DACON

Dacon 2회 천체 유형 분류 모델링 경진대회

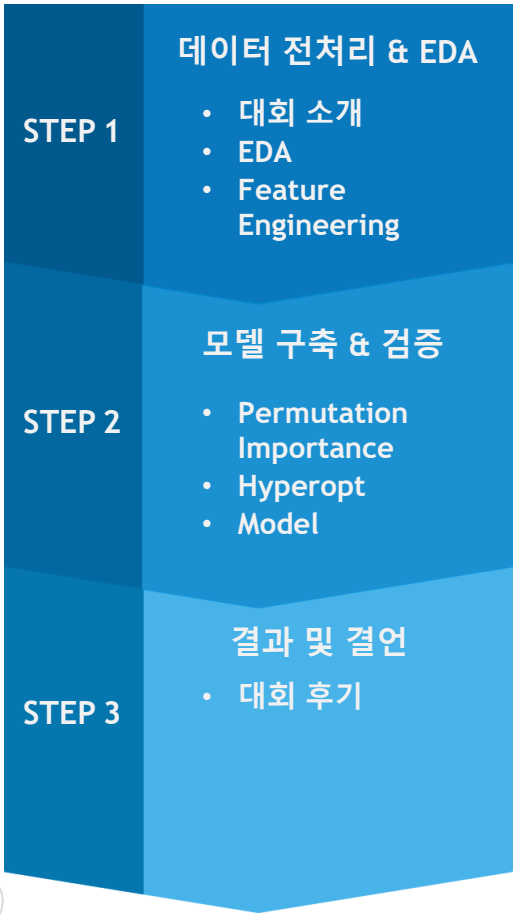
처음해봐요

목차

1 데이터 전처리 & EDA

2 모델 구축& 검증

3 결과 및 결언



1. 데이터 전처리 & EDA – 대회 소개

1. 배경

찰나의 순간에도 우주는 천문학적인 양의 데이터를 생산해왔고, 오래 전부터 천문학자들은 우주를 관측했으며 그 방대함에 비례하는 데이터를 수집 및 분석했기 때문입니다.

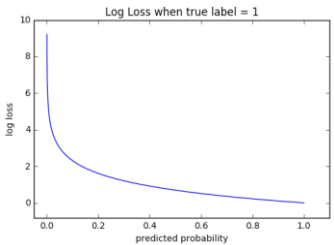
슬론 디지털 천체 관측(Sloan Digital Sky Survey: 이하 SDSS)는 세계적 천체 관측 프로젝트로, 우주에 대한 천문학적인 규모의 데이터를 수집하고 있습니다. 점점 거대해지는 규모에 따라 **데이터 처리에는 머신러닝과 딥러닝 기법이 활용되기 시작**했습니다.

여전히 우주에는 다양한 미지의 이야기가 남아있고, 오늘날 인간은 하늘에서 많은 데이터를 얻어낼 정도로 발전했습니다. 이 데이터를 분석하여 어쩌면 드러나지 않은 규칙이 여러분의 손끝에서 밝혀질 수 있습니다. **새로운 알고리즘을 통해 우주의 비밀을 찾아주세요!**

2. 평가 지표

- Log Loss

$$-\sum_{c=1}^{M} y_{o,c} \log(p_{o,c})$$



Log Loss의 특성

3. 대회 기간

- Free Stage(~2월 21일 23:59)
- Regular Stage Ⅰ(~2월 23일 23:59) if score > baseline_1
- Regular Stage Ⅱ(~2월 25일 23:59) if score > baseline_2
- Final Stage(~2월 29일 23:59) if ranking > 50%

