MIRIPLANTS

유전체 정보 품종 분류 AI 경진대회

01 / 개발환경

· Version 명시

02 / EDA

· Target 분포 확인

03 / FEATURE ENGINEERING & PREPROCESSING

· SNP 합
· A,C,G 값 개수
· trait column 문자열 변환

04 / SCALING & ENCODING & OVERSAMPLING

· StandardScaler
· CatboostEncoder
· BorderlineSMOTE

05/ MODELING

· VotingClassfier

06/ SUBMIT

·Submit

Google Colab

Sklearn 버전 : 1.0.2

Pandas 버전 : 1.3.5

Numpy 버전 : 1.21.6

런타임 유형 : GPU

Linux-5.10.147+-x86_64-with-glibc2.27

Ubuntu 18.04.6 LTS

Python 3.8.16

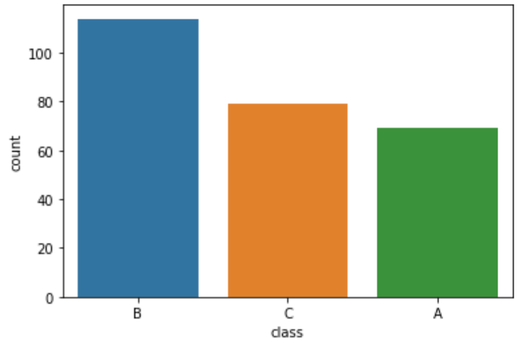
GPU : NVIDIA-SMI 460.32.03 Driver Version: 460.32.03 CUDA Version: 11.2 Tesla T4

EDA

```
[ ]  1 #X Y 데이터분리 및 id column drop
     2 def get_x_y(df):
     3     if 'class' in df.columns:
     4         df_x = df.drop(columns=['id', 'class'])
     5         df_y = df['class']
     6         return df_x, df_y
     7     else:
     8         df_x = df.drop(columns=['id'])
     9         return df_x
    10
    11 train_x, train_y = get_x_y(train)
    12 test_x = get_x_y(test)
```

```
[ ]  1 #class imbalanced -> Oversampling 적용
     2 sns.countplot(train_y)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fce0608c730>
```



[Target 데이터 분포 확인]

- Target A,B,C의 분포가 불균형한 것을 확인할 수 있음

1. Data Pre-processing

```
[ ]  1 #train_x 데이터를 확인해본 결과, 동일한 값을 가지는 columns 제거 -> 모델링에 좋지 않은 영향을 줄 것으로 판단
     2 train_x = train_x.drop(columns=['father','mother','gender'])
     3 test_x = test_x.drop(columns=['father','mother','gender'])
```

2. trait 변수 type 변환

```
[ ]  1 train_x['trait'] = train_x['trait'].astype('object')
     2 test_x['trait'] = test_x['trait'].astype('object')
```

- father, mother, gender 피처의 경우는 nunique 값이 1이여서 drop을 진행
- trait 피처는 수치형이지만 범주형 속성을 띄고있어서 문자형으로 변환

