https://www.kaggle.com/datafan07/analysis-of-melanoma-metadata-and-effnet-ensemble

3주차. Analysis of Melanoma Metadata and EffNet Ensemble

최하경

목차

1. 기본적인 정보

데이터셋 정보

(1)Train data : 33126개

- image name
- patient_id
- sex
- age_approx
- anatom_site_general_challenge
  (location으로 변경)
- diagnosis
- benign_malignant
- target

(2) Test data : 10982개

- image name
- patient_id
- sex
- age_approx
- anatom_site_general_challenge
  (location으로 변경)

1. 기본적인 정보

알아내고자 하는 것

(1) 데이터가 어떻게 생겼는지

(2) 데이터셋이 완벽한지

(3) target의 unbalance 확인

(4) 스캔된 위치가 결과에 어떤 영향을 주는지

(5) 나이가 피부 병변 (skin lesion)에 영향을 주는지

(6) 성별에 따라 차이가 있는지

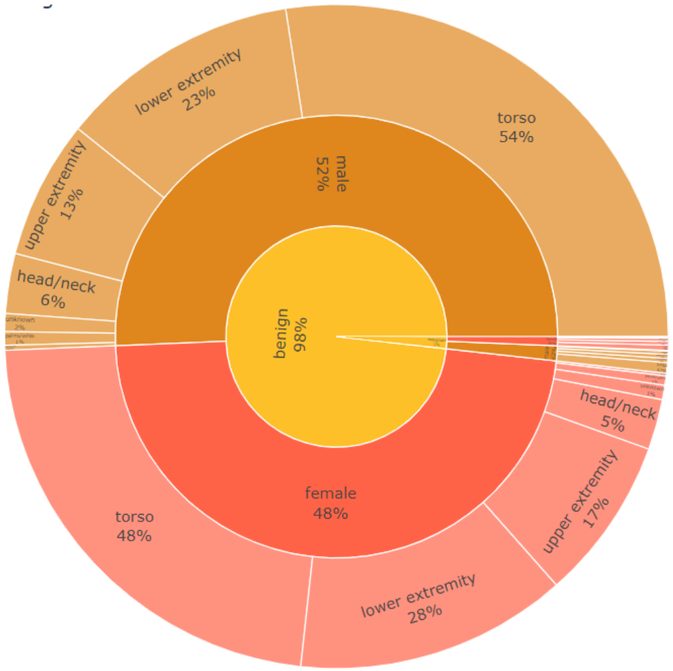
(7) unique한 환자 데이터가 얼마나 있는지 & 중요한지

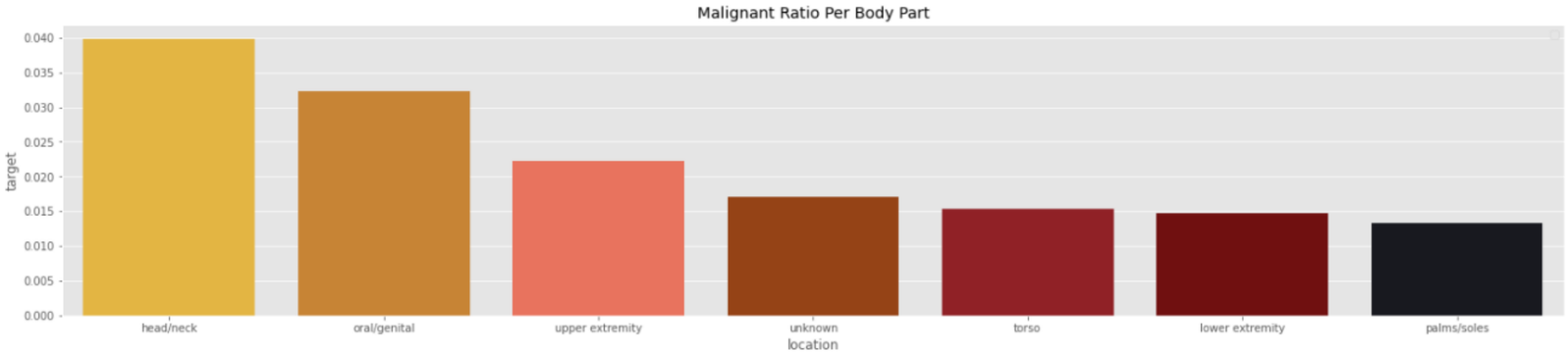
(8) 이미지 퀄리티나 색깔, 사이즈가 결과에 영향을 미치는지

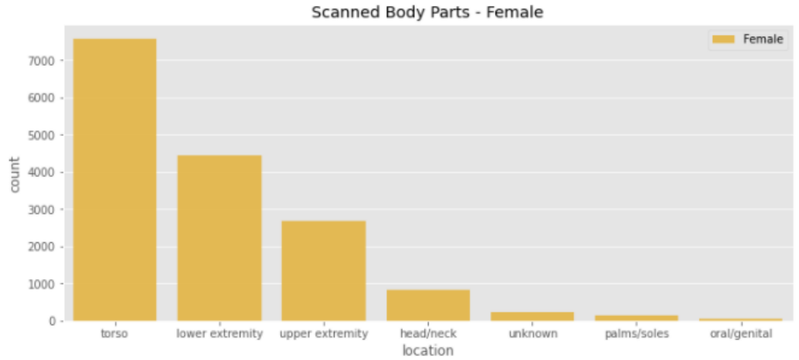
(9) train, test data들에서 비슷한 관찰 결과가 보이는지 & 아니라면 그 원인은?