4. 대회 규칙

- 본 대회에는 어떠한 외부데이터 사용이 불가합니다.
- Public: 30%, Private: 70%

1. 데이터 전처리 & EDA – EDA

1. 데이터 라이센스

- https://www.sdss.org/dr16/
- Dacon에서 별도 가공
- CC BY 4.0 라이선스

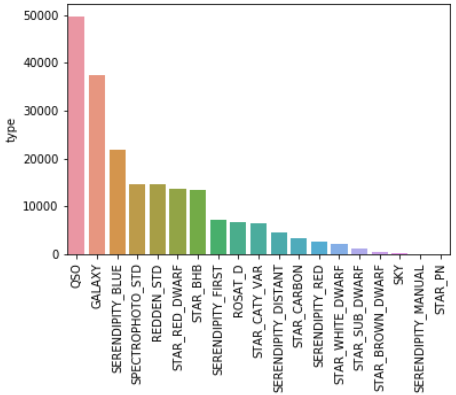
2. 데이터 형태

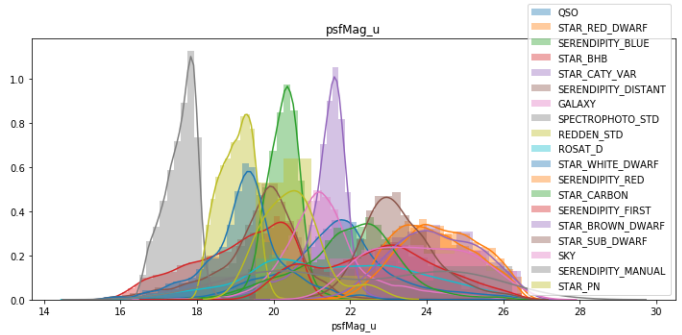
- 데이터 크기: Train(199991, 24), Test(10009, 22)

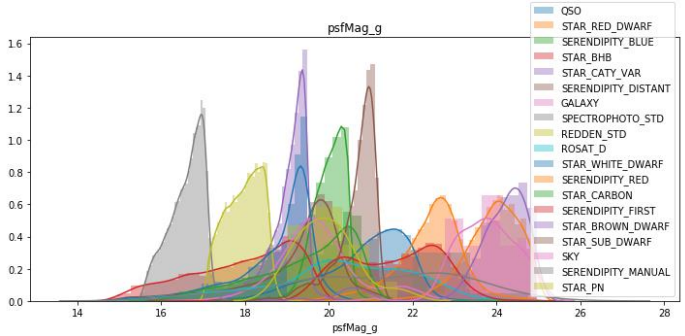
- Type: Target **19개 천체 Type의 Multi Class 분류문제**
  - http://skyserver.sdss.org/dr8/en/help/browser/enum.asp?n=PrimTarget

- psfMag: Point spread function magnitudes : 먼 천체를 한 점으로 가정하여 측정한 빛의 밝기입니다.
- fiberMag: Fiber magnitudes : 3인치 지름의 광섬유를 사용하여 광스펙트럼을 측정합니다. 광섬유를 통과하는 빛의 밝기입니다.
- petroMag: Petrosian Magnitudes : 은하처럼 뚜렷한 표면이 없는 천체에서는 빛의 밝기를 측정하기 어렵습니다. 천체의 위치와 거리에 상관없이 빛의 밝기를 비교하기 위한 수치입니다.
- modelMag: Model magnitudes : 천체 중심으로부터 특정 거리의 밝기입니다.
- fiberID: 관측에 사용된 광섬유의 구분자

1. 데이터 전처리 & EDA – EDA



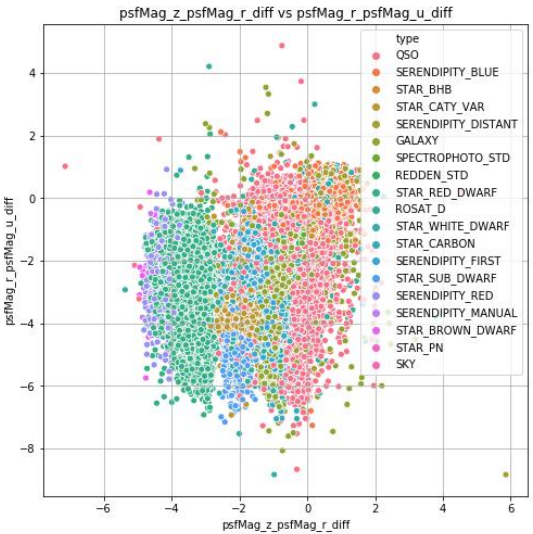




- 일부 Target Class는 매우 드문 것을 볼 수 있습니다.
- Target별로 매우 다른 magnitude 분포를 가진 것을 볼 수 있고, 이러한 특징으로 볼 때 원본 magnitude도 좋은 feature가 될 수 있습니다.

