[2022 딥러닝/클라우드 경진대회 보고서]

32200327 김경민

1. 데이터 분석

① Boxplot 확인

- train 데이터 feature별 boxplot 확인
- train 데이터의 feature별, label별 boxplot 확인
- test 데이터의 feature별 boxplot 확인 (추후 테스트데이터 예측값 제출 과정에서 train 데이터와 test 데이터 간의 차이가 있는 것으로 추정되어 추가로 확인을 진행함)

⇨ train 데이터 label 간에 boxplot 차이가 B label를 제외하고는 거의 비슷함

⇨ 이상치가 전혀 없는 feature도 있었고, 주로 C, F label에서 이상치 빈번히 발생

⇨ Train, Test 간 boxplot 차이 또한 크지 않았으나 전체적으로 train 데이터값들이 test 데이터 값보다 큰 것으로 관찰

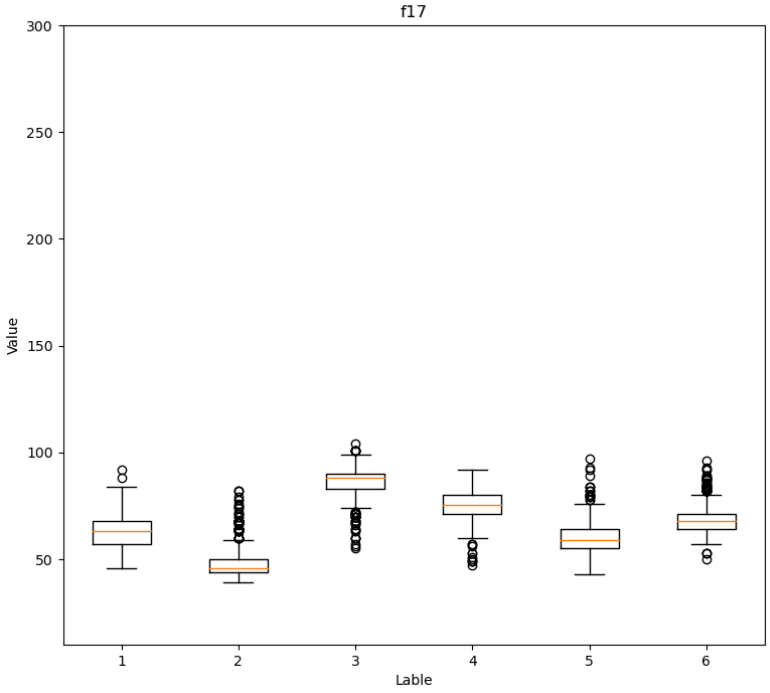
```
def boxplot(data):
    import matplotlib.pyplot as plt
    data =
pd.read_csv("C://Users//DKU//Desktop//경진대회_dataset_2022//te
st_open.csv")

    # 1. 기본 스타일 설정
    plt.style.use('default')
    plt.rcParams['figure.figsize'] = (9, 8)
    plt.rcParams['font.size'] = 10

    # 2. 데이터 준비
    '''
    ## label 별 boxplot 그리기
    for i in range(len(data.columns)-2):
```

