데이콘 온라인 스터디 2기
2주차 발표

1. 평가 지표 이해

2. 1등 코드 리뷰

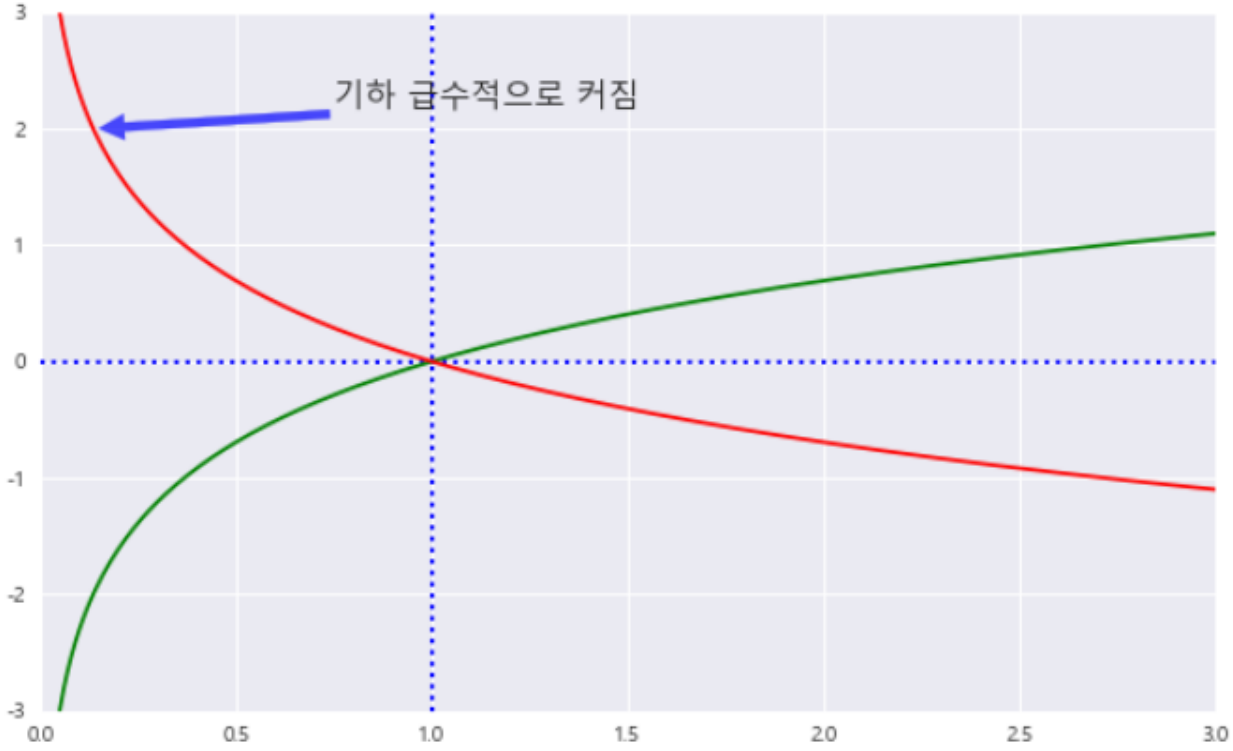
3. 개인 EDA(Exploratory Data Analysis)

정인호

1. 평가지표 이해

- 모델을 비교하는 데에는 손실 함수(Loss Function)를 이용해 어떤 모델이 좋은 모델인지를 가려냅니다.
- 해당 대회는 데이터들을 통해 Test set 하나하나가 0~197의 라벨 중 어디에 해당 되는지를 **분류**하는 대회입니다.
- 따라서 데이콘에서 평가지표를 Log loss로 정한 것은 매우 합리적입니다.

그렇다면 Log loss 란?



수식 참고 https://ynebula.tistory.com/28

- 구현한 모델에 Test Set Data를 넣으면, 0~197까지의 레이블에 확률이 부여됩니다.
- 이 확률을 음의 로그 함수인 빨간선에 대입합니다.
- 모델의 오차에 반비례하기 때문에 정답인 레이블을 높은 확률로 예측한 모델일수록 점수가 낮고, 오답일수록 점수가 높습니다.
- 따라서 오답에 높은 패널티를 부여하는 특징을 지닌 평가 지표입니다.