1. 데이터 전처리 & EDA – EDA

- 2개의 Filter difference를 scatterplot으로 표현해보면 target들이 구분되는 것을 볼 수 있고 이러한 difference feature들의 상호작용이 좋은 효과를 낼 것으로 기대됩니다.
- 또한 SDSS Algorithm을 살펴보면 r-I, g-r 같은 diff feature들로 천체를 구분하는 것을 알 수 있습니다.

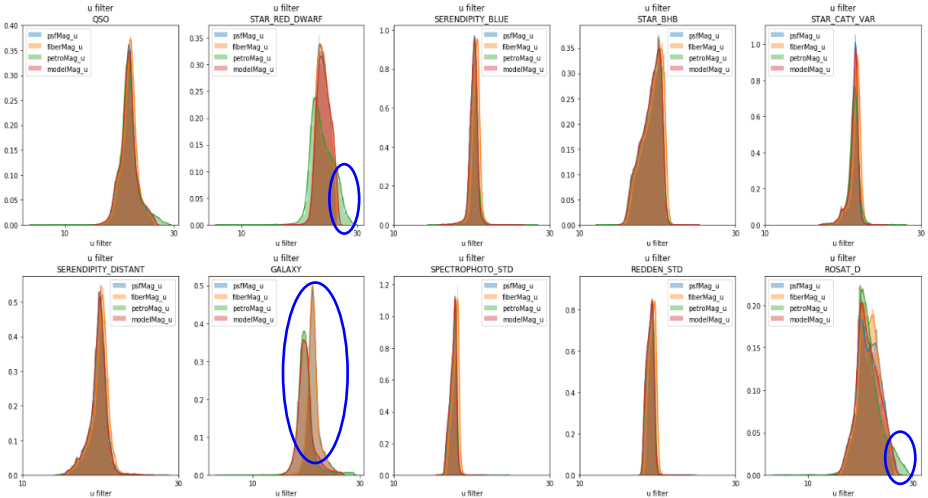


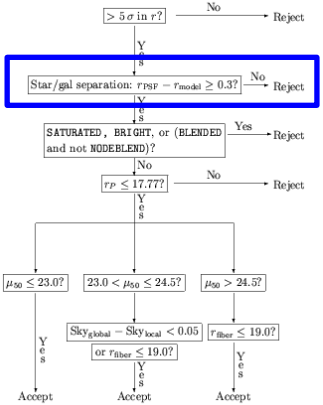
Red Dwarf	0x00040000	262144	2^{18}	i < 19.5 psfmagerr(i) < 0.2 (r-i) > 1.0 (r-i) > 1.8	14649
White Dwarf	0x00080000	524288	2^{19}	(g-r) < -0.15 (u-g)+2(g-r) < 0 (r-i) < 0 H_g > 19 H_g > 16.136 + 2.727(g-i) (g-i) > 0	6059