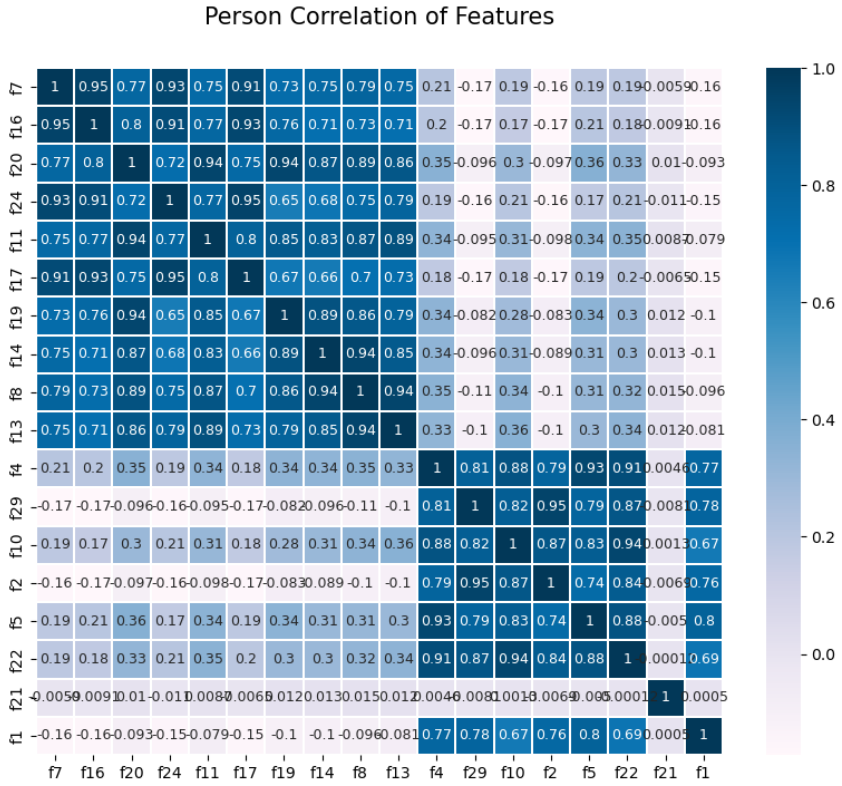
```
        data_a = []
        data_b = []
        data_c = []
        data_d = []
        data_e = []
        data_f = []
        for j in range(len(data)):
            label = data["real_label"].loc[j]
            if (label == 0.0):
                data_a.append(data.iloc[j,i])
            elif (label == 1.0):
                data_b.append(data.iloc[j,i])
            elif (label == 2.0):
                data_c.append(data.iloc[j,i])
            elif (label == 3.0):
                data_d.append(data.iloc[j,i])
            elif (label == 4.0):
                data_e.append(data.iloc[j,i])
            elif (label == 5.0):
                data_f.append(data.iloc[j,i])
        '''
    ## 전체 boxplot 그리기
    for i in range(len(data.columns)):
        # 3. 그래프 그리기
        fig, ax = plt.subplots()
        ax.boxplot([data.iloc[:,i]])
        if (data.iloc[0, i] < 3.0):
            ax.set_ylim(0.0, 2.0)
        elif (data.iloc[0, i] >= 3.0):
            ax.set_ylim(10, 300)
        ax.set_xlabel('Lable')
        ax.set_ylabel('Value')
        plt.title("f" + str(data.columns[i]))

#plt.savefig(f'C://Users//DKU//Desktop//딥러닝_경진_boxplot//tes
t_전체//f{data.columns[i]}.png')
```



[train 데이터의 feature별, label별 boxplot 예시 사진]