2. 우승자 코드 리뷰 (밍둥이)

1. 데이터 전처리
2. 모델구축&검증
3. 결과 및 결언

우승자 데이터 전처리

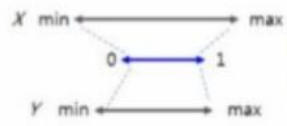
- **NAN, Object data 처리**

```
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 일괄 바꾸기
    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
```

  - Dataset에 문자열과 NA 값이 포함되어 있음
  - Dacon에서 제공한 data_loader_v2 를 사용하여 NA와 문자열 값을 0으로 대체

- **Principal Component Analysis(PCA)** ✖
  - Feature 간에 correlation 결과, high correlation feature 들이 많이 존재하였음
  - Feature간 correlation 이 높은 것들을 선별하여 feature 삭제

- **표준화 Min-Max Scaling** ✖
  - 데이터가 가진 feature의 스케일이 차이가 나는 경우 Feature 를 정규화 함
    (데이터의 중요도를 동일하게 반영되도록 하기 위함)



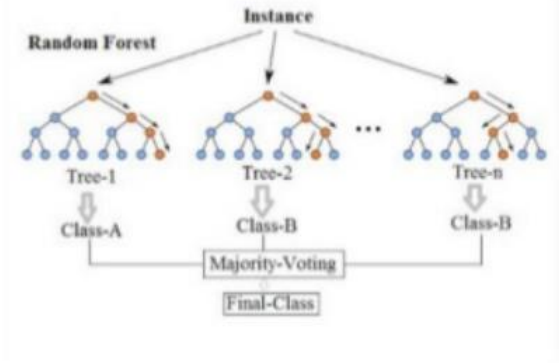
sklearn.preprocessing.MinMaxScaler()
sklearn.preprocessing.minmax_scale()

효과가 없었다.