https://www.sdss.org/dr16/algorithms/segue_target_selection/

1. 데이터 전처리 & EDA – EDA

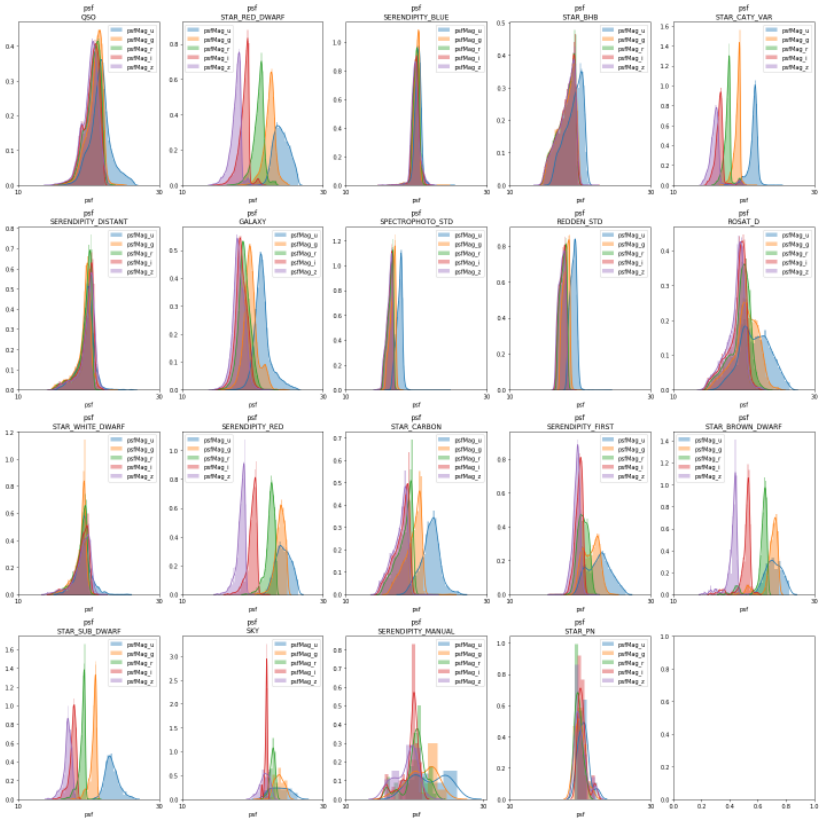
- 모든 Magnitude의 같은 filter u를 Target 별로 살펴봤을 때 대부분의 분포가 Magnitude에서 비슷한 것을 볼 수 있지만 일부 Target은 구분되는 것을 볼 수 있습니다.
- Legacy Galaxy Algorithm에서도 다른 Magnitude filter끼리 빼는 것을 볼 수 있습니다.





https://www.sdss.org/dr16/algorithms/legacy_target_selection/

1. 데이터 전처리 & EDA – EDA



- Magnitude의 첨도가 다르기 때문에 표준편차로 Target을 구분할 수 있을 것 같습니다.
- 일부 Target은 중심값의 위치가 다릅니다.
- Max Magnitude, Min Magnitude의 Filter가 다릅니다.
- Magnitude Min, Max의 차이가 큰 것도 있습니다.
- 분포가 겹치는 것도 있지만 Filter별로 겹치지 않는 것도 있습니다.

1. 데이터 전처리 & EDA – Feature Engineering

EDA 결과와 SDSS 문서를 바탕으로 다양한 Feature를 추가합니다.

1. Magnitude, row별 max, min, max-min, std, sum Feature 추가

```
# zip 함수를 이용하여 각 Row별, Magnitude별 max, min, max-min, std, sum을 구한다.
# mean, skew, 등 다른 것들 시도 시 cv 점수가 안 좋아져서 사용하지 않음
for prefix, g in zip(['psfMag','fiberMag','petroMag','modelMag'], [psfMag_col, fiberMag_col, petroMag_col, modelMag_col]):
    train[f'{prefix}_max'] = train[g].max(axis=1)
    test[f'{prefix}_max'] = test[g].max(axis=1)

    train[f'{prefix}_min'] = train[g].min(axis=1)
    test[f'{prefix}_min'] = test[g].min(axis=1)

    train[f'{prefix}_diff'] = train[f'{prefix}_max'] - train[f'{prefix}_min']
    test[f'{prefix}_diff'] = test[f'{prefix}_max'] - test[f'{prefix}_min']

    train[f'{prefix}_std'] = train[g].std(axis=1)
    test[f'{prefix}_std'] = test[g].std(axis=1)

    train[f'{prefix}_sum'] = train[g].sum(axis=1)
    test[f'{prefix}_sum'] = test[g].sum(axis=1)
```

2. 모든 magnitude들의 조합(2)으로 diff feature 추가

```
# diff feature 추가 예: psfMag_z - psfMag_i
# sdss lagacy solution 등을 보면 대 부분 mag간 차이를 사용하기 때문에 이런 diff feature가 의미가 있을 것이라고 판단
# 그리고 각 magnitude에서만 diff를 구하는 것이 아닌 itertools combinations를 활용하여 전체 magnitude에서 diff를 구함
# 총 190가지 조합이 나오고 여기서 안 좋은 것은 permutation importance를 활용하여 feature 제거 수행
diff_feature = []
for c1, c2 in itertools.combinations(psfMag_col[::-1]+fiberMag_col[::-1]+petroMag_col[::-1]+modelMag_col[::-1],2):
    new_c = f'{c1}_{c2}_diff'
    train[new_c] = train[c1]-train[c2]
    test[new_c] = test[c1]-test[c2]
    diff_feature.append(new_c)
```

