out[29]:

	item_금요일	item_목요일	item_수요일	item_월요일	item_일요일	item_토요일	item_화요일
0	0	0	0	1	0	0	0
1	0	0	0	0	1	0	0
2	1	0	0	0	0	0	0
3	1	0	0	0	0	0	0
4	0	1	0	0	0	0	0
5	0	0	0	0	0	0	1
6	0	0	1	0	0	0	0
7	0	0	0	0	0	1	0

피처 스케일링

피처 스케일링

서로 다른 변수의 값 범위를 일정한 수준으로 맞추는 작업

표준화와 정규화

표준화

데이터의 피처 각각이 평균이 0이고 분산이 1인 가우시안 정규 분포를 가진 값으로 변환하는 것

$$X_{new} = \frac{X - \mu}{\sigma} = \frac{X - \text{Mean}(X)}{\text{StdDev}(X)}$$

표준화를 통해 변환될 피처 x의 새로운 i번째 데이터를 x_new라고 하면

➔ 원래 값에서 피처 x의 평균을 뺀 값을 피처 x의 표준편차로 나눈 값

정규화

서로 다른 피터의 크기를 통일하기 위해 크기를 변환해주는 개념

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

개별 데이터의 크기를 모두 똑같은 단위로 변경

새로운 데이터 x_new는

➔ 원래 값에서 피처 x의 최솟값을 뺀 값을 피처 x의 최댓값과 최솟값의 차이로 나눈 값

피처 스케일링

StandardScaler

사이킷런에서 제공하는 대표적인 피처 스케일링 클래스1

표준화를 쉽게 지원하기 위한 클래스

데이터가 가우시안 분포를 가지고 있다고 가정하고 구현된 선형회귀, 로지스틱 회귀, 소프트 벡터 머신 등 적용 가능

->모든 칼럼 값의 평균이 0에 가까운 값으로, 분산은 1에 가까운 값으로 변환

MinMax

사이킷런에서 제공하는 대표적인 피처 스케일링 클래스2

데이터값을 0과 1사이의 범위 값으로 변환 (음수값의 경우 -1~1)

데이터 분포가 가우시안 분포가 아닐 경우 적용 가능

정리

필요시 MaxMinScaler 적용이 적절

앙상블

앙상블학습

여러 개의 분류기를 결합함으로써 보다 정확한 최종 예측을 도출하는 기법

다양한 분류기의 예측 결과를 결합 -> 단일 분류기보다 신뢰성이 높은 예측값을 얻음

```
# MODEL LOAD & TEST PREDICT
# 12 MODELS 평균 사용
models = os.listdir('../2_Code_pred/')
models_list = [x for x in models if x.endswith(".pkl")]
assert len(models_list) ==12
temp_predictions = np.zeros((test.shape[0],198))

for m in models_list:
    model = joblib.load('../2_Code_pred/'+m)
    predict_proba = model.predict_proba(test)
    temp_predictions += predict_proba/12
```