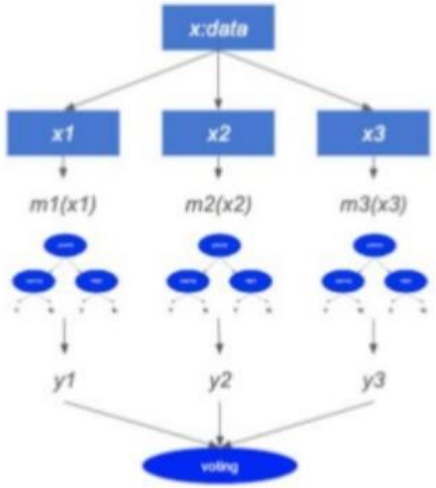
우승자 모델 선정

- **모델 선택**

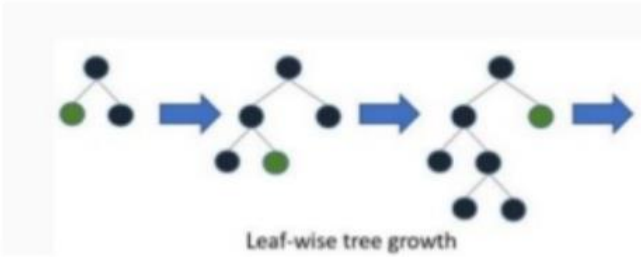
Random Forest



XGBoost



lightGBM

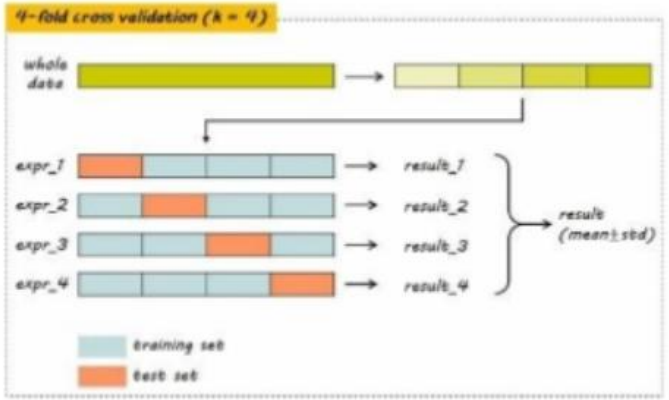


- 모델에 대해 각각 실험하여 성능 비교
- light GBM이 가장 좋은 성능을 보여 **lightGBM**을 모델로 선택

참고 자료 https://www.slideshare.net/freepsw/boosting-bagging-vs-boosting

우승자 모델 최적화

- **모델 최적화**
  - 하이퍼 파라미터 튜닝을 위해 K-fold 를 사용해 cross validation(CV) 함
  - CV를 가장 좋게 만드는 하이퍼 파라미터를 선택
    : n_estimator(Early stopping) 값을 작게 하여 overfitting 을 방지
    : min_child_weight 의 값이 낮으면 over-fitting이 되어 CV 를 통해 값을 튜닝하여 over-fitting을 방지
    : colsample_bytree 훈련 데이터에서 feature를 샘플링해주는 비율로, feature 선택
  - Data split을 robust 하게 해주기위해 3개의 random seed를 사용
  - 총 12개의 모델 생성




```
parms = {
    'learning_rate' : 0.06,
    'num_leaves' : 400,
    'n_estimators' : 300,
    'max_depth': -1,
    'min_child_weight' : 3,
    'subsample' : 0.8,
    'colsample_bytree' : 0.5,
    'objective' : 'multiclass',
    'n_jobs': -1
}
```

우승자 모델 파라미터

데이터의 크기가 큰 만큼 오버피팅을 경계하는 데에 중점

```
parms = {
    'learning_rate' : 0.06,
    'num_leaves' : 400,
    'n_estimators' : 300,
    'max_depth': -1,
    'min_child_weight' : 3,
    'subsample' : 0.8,
    'colsample_bytree' : 0.5,
    'objective' : 'multiclass',
    'n_jobs': -1
}
```

https://lightgbm.readthedocs.io/en/latest/pythonapi/lightgbm.LGBMClassifier.html

n_estimators : 트리의 개수

랜덤 포레스트에서는 이 값이 크면 좋지만, 부스팅 모델에서는 이 값이 크면 트리가 너무 많아져서 모델이 복잡해져 과적합을 일으키고, 학습 속도가 느림