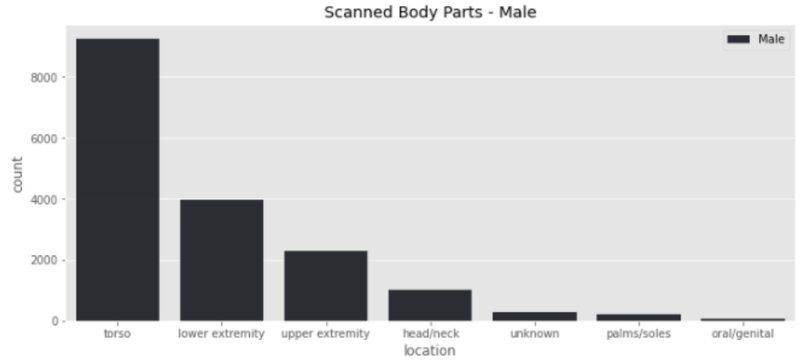
1. 기본적인 정보

알아내고자 하는 것









=> EDA를 진행하며 확인 (plotly 사용)

2. Adversarial Validation

https://www.kaggle.com/carlmcbrideellis/what-is-adversarial-validation

(1) 목적

- 학습된 모델의 overfitting을 피하기 위해 고안된 방법
- 이진분류를 이용해 train, test data가 서로 얼마나 비슷한지 확인

Train, test data의 분포가 달라,
train data에서는 예측성능이 좋아도
test data에서의 성능은 떨어질 수 있다는 의미

(2) Procedure

- train data에서 target column을 제거
- train, test data를 각각 0 또는 1로 라벨링
- train, test data를 하나의 데이터셋으로 병합
- 이진분류 모델을 사용해 AUC ROC 결과 확인

2. Adversarial Validation

(3) 결과 해석

- ROC < 0.5 일 때, train과 test data를 구분하지 못하는 것이므로 좋은 상태
- ROC > 0.5 일 때, train과 test data를 구분하므로 안 좋은 상태

(4) ROC > 0.5 일 때 해결방법

- 중요도 또는 ROC가 높은 피처를 제거

이 방법 말고는 다른 방법이 없는지 궁금 (>.<)

https://medium.com/mlearning-ai/adversarial-validation-battling-overfitting-334372b950ba