우승자 코드 1등팀 자료 중

앙상블

보팅

여러 개의 분류기가 투표를 통해 최종 예측 결과를 결정하는 방식

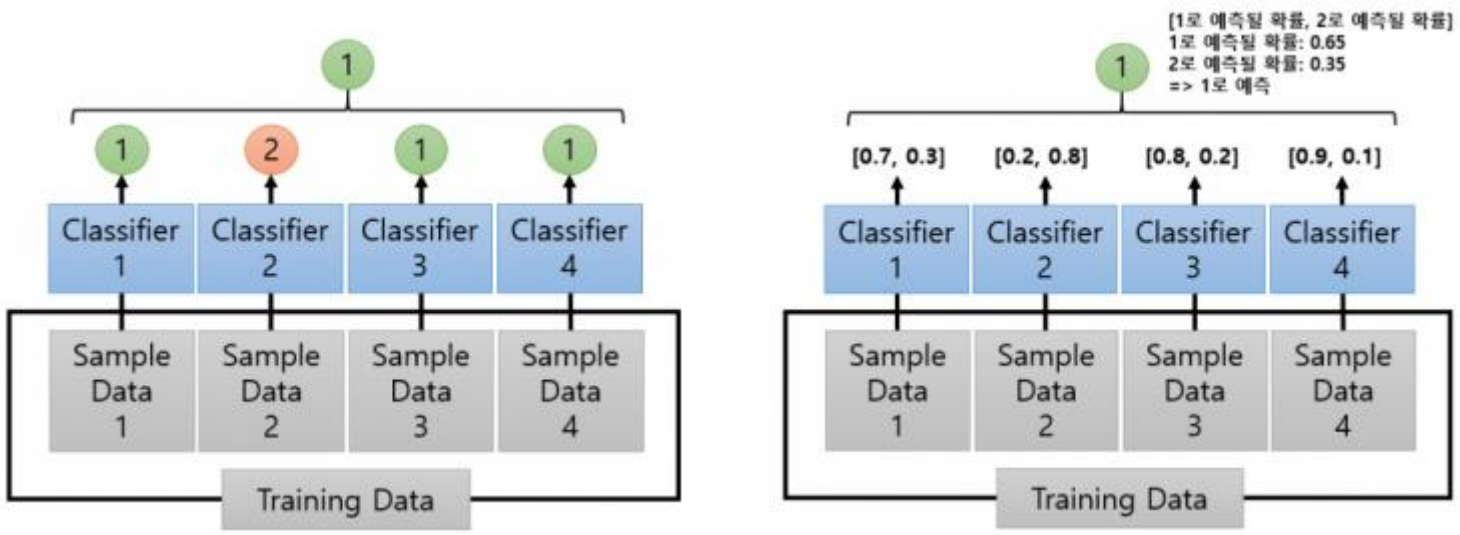
서로 다른 알고리즘을 가진 분류기를 결합

하드보팅 : 예측 결과값 중 다수의 분류기가 결정한 예측값을 최종으로 선정

소프트 보팅 : 분류기들의 레이블 값 결정 확률을 모두 더하고 평균내어 확률이 가장 높은 레이블 값을 최종 선정

➔ 일반적으로 소프트 보팅이 보팅 방법으로 적용

cf. 보팅 분류기 - 사이킷런의 보팅앙상블 VotingClassifier클래스



앙상블

배깅

여러 개의 분류기가 투표를 통해 최종 예측 결과를 결정하는 방식

서로 같은 유형의 알고리즘 기반으로 하되 데이터 샘플이 각기 다른 분류기 사용

데이터 세트 간 중첩을 허용

부스팅

여러 개의 분류기가 순차적으로 학습을 수행하되 예측이 틀린
데이터에 대해서는 올바르게 예측할 수 있도록 가중치를 부여

➔ XGBoost, LGBM, ..