※ Magnitude List에 ::-1 해주는 것은 z -> i -> r -> g -> u 순서로 difference를 구하기 위해서입니다. 처음에 이미 이렇게 정하고 Permutation Importance로 Feature Selection을 수행한 상태라, u->g->r->i->z 순서의 성능 테스트는 하지 못했습니다.

EDA 결과와 SDSS 문서를 바탕으로 다양한 Feature를 추가합니다.

3. 각 magnitude 별 max-max, min-min, sum-sum 추가

```
# 각 magnitude 별 max-max, min-min, sum-sum 을 구함
for c in itertools.combinations(['psfMag','fiberMag','petroMag','modelMag'],2):
    train[f'{c[0]}_{c[1]}_max_diff'] = train[f'{c[0]}_max'] - train[f'{c[1]}_max']
    test[f'{c[0]}_{c[1]}_max_diff'] = test[f'{c[0]}_max'] - test[f'{c[1]}_max']

    train[f'{c[0]}_{c[1]}_min_diff'] = train[f'{c[0]}_min'] - train[f'{c[1]}_min']
    test[f'{c[0]}_{c[1]}_min_diff'] = test[f'{c[0]}_min'] - test[f'{c[1]}_min']

    train[f'{c[0]}_{c[1]}_sum_diff'] = train[f'{c[0]}_sum'] - train[f'{c[1]}_sum']
    test[f'{c[0]}_{c[1]}_sum_diff'] = test[f'{c[0]}_sum'] - test[f'{c[1]}_sum']
```

4. Ugriz -> sdssUBVRITransform

```
def quasar_UB_jester(x1, x2):
    return 0.75*(x1-x2)-0.81

make_2flux_feature(train, test, 'u','g',quasar_UB_jester,['psfMag'])

def quasar_BV_jester(x1, x2):
    return 0.62*(x1-x2)+0.15

make_2flux_feature(train, test, 'g','r',quasar_BV_jester,['psfMag'])

def quasar_VR_jester(x1, x2):
    return 0.38*(x1-x2)+0.27

make_2flux_feature(train, test, 'r','i',quasar_VR_jester,['psfMag'])

def quasar_RcIc_jester(x1, x2):
    return 0.72*(x1-x2)+0.27

make_2flux_feature(train, test, 'u','g',quasar_RcIc_jester,['psfMag'])
```

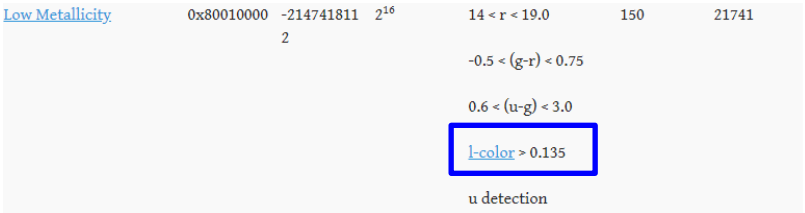
```
ugriz -> UBVRcIc
================

Quasars at z <= 2.1 (synthetic)
      Transformation                RMS residual
U-B   =   0.75*(u-g) - 0.81         0.03
B-V   =   0.62*(g-r) + 0.15         0.07
V-R   =   0.38*(r-i) + 0.27         0.09
Rc-Ic =   0.72*(r-i) + 0.27         0.06
B     =   g + 0.17*(u-g) + 0.11     0.03
V     =   g - 0.52*(g-r) - 0.03     0.05

Stars with Rc-Ic < 1.15 and U-B < 0
      Transformation                RMS residual
U-B   =   0.77*(u-g) - 0.88         0.04
B-V   =   0.90*(g-r) + 0.21         0.03
V-R   =   0.96*(r-i) + 0.21         0.02
Rc-Ic =   1.02*(r-i) + 0.21         0.01
B     =   g + 0.33*(g-r) + 0.20     0.02
V     =   g - 0.58*(g-r) - 0.01     0.02

All stars with Rc-Ic < 1.15
      Transformation                RMS residual
U-B   =   0.78*(u-g) - 0.88         0.05
B-V   =   0.98*(g-r) + 0.22         0.04
V-R   =   1.09*(r-i) + 0.22         0.03
Rc-Ic =   1.00*(r-i) + 0.21         0.01
B     =   g + 0.39*(g-r) + 0.21     0.03
V     =   g - 0.59*(g-r) - 0.01     0.01
```