2. Adversarial Validation

해당 노트에서의 코드

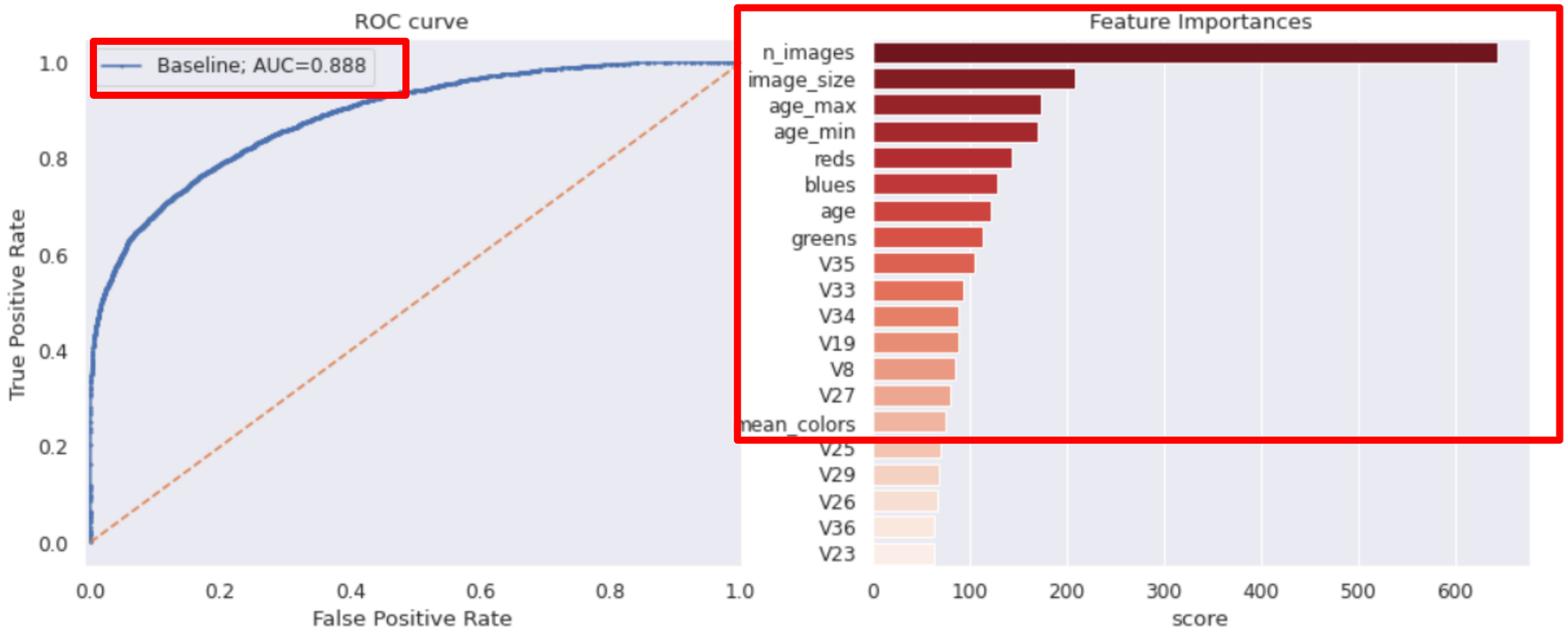
```
adv_train = train.copy()
adv_train.drop('target', axis=1, inplace=True)
adv_test = test.copy()

adv_train['dataset_label'] = 0
adv_test['dataset_label'] = 1

adv_master = pd.concat([adv_train, adv_test], axis=0)

adv_X = adv_master.drop('dataset_label', axis=1)
adv_y = adv_master['dataset_label']
```

```
xg_adv = xgb.XGBClassifier(
    random_state=42,
    n_jobs=-1,
)

# Fitting train data

xg_adv.fit(adv_X_train, adv_y_train)

# Predicting on holdout set
validation = xg_adv.predict_proba(adv_X_test)[:,1]
```

```
adv_X_train, adv_X_test, adv_y_train, adv_y_test = train_test_split(adv_X,
                                                    adv_y,
                                                    test_size=0.4,
                                                    stratify=adv_y,
                                                    random_state=42)
```

2. Adversarial Validation

해당 노트에서의 Adversarial Validation 결과



이미지 사이즈와 숫자와 관련된 column들 모두 drop

2. Adversarial Validation

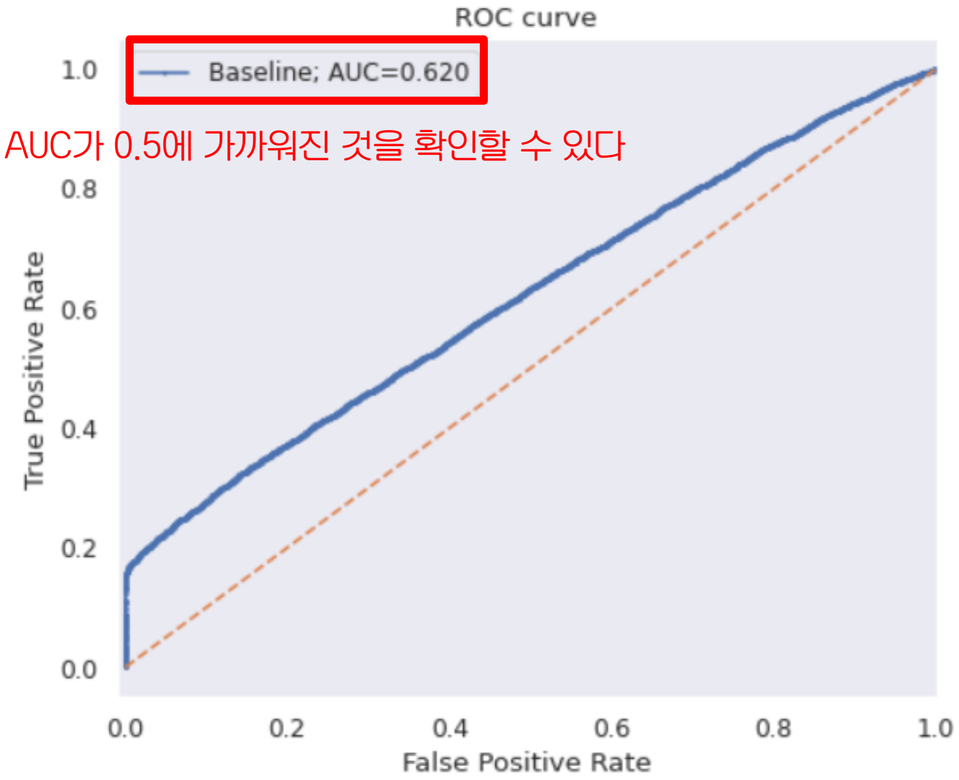
해당 노트에서의 Adversarial Validation 결과

```
adv_X.drop(['n_images', 'image_size','width','height','total_pixels','reds','blues','greens','mean_colors',
            'age_min', 'age_max'], axis=1, inplace=True)


adv_X_train, adv_X_test, adv_y_train, adv_y_test = train_test_split(adv_X,
                                                    adv_y,
                                                    test_size=0.4,
                                                    stratify=adv_y,
                                                    random_state=42)

# fitting train data

xg_adv.fit(adv_X_train, adv_y_train)

# predicting on holdout set
validation = xg_adv.predict_proba(adv_X_test)[:,1]
```