스태킹

스태킹은 여러가지 다른 모델의 예측 결과값을 다시 학습 데이터로 만들어서 다른 모델로 재학습시켜 결과를 예측

성능 수치를 조금이라도 높여야 할 경우 다시 사용

-〉 많은 개별 모델 필요

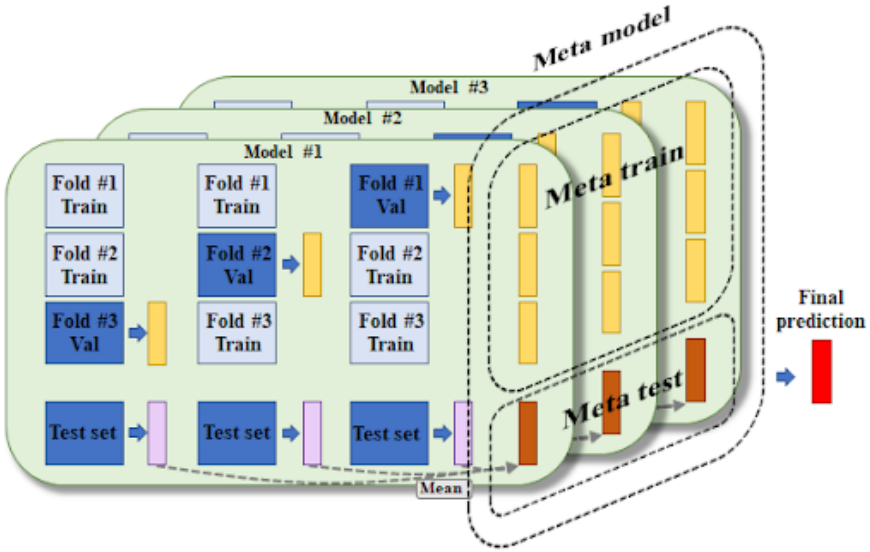
앙상블학습

스태킹 앙상블

개별 모델이 예측한 데이터를 다시 training set 으로 사용하여 학습

메타모델 : 개별 모델의 예측된 데이터 세트를 다시 기반으로 학습하고 예측하는 방식

CV기반의 스태킹 앙상블



정리

Log loss 방식의 측정지표를 고려 -〉 소프트 보팅 적용

이때 사용하는 분류기로 XGBoost, LGBM 등의 모델을 사용하여 부스팅 적용

최종 단계에서 성능 수치를 올리고자 한다면 스태킹앙상블 적용

교차검증

교차검증의 필요성

일반적으로 알고리즘을 학습시키는 학습 데이터와 예측 성능을 평가하기 위한 별도의 테스트용 데이터 필요

만약 별도의 테스트용 데이터 (즉, 학습용 데이터로만 테스트를 진행하는 경우)가 없다면 정확도가 100%에 이르는 모델 탄생..!

하지만 단순히 테스트용 데이터를 생성한다면 **과적합**에 취약하게 됨

과적합(Overfitting) : 모델이 학습 데이터에만 과도하게 최적화되어 실제 예측을 다른 데이터로 수행할 경우 예측 성능이 과도하게 떨어지는 것

즉, 고정된 학습 데이터와 테스트 데이터로 평가를 하는 경우, 해당 테스트 데이터에만 최적의 성능을 발휘하도록 **편향**됨
-> 다른 테스트용 데이터로 테스트하는 경우 성능이 급격히 저하

이를 개선하기 위해 교차검증 방식이 도입됨

교차검증의 정의

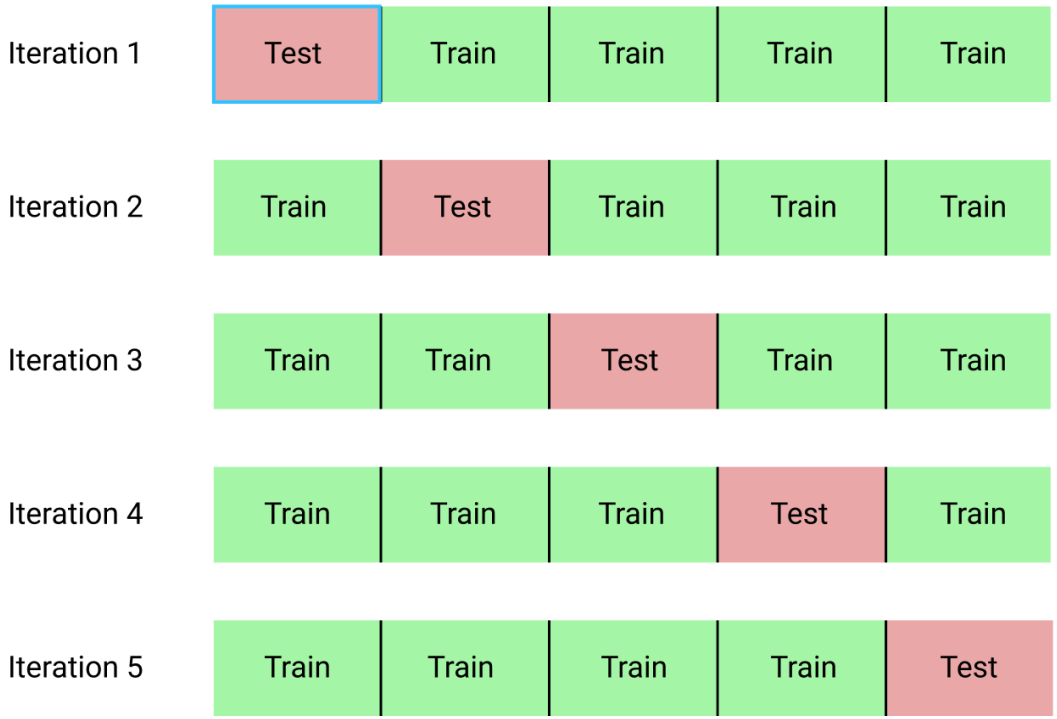
데이터 편중을 막기 위해 여러 세트로 구성된 학습 데이터 세트와 검증 데이터 세트에서 학습과 평가를 수행하는 것