※ 이 Feature가 정확히 어떤 Feature인지 모르겠으나 성능 Test 시 CV, LB 모두 좋아져서 사용하였습니다. 처음에는 모든 magnitude에서 변환을 시도하였으나 Test 시 psfMag만 변환 시 성능이 좋았습니다.

EDA 결과와 SDSS 문서를 바탕으로 다양한 Feature를 추가합니다.

5. fiberID별 fiberMag mean, (fiber_u,g,r,i,z)/fiberMag_mean
   - 다른 대회를 할 때 많이 사용하는 Category 변수를 Source로 두고 연속 변수의 mean, std 같은 통계값을 추가하였습니다.
   - Filter값을 Fiber filter mean 값으로 나누어 주었습니다.

6. SDSS 문서를 바탕으로 피쳐를 추가하였습니다.
   - 도메인 지식이 없기 때문에 정확히 어떤 역할을 하는지는 모르지만 어떤 Target을 분류할 때 사용하는 값이라고 생각되어 구현하였습니다.



☆ $c_{\perp} = (r-i) - (g-r)/4 - 0.177$

☆ $c_{\parallel} = 0.7(g-r) + 1.2[(r-i) - 0.177]$

★ l-color = -0.436u + 1.129g - 0.119r - 0.574i + 0.1984 and is a photometry metallicity indicator for stars in the color range 0.5 < (g-r) < 0.8.
Reference: Lenz et al. (1998)

★ s-color = -0.249u + 0.794g - 0.555r + 0.234

★ P1(s) = P1 = 0.91(u-g) + 0.415(g-r) -1.280
Reference: Helmi et al. (2003)

https://www.sdss.org/dr16/algorithms/segue_target_selection/#Legacy
https://www.sdss.org/dr16/algorithms/legacy_target_selection/

1. 데이터 전처리 & EDA – Feature Engineering

EDA 결과와 SDSS 문서를 바탕으로 다양한 Feature를 추가합니다.

7. Asinh 변환

$$m = -2.5/\ln(10) * [\operatorname{asinh}((f/f_0)/(2b)) + \ln(b)].$$

Asinh Softening Parameters

Filter	b	Zero-flux Magnitude $[m(f/f_0 = 0)]$	$m(f/f_0 = 10b)$
u	1.4×10^{-10}	24.63	22.12
g	0.9×10^{-10}	25.11	22.60
r	1.2×10^{-10}	24.80	22.29
i	1.8×10^{-10}	24.36	21.85
z	7.4×10^{-10}	22.83	20.32

8. Diff feature들의 표준 편차

9. Original Magnitude만 차원 축소하여 Feature로 사용

```
org_feature = psfMag_col+fiberMag_col+petroMag_col+modelMag_col
# decompostion해서 다시 feature로 추가, 원래 original feature만 사용하고 5개로 축소
decom_common_param = {'n_components':5,'random_state':42}
train, test = get_decomposition_feature(train, test, org_feature, decom_common_param, TruncatedSVD, 'tsvd5')
train, test = get_decomposition_feature(train, test, org_feature, decom_common_param, FastICA, 'ica5')
```