SNP 합

```
[ ]  1 train_x['2_BTA'] = train_x['SNP_01']
     2 train_x['chrom_6'] = train_x['SNP_02'] + '-'+ train_x['SNP_03'] + '-' + train_x['SNP_04'] + '-' + train_x['SNP_05'] + '-' + train_x['SNP_06'] + '-' + train_x['SNP_07'] + '-' + train_x['SNP_08'] + '-' + train_x['SNP_09']
     3 train_x['6_ARS_Parent'] = train_x['SNP_02']
     4 train_x['6_ARS_BFGL'] = train_x['SNP_03'] + '-' + train_x['SNP_04'] + '-' + train_x['SNP_09']
     5 train_x['6_BOVINE'] = train_x['SNP_05'] + '-' + train_x['SNP_06'] + '-' + train_x['SNP_08']
     6 train_x['6_HAPMAP'] = train_x['SNP_07']
     7 train_x['7_BTB'] = train_x['SNP_10']
     8 train_x['8_ARS'] = train_x['SNP_11']
     9 train_x['chrom_9'] = train_x['SNP_12'] + '-' + train_x['SNP_13'] + '-' + train_x['SNP_14']
    10 train_x['9_HAPMAP'] = train_x['SNP_12'] + '-' + train_x['SNP_14']
    11 train_x['9_BTB'] = train_x['SNP_13']
    12 train_x['10_BOVINE'] = train_x['SNP_15']
    13 train_x['SNP_total'] = train_x['SNP_01'] + '-' + train_x['SNP_02'] + '-' + train_x['SNP_03'] + '-' + train_x['SNP_04'] + '-' + train_x['SNP_05'] + '-' + train_x['SNP_06'] + '-' + train_x['SNP_07'] + '-' + train_x['SNP_08'] + '-' + train_x['SNP_09'] + '-
```

```
[ ]  1 test_x['2_BTA'] = test_x['SNP_01']
     2 test_x['chrom_6'] = test_x['SNP_02'] + '-'+ test_x['SNP_03'] + '-' + test_x['SNP_04'] + '-' + test_x['SNP_05'] + '-' + test_x['SNP_06'] + '-' + test_x['SNP_07'] + '-' + test_x['SNP_08'] + '-' + test_x['SNP_09']
     3 test_x['6_ARS_Parent'] = test_x['SNP_02']
     4 test_x['6_ARS_BFGL'] = test_x['SNP_03'] + '-' + test_x['SNP_04'] + '-' + test_x['SNP_09']
     5 test_x['6_BOVINE'] = test_x['SNP_05'] + '-' + test_x['SNP_06'] + '-' + test_x['SNP_08']
     6 test_x['6_HAPMAP'] = test_x['SNP_07']
     7 test_x['7_BTB'] = test_x['SNP_10']
     8 test_x['8_ARS'] = test_x['SNP_11']
     9 test_x['chrom_9'] = test_x['SNP_12'] + '-' + test_x['SNP_13'] + '-' + test_x['SNP_14']
    10 test_x['9_HAPMAP'] = test_x['SNP_12'] + '-' + test_x['SNP_14']
    11 test_x['9_BTB'] = test_x['SNP_13']
    12 test_x['10_BOVINE'] = test_x['SNP_15']
    13 test_x['SNP_total'] = test_x['SNP_01'] + '-' + test_x['SNP_02'] + '-' + test_x['SNP_03'] + '-' + test_x['SNP_04'] + '-' + test_x['SNP_05'] + '-' + test_x['SNP_06'] + '-' + test_x['SNP_07'] + '-' + test_x['SNP_08'] + '-' + test_x['SNP_09'] + '-' + test_x
```

- SNP_info 정보를 활용하여 비슷한 특성이 있는 SNP끼리 결합

EX) train_x['chrom_6'] - 6번 chrom에 있는 모든 SNP
EX) train_x['6_ARS_Parent'] - 6번 chrom 중 이름이 ARS_Parent 로 시작

SNP A,C,G 값 개수

```
[ ]  1 train_x['concat'] = train_x.iloc[:,1:16].sum(axis=1).apply(lambda x : x.replace(" ",""))
     2 train_x['numGC'] = train_x['concat'].apply(lambda x : x.count('C')+x.count('G'))
     3 train_x['numA'] = train_x['concat'].apply(lambda x : x.count('A'))
     4 train_x['numGC^2'] = train_x['numGC']**2
     5 train_x['sub'] = train_x['numGC'] - train_x['numA']
     6 train_x['H'] = train_x['numGC']*3 + train_x['numA']*2
```

```
[ ]  1 test_x['concat'] = test_x.iloc[:,1:16].sum(axis=1).apply(lambda x : x.replace(" ",""))
     2 test_x['numGC'] = test_x['concat'].apply(lambda x : x.count('C')+x.count('G'))
     3 test_x['numA'] = test_x['concat'].apply(lambda x : x.count('A'))
     4 test_x['numGC^2'] = test_x['numGC']**2
     5 test_x['sub'] = test_x['numGC'] - test_x['numA']
     6 test_x['H'] = test_x['numGC']*3 + test_x['numA']*2
```

- 구글링하여 관련 배경지식을 활용하여 파생변수 생성

EX) train_x['numGC'] , train_x['numA'] 은 각각 GC의 개수와, A의 개수
EX) train_x['numGC^2'] 는 train_x['numGC'] 의 제곱
EX) train_x['sub'] 는 GC 의 개수와 A 의 개수의 차
EX) train_x['H'] 는 GC 의 개수 * 3 + A의 개수 * 2

출처 : https://ko.wikipedia.org/wiki/GC_%ED%95%A8%EB%9F%89

측정 [편집]

GC 함량은 일반적으로 백분율 값으로 표현되지만, GC 비율(GC ratio)로도 표시된다. GC 함량 백분율은 다음과 같이 계산한다.[5]

$$\frac{G+C}{A+T+G+C} \times 100\%$$

A-T/G-C 비율은 다음과 같이 계산한다.[6]

$$\frac{A+T}{G+C}.$$

GC 비율뿐만 아니라 GC 함량은 분광광도법을 사용하여 측정할 수 있다. DNA 이중 나선의 용융 온도를 측정함으로써 알 수 있다. 260nm의 파장에서 DNA의 흡광도는 이중 가닥이 단일 가닥으로 분리될 때 급격히 증가한다.[7] GC 비율을 결정하기 위해 가장 일반적으로 사용되는 프로토콜은 유세포 분석을 사용한다.[8]