➔ 각 세트 별 수행 결과에 따라 하이퍼파라미터 튜닝 등 모델 최적화를 더욱 손쉽게 할 수 있음

대부분 ML의 성능 평가는 교차검증 기반 1차 평가한 후 최종 테스트 데이터 세트에 적용하여 평가하는 방식

➔ 테스트 데이터 세트 외 별도의 검증 데이터 세트를 둬서 최종 평가 이전에 학습된 모델을 **다양하게 평가**하는 데 사용

K-fold


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

K-fold

가장 보편적으로 사용되는 교차 검증 기법

K개의 데이터 폴드 세트를 구성하여 K번 만큼 각 폴드 세트에 학습과 검증 평가를 반복적으로 수행

```
kfold = KFold(n_splits=4, random_state = rs, shuffle = True)
```

〈우승자 코드 중..〉

n_splits=4 K=4,

random_state = rs random_state는 고정되지 않음(random seed)

shuffles = True 데이터를 섞어서 샘플의 숫자를 랜덤하게 만듦 (학습 데이터의 다양성 높아짐)

주요 단점은 연산 비용이 늘어난다는 것

K-fold

In [ ]:

```
# 4FOLD, 3SEED ENSEMBLE
# 총 12개의 모델을 평균내어 예측한다

lucky_seed=[4885,1992,1022]

for num,rs in enumerate(lucky_seed):

    kfold = KFold(n_splits=4, random_state = rs, shuffle = True)
```

fix random seed for ‘reproducibility’

우승자 코드에서 seed값을 임의로 부여

-> 재현성을 위해 보통 시드값을 지정하여 고정

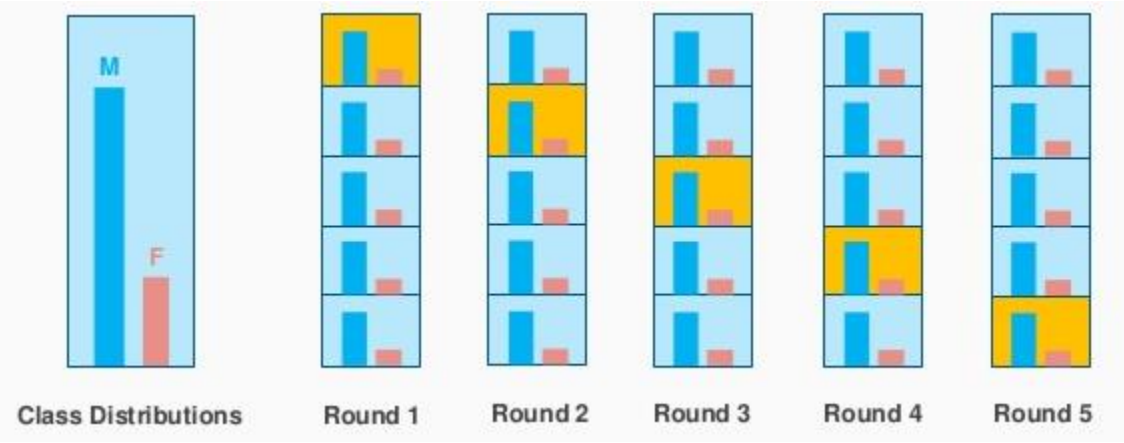
Stratified K-fold

불균형한 분포도를 가진 레이블 데이터 집합을 위한 K폴드 방식

K폴드가 레이블 데이터 집합이 원본 데이터 집합의 레이블 분포를 학습 및 테스트 세트에 제대로 분배하지 못하는 경우의 문제를 해결

➔ 원본 데이터의 레이블 분포를 고려 + 해당 분포와 동일하게 학습과 검증 데이터 세트 분배

(단, 회귀 모델의 경우 지원하지 않음)