https://www.sdss.org/dr12/algorithms/magnitudes/

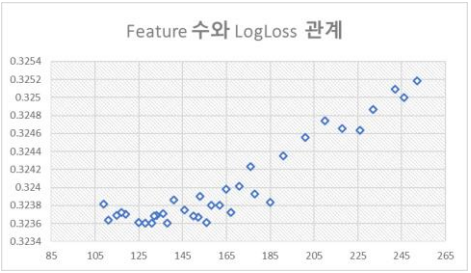
2. 모델 구축 & 검증 – Permutation Importance

Permutation Importance란 **기능을 사용할 수 없을 때 Score(LogLoss, Accuracy 등)가 얼마나 감소하는지를 측정하여 Feature의 영향도를 측정하는 방법**입니다.
각 데이터 세트에서 기능을 제거하고 다시 학습시켜 점수를 확인해야 하는데 이렇게 되면 Feature가 많을 때 계산 비용이 많이 발생합니다.
이러한 것을 해결하고자 전체 데이터로 한번 학습하고 **predict시 원본 feature를 제거하고 제거된 feature 값에 random-noise를 추가하여 학습**합니다. 이렇게 되면 이 피쳐는 더 이상 feature 기능을 하지 않는 것이기 때문에 제거한 것과 마찬가지의 효과를 가지고 이 때 점수가 얼마나 감소하는지 확인하는 방법입니다. (특히 **random-noise를 추가하는 가장 간편한 방법은 값을 섞는 것**이라고 합니다.)

eli5 library의 Permutation Importance를 사용하여 학습이 끝난 뒤 get_score_importance에 score 계산을 해주는 함수, x, y, iteration, seed값을 넣고 score가 얼마나 감소하는지 return 받습니다.

```
def score(X, y):
    y_pred = clf.predict(X)
    return 1-log_loss(y, y_pred)

print("get_score_importances")
base_score, score_decreases = get_score_importances(score, val_x.values, val_y.values, n_iter=2, random_state=SEED)
perm_importances = np.mean(score_decreases, axis=0)
fi_df['perm_importance'] += perm_importances/(NFOLD)
```

이렇게 Permutation Importance가 구해지고 나면 실험을 통해 적절한 Cutoff를 정하여 추후 학습 시 이 피쳐들을 제거합니다.
성능 향상이 약 CV, LB 0.02~0.03 정도 있었습니다.



2. 모델 구축 & 검증 – HyperOpt

LightGBM Parameter는 모두 HyperOpt를 사용하여 찾았습니다.

제 컴퓨터에서 boosting_type: gbdt StratifiedKFold 5Fold가 약 20~30분 정도 걸렸는데 대부분 1~2Fold의 성능이 향상되면 3~5Fold도 성능이 향상되는 것을 확인하여 Dataset은 기존대로 5Fold 나누고 2Fold까지만 CV Score 확인하여 HyperOpt를 진행하였습니다.

탐색 Space는 아래와 같고 100회 안에 대부분 기존보다 좋은 성능을 가진 Parameter를 찾아내었습니다.

```
space = {
    'subsample_freq':hp.quniform('subsample_freq', 1, 10, 1),
    'num_leaves': hp.quniform('num_leaves', 4, 512, 2),
    'max_depth': hp.quniform('max_depth', 4, 17, 1),
    'min_data_in_leaf': hp.quniform('min_data_in_leaf', 1, 144, 2),
    'feature_fraction': hp.uniform('feature_fraction', 0.3, 1.0),
    'bagging_fraction': hp.uniform('bagging_fraction', 0.3, 1.0),
    'lambda_l1': hp.uniform('lambda_l1', 0, 10.0),
    'lambda_l2': hp.uniform('lambda_l2', 0, 10.0),
    'min_child_weight': hp.uniform('min_child_weight', 0, 50.0),
    'learning_rate': hp.uniform('learning_rate', 0.01, 0.1),
    'min_sum_hessian_in_leaf': hp.quniform('min_sum_hessian_in_leaf', 1, 144, 1),
    'subsample_for_bin': hp.quniform('subsample_for_bin', 1000, 50000, 1000),
    }

best = fmin(fn=objective,
        space=space,
        algo=tpe.suggest,
        max_evals=100)
```

먼저 boosting_type: gbdt로 위의 Parameter를 탐색하고 dart는 EarlyStopping을 사용할 수 없기 때문에 위 Parameter를 고정하고 DART의 Parameter만 따로 탐색하였습니다.
그래서 기본 DART Parameter에서 num_boost_round를 어느정도 해야 결과가 좋은지 찾고 그 Iteration까지 고정 후 HyperOpt를 수행하였습니다.