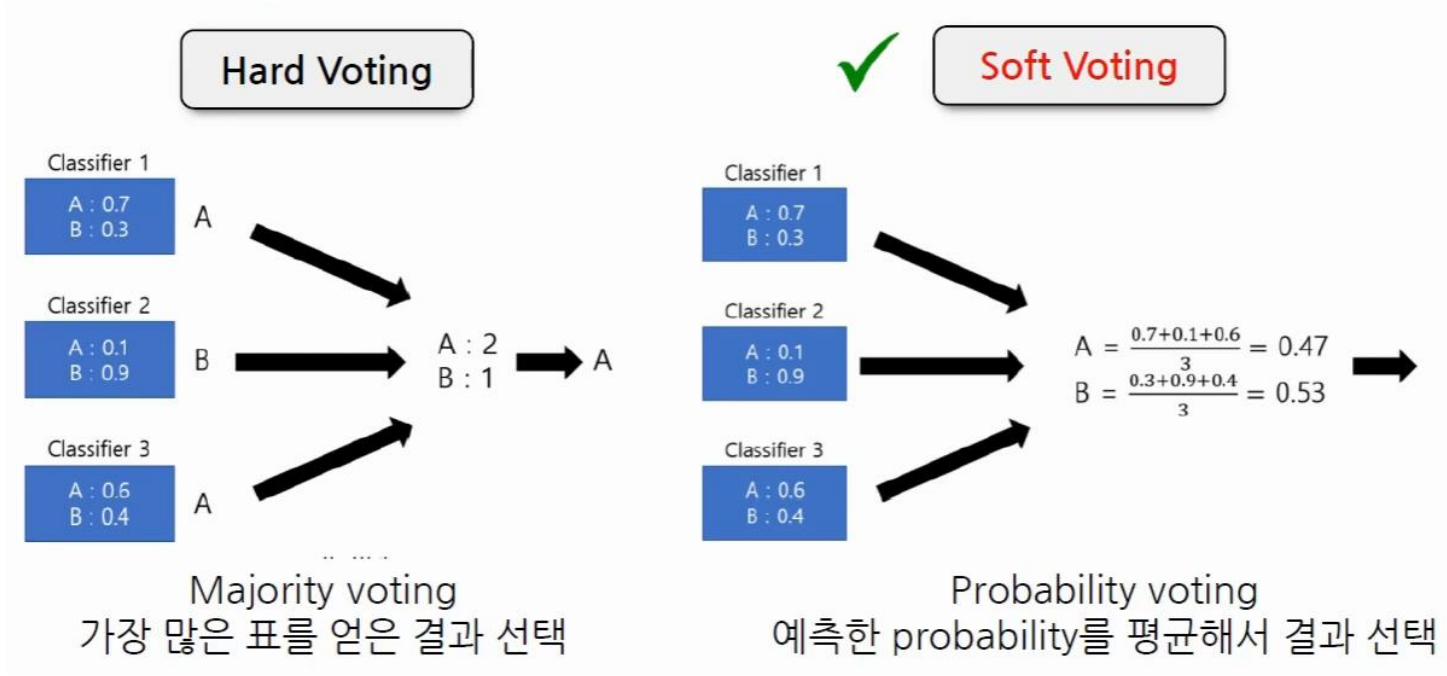
min_child_weight : 자식 노드(잎)의 최소 가중치

이 값이 너무 작으면 모델이 복잡해져 과적합을 일으킴

우승자 최종 모델 적합

12개 모델

- 0_fold_model_1022.pkl
- 0_fold_model_1992.pkl
- 0_fold_model_4885.pkl
- 1_fold_model_1022.pkl
- 1_fold_model_1992.pkl
- 1_fold_model_4885.pkl
- 2_fold_model_1022.pkl
- 2_fold_model_1992.pkl
- 2_fold_model_4885.pkl
- 3_fold_model_1022.pkl
- 3_fold_model_1992.pkl
- 3_fold_model_4885.pkl



12개의 LightGBM모델을 평균 내어 제출

우승자 코드 요약

- 데이터에 대한 정보가 너무 없기 때문에 EDA는 하지않음
- FE는 값이 컬럼 값이 일정한 피쳐들과 서로 연관성이 깊은 피쳐들이 있어서 PCA도 진행했지만, 성능이 나아지지 않아서 데이콘에서 제공한 Baseline만 전처리
- 모델 최적화에서는 과적합에 많이 주의하면서 파라미터를 튜닝했음
- 주요 전략은 난수 Seed 3개, K(4)-fold를 써서 12개의 가장 성능이 좋았던, LightGBM 모델을 Soft_voting을 이용해 우승

3. 개인 EDA

- 데이터 파악을 위한 웹 서핑을 했지만, 어떠한 정보도 얻을 수가 없음.

-> 실패

- 이미 상위 랭커들을 보면 데이터 정보의 부재 때문인지 거의 EDA나 FE는 건너뛰고, 모델링에 집중하여 순위에 오름

- 혹시나 데이터 자체를 통해서는 얻을 수 있는게 없을까 고민함

데이터에 대한 고심

Train

상위 랭커들은 모두 baseline을 건들지 않았기 때문에

827개의 train 데이터가 10초~60초까지 50개씩 41,350 row로 이루어져있음.

데이콘에서는 각 Train 파일마다 Event time을 10초로 고정해 놓았지만, Train 파일마다 상태가 변하는 Event time이 다를 수 있다고 말함.