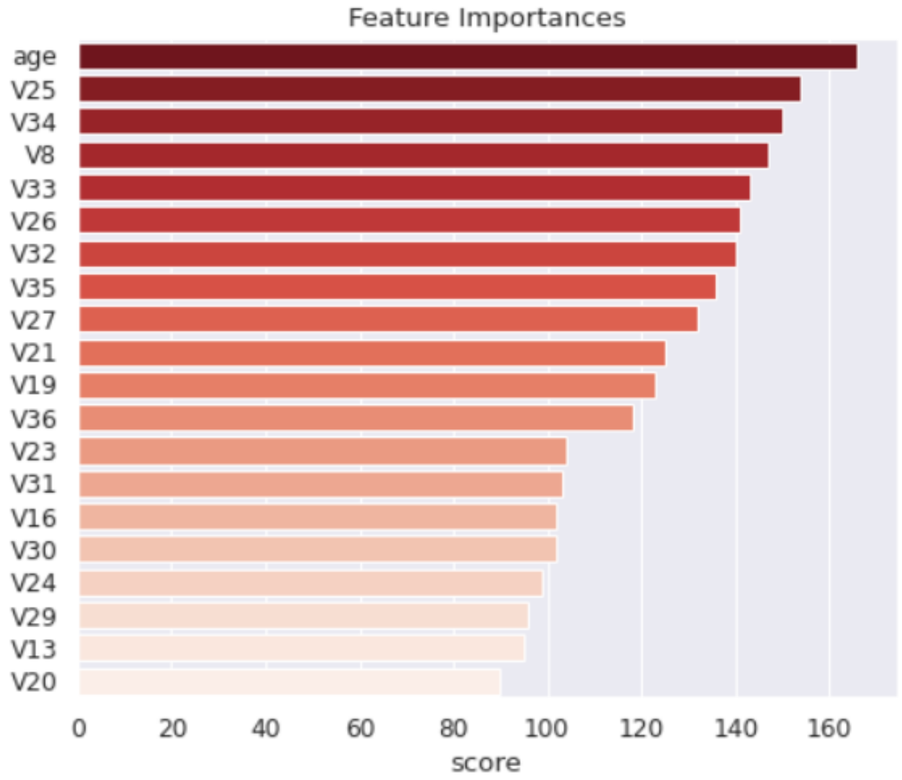
이미지 사이즈와 숫자와 관련된 column들 모두 drop

2. Adversarial Validation

해당 노트에서의 Adversarial Validation 결과



AUC가 0.5에 가까워진 것을 확인할 수 있다



=> 해당 column들을 drop한 train, test data로 마저 훈련 진행

2. Adversarial Validation

아쉬운 점

```
# setting model hyperparameters, didn't include fine tuning here because of timing reasons...

xg = xgb.XGBClassifier(
    n_estimators=750,
    min_child_weight=0.81,
    learning_rate=0.025,
    max_depth=2,
    subsample=0.80,
    colsample_bytree=0.42,
    gamma=0.10,
    random_state=42,
    n_jobs=-1,
)
```

하이퍼 파라미터 튜닝을 제외해
익숙한 파라미터들 (n_estimators, learning_rate,
max_depth, gamma 등)을 제외한 나머지들이 어쩌다가
튜닝된건지 그러한 과정을 몰라 아쉬움.