2. 모델 구축 & 검증 – Model

최종적으로 **LightGBM DART Single Model**로 Public 0.3255 Private 0.3246 점수를 기록했습니다.

Public 기준
GBDT는 0.3285
GOSS는 0.330
DART는 0.3255

속도
GOSS > GBDT > DART

DART 특징
Gradient Boosting 알고리즘은 높은 정확도를 가지지만 나중에 추가된 트리는 큰 기여를 하지 못하고 아주 작은 영향만 미치게 됩니다. 이는 초기에 추가된 일부 tree 과민하게 되며 성능에 부정적인 영향을 미치게 됩니다. 이러한 것을 over-specialization이라고 하는데 DART는 dropout을 사용하여 이러한 문제를 해결하려고 합니다.

Early Stopping을 사용할 수 없고 속도가 느립니다.

대부분의 대회에서 gbdt보다 근소하게 성능이 좋거나 비슷했는데,
느리기 때문에 보통 gbdt로 Feature Engineering을 하고 나중에 ensemble 시 gbdt+dart를 사용하였습니다.

이 번 대회에서는 Single Model도 성능이 좋아서 DART Single Model로 마감하였습니다.

2. 모델 구축 & 검증 – Model

```
# hyper optimization으로 찾아낸 parameter
# lightgbm dart 사용, 보다 lb 0.03 정도 좋음
# gbdt가 0.3285라면 dart는 0.3255, goss는 0.3300
lgb_param_dart = {'objective': 'multiclass',
 'num_class': 19,
 'boosting_type': 'dart',
 'subsample_freq': 5,
 'num_leaves': 92,
 'min_data_in_leaf': 64,
 'subsample_for_bin': 23000,
 'max_depth': 10,
 'feature_fraction': 0.302,
 'bagging_fraction': 0.904,
 'lambda_l1': 0.099,
 'lambda_l2': 1.497,
 'min_child_weight': 38.011,
 'nthread': 32,
 'metric': 'multi_logloss',
 'learning_rate': 0.021,
 'min_sum_hessian_in_leaf': 3,
 'drop_rate': 0.846244,
 'skip_drop': 0.792465,
 'max_drop': 65,
 'seed': 42,
 'n_estimators': 1000}
```

왼쪽에 보이는 Parameter를 최종적으로 사용하였습니다.

StartifiedKFold를 사용했고 5Fold로 검증하였으며, Stratified는 target값을 사용하여 각 Fold마다 Target의 비율이 비슷하게 분배되도록 하였습니다.

```
# stratifiedkfold 5 fold 사용
folds = StratifiedKFold(n_splits=NFOLD, shuffle=True, random_state=42)
```

최종 모델 성능

```
[0.3200738375559559, 0.32134818588686026, 0.3181328008151575, 0.32361092772752614, 0.32214748527121173]

(0.3210626474513423, 0.001860853781485379)
```

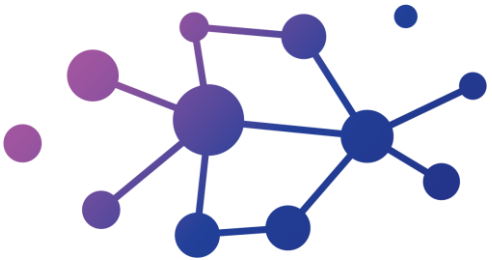
Public LB: 0.325529
Private LB: 0.324615

3. 결과 및 결언 - 대회 후기

운이 좋게도 Private 1등으로 마감하였습니다.

기존에 PLAsTiCC 이라는 대회에 참여하면서 배운 Domain 지식이 조금 도움이 되었습니다.
https://www.kaggle.com/c/PLAsTiCC-2018

다양한 Ensemble을 해보려 했지만 효과가 좋지 않아서 Feature Engineering에 집중하였고 그게 좋은 성적으로 이어진 것 같습니다.



THANK YOU

대회 바로가기