데이터에 대한 고심1

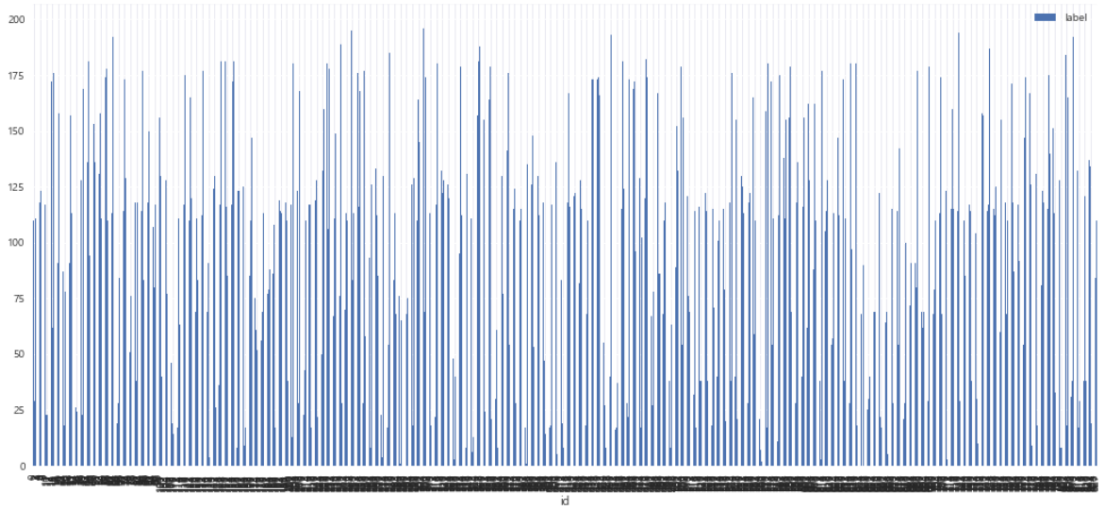
데이콘에서는 각 Train 파일마다 Event time을 10초로 고정해 놓았지만,
Train 파일마다 상태가 변하는 Event time이 다를 수 있다고 말함.
즉, Event time이 Label과 관계가 있을 수 있다.

1등한 밍둥이 팀의 스케일링, PCA 코드는 공개되지 않아서 모르지만,

아마 data_roader_v2로 Train 데이터를 전부 concat한 후에 41350 row의 데이터에 대해 스케일링과 PCA를 했더라면,

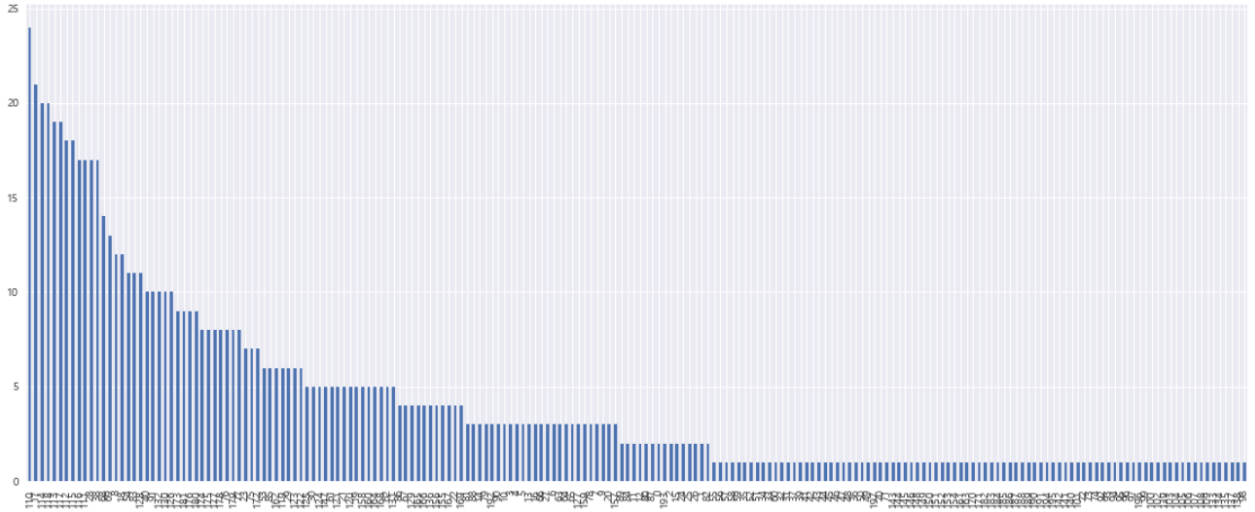
Concat 하기전에 827개의 데이터를 스케일링과 PCA를 한다면 성능은 달라질 수 있다고 생각함

시각화



X축: 0~ 826의 train 파일 (정렬됨)
Y축: 0~ 197의 라벨

특징을 가진 패턴을 찾기는 힘듦



0~197의 라벨들에 속한 train의 개수를 시각화

110라벨에 속한 Train이 가장 많았고, 이는 24개의 train이 속했다. 약(2.9%)

1개의 라벨을 대표하는 train이 굉장히 많이 있음
-> Data Imbalance가 너무 심함