정리

일반적인 분류에서의 교차 검증은 KFold 보다 StratifiedKFold를 이용하는 것이 더욱 적합하다고 판단

Cross_val_score()

교차 검증을 간단하게 할 수 있는 사이킷런의 API

일반적인 KFold의 코드 구성

1) 폴드 세트 설정

2) for문 반복으로 학습 및 테스트 데이터의 인덱스 추출

3) 반복적으로 학습 및 예측 수행

4) 예측 성능 반환

위 일련의 과정을 Cross_val_score()는 한꺼번에 수행

Cross_val_score()

```
cross-val-score(estimator, X, y=None, scoring=None, cv=None, n_jobs=1,
                verbose=0, fit_params=None, pre_dispatch='2*n_jobs')
```

〈cross_val_score() 선언형태〉

estimator : 사이킷런의 분류 알고리즘 클래스인 Classifier 또는 회귀 알고리즘 클래스인 Regressor을 의미

Classifier 가 입력되면 stratified K폴드 방식으로 레이블 값의 분포에 따라 학습/테스트 세트를 분할

X : 피처 데이터 세트

y : 레이블 데이터 세트

scoring : 예측 성능 평가 지표를 기술

cv : 교차 검증 폴드 수를 의미

반환값은 scoring파라미터로 지정된 성능 지표 측정값을 배열 형태로 반환

cf. Cross_val_score()는 하나의 평가지표만 반환하지만, cross_validate()는 여러 개의 평가지표를 반환할 수 있음

GridSearchCV

사이킷런에서 제공하는 API로 **교차 검증과 최적 하이퍼파라미터 튜닝을 한번에** 할 수 있음

하이퍼 파라미터 : ML 알고리즘을 구성하는 주요 구성 요소 -> 이 값을 조정하여 알고리즘의 예측 성능 개선 가능

Classifier나 Regressor 알고리즘에 사용되는 하이퍼 파라미터를 순차적으로 입력하여 **최적의 파라미터 도출**

1) 데이터 세트를 교차 검증을 위한 학습-테스트 세트로 자동 분할

2) 하이퍼파라미터그리드에 기술된 모든 파라미터를 순차적으로 적용

3) 최적의 파라미터를 찾음

즉, for루프로 모든 파라미터를 번갈아 입력하면서 학습시키는 방법을 api레벨에서 제공한 것

➔ **교차 검증을 기반으로 하이퍼 파라미터의 최적 값을 찾게 해줌**

GridSearchCV

```
GridSearchCV(estimator, prarm_grid, scoring, cv, refit)
```

〈선언 형태〉

estimator : classifier, regressor, pipeline이 사용될 수 있음

param_grid : parameters값 지정(미리 딕셔너리 형태로 설정하여 불러오기)

scoring : 예측 성능을 측정할 평가 방법을 지정 *보통은 사이킷런의 성능 평가 지표를 지정하는 문자열로(accuracy이런 식으로!)*

cv: 교차 검증을 위해 분할되는 학습/데이터 세트의 개수 지정

refit : 디폴트값 true. 가장 최적의 하이퍼 파라미터를 찾은 뒤 입력된 estimator객체를 해당 하이퍼파라미터로 재학습

연산 비용 상당히 큰 편

3주차 목표 - 모델링

데이터 로드

데이콘 baseline 사용

데이터 전처리

데이터 결손값을 1)평균 대체, 2)최빈값 대체 3) 제거 로 성능 비교

불필요한 데이터 제거

피처 스케일링

필요 시 MaxMinScaler 적용이 적절

앙상블

Log loss 방식의 측정지표를 고려 -〉 **소프트 보팅** 적용

이때 사용하는 분류기로 XGBoost, LGBM 등의 모델을 사용하여 부스팅 적용

최종 단계에서 성능 수치를 올리고자 한다면 **스태킹앙상블** 적용

검증

일반적인 분류에서의 교차 검증은 KFold 보다 **StratifiedKFold**를 이용하는 것이 더욱 적합하다고 판단

GridSearchCV 적용하여 성능 비교