Scaling

```
[ ]  1 scaler = StandardScaler()
     2 train_x[num_features] = scaler.fit_transform(train_x[num_features])
     3 test_x[num_features] = scaler.transform(test_x[num_features])
```

Encoding

```
[ ]  1 def catboost_encoder_multiclass(X,X_t,y):
     2     y = y.astype(str)
     3     enc = ce.OneHotEncoder().fit(y)
     4     y_onehot = enc.transform(y)
     5     class_names = y_onehot.columns
     6     X_obj = X.select_dtypes('object')
     7     X_t_obj = X_t.select_dtypes('object')
     8     X = X.select_dtypes(exclude='object')
     9     X_t = X_t.select_dtypes(exclude='object')
    10     for class_ in class_names:
    11         enc = ce.CatBoostEncoder()
    12         enc.fit(X_obj,y_onehot[class_])
    13         temp = enc.transform(X_obj)
    14         temp_t = enc.transform(X_t_obj)
    15         temp.columns = [str(x)+'_'+str(class_) for x in temp.columns]
    16         temp_t.columns = [str(x)+'_'+str(class_) for x in temp_t.columns]
    17         X = pd.concat([X,temp],axis=1)
    18         X_t = pd.concat([X_t,temp_t],axis=1)
    19
    20     return X, X_t
    21
    22 train_x, test_x = catboost_encoder_multiclass(train_x,test_x,train_y)
```

SMOTE

```
[ ]  1 # Class 불균형 문제 해결
     2 train_x,train_y = BorderlineSMOTE(random_state=CFG.SEED).fit_resample(train_x,train_y)
```

Scaling

- Scaler중에서 성능이 좋다고 알려진 StandardSclaer와 MinMaxScaler중에서 분류의 경우는 StandardScaler의 성능이 일반적으로 더 좋아서 채택

Encoding

- Encoding의 경우도 OneHotEncoding, LableEncoding, OrdinaryEncoding, TargetEncoding 등등을 사용해 봤지만 CatboostEncoder가 성능이 가장 좋았음

OverSampling

- 클래스 불균형 해결을 위해 BorderlineSMOTE를 사용하여 OverSampling

Modeling & Ensemble

```
[ ]  1 #Submission file 준비
     2 submit = pd.read_csv(DATA_PATH  + 'sample_submission.csv')
```

```
[ ]  1 #Model Selection -> 여러 모델링 실험결과 종류가 다른 모델 여러개를 앙상블 하는 것이 좋다 판단함
     2 models = [
     3     ('bag', BaggingClassifier(random_state=CFG.SEED)),
     4     ('dt', DecisionTreeClassifier(random_state=CFG.SEED)),
     5     ('rc', RidgeClassifier(random_state=CFG.SEED)),
     6     ('xgb', XGBClassifier(random_state=CFG.SEED)),
     7     ('lgb', LGBMClassifier(random_state=CFG.SEED)),
     8     ('gb', GradientBoostingClassifier(random_state=CFG.SEED)),
     9     ('svc', SVC(random_state=CFG.SEED)),
    10     ('rcc', RidgeClassifierCV()),
    11     ('rf', RandomForestClassifier(random_state=CFG.SEED))
    12 ]
```

```
[ ]  1 #최종모델은 Votingclassifier 사용하여 ensemble -> 제출결과 public score기준 XGBClassifier와 RandomForestClassifier 성능이 좋아 가중치를 주었음
     2 best_model  = VotingClassifier(models, voting='hard', weights=[1,1,1,2,1,1,1,1,2])
     3 best_model.fit(train_x,train_y)
```

- Sklearn 에 내장된 모델 및 Boosting계열 모델들을 선별하여 VotingClassfier을 사용하여 HardVoting 앙상블 진행

Submit

```
[ ]  1 #test predict
     2 pred = class_le.inverse_transform(best_model.predict(test_x))
     3 submit['class'] = pred
```

```
[ ]  1 submit
```

	id	class
0	TEST_000	A
1	TEST_001	B
2	TEST_002	C
3	TEST_003	C
4	TEST_004	A
...	...	...
170	TEST_170	B
171	TEST_171	C
172	TEST_172	C
173	TEST_173	B
174	TEST_174	B

175 rows × 2 columns

```
[ ]  1 submit.to_csv('Fine_20.csv',index=False)
```

- Inverse_transform을 사용하여 LabelEncoding한 Target을 다시 기존 형식으로 변환하였음


감사합니다!!