② 상관관계 분석



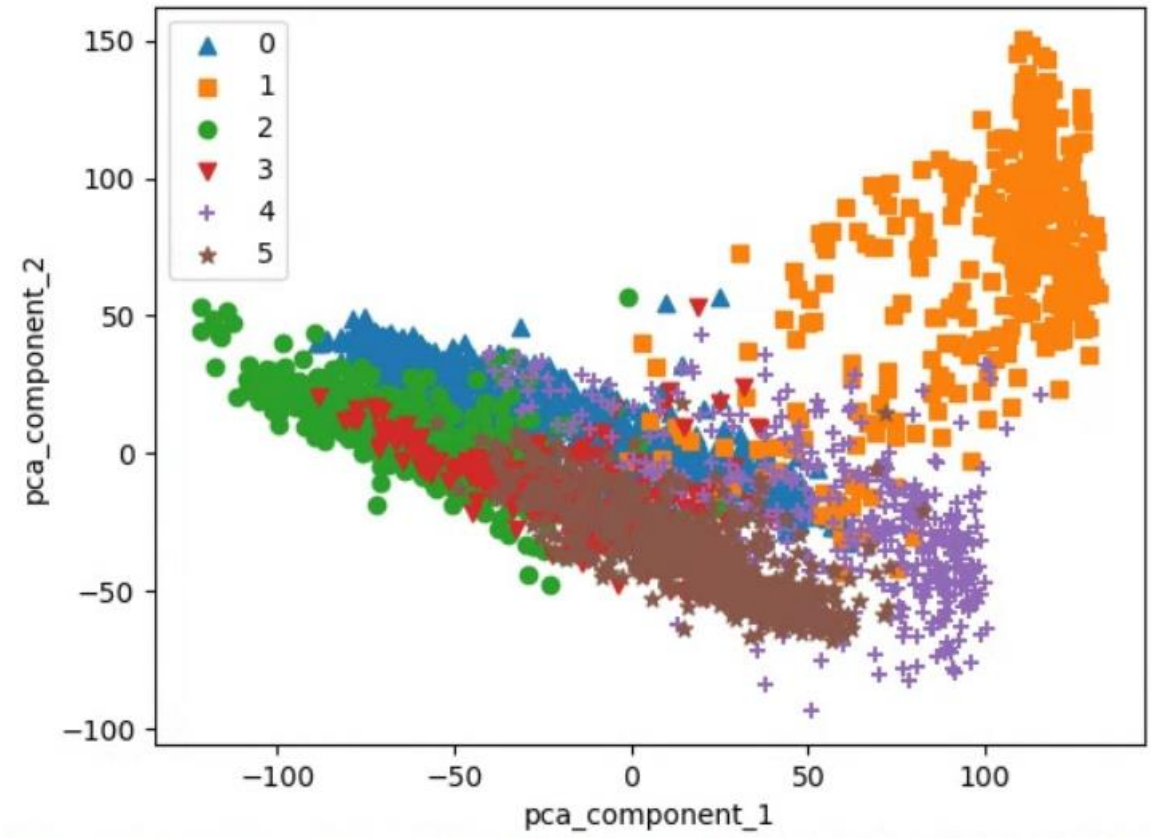
- 각 feature들 사이의 피어슨 상관관계를 확인

- f7-16/f7-f16/f17-f24/f29-f2 에서 95%이상의 상관관계 발견
- f7,f16,f20,f24,f11,f17,f19,f14,f8,f13 끼리의 조합, f4,f29,f10,f2,f5,f22 끼리의 조합이에서 상관관계 높음
- f21 은 전체적으로 상관관계 낮으므로 제외

```
#상관관계 분석
def Corr_data(data):
    import seaborn as sns
    colormap = plt.cm.PuBu
    plt.figure(figsize=(10, 8))
    plt.title("Person Correlation of Features", y=1.05,
size=15)
    sns.heatmap(data.astype(float).corr(), linewidths=0.1,
vmax=1.0,
                square=True, cmap=colormap,
linecolor="white", annot=True, annot_kws={"size": 9})
    plt.show()
```

③ 차원축소 산점도 확인

- feauture 들을 2 개의 feature 로 차원축소 후 2 차원 그래프에 label 별 산점도를 그림



⇨ boxplot 에서 확인했듯이 label B 는 확실하게 구분되어 있고 A,C,D,F 는 많은 데이터가 겹쳐져서 분포

```
def pca(data):
    from sklearn.decomposition import PCA

    pca = PCA(n_components=3)

    # fit( )과 transform( ) 을 호출하여 PCA 변환 데이터 반환
    pca.fit(data)
    data_pca = pca.transform(data)
    print(data_pca.shape)

    pca_columns = ['pca_component_1', 'pca_component_2']
    dataDF_pca = pd.DataFrame(data_pca,
columns=pca_columns)
    dataDF_pca['target'] = data.real_label
    print(dataDF_pca.head(6))

    markers = ['^', 's', 'o', 'v', '+', '*']
```

```
    # pca_component_1 을 x 축, pc_component_2 를 y 축으로
scatter plot 수행.
    for i, marker in enumerate(markers):
        print(i)
        x_axis_data = dataDF_pca[dataDF_pca['target'] ==
i]['pca_component_1']
        y_axis_data = dataDF_pca[dataDF_pca['target'] ==
i]['pca_component_2']
        plt.scatter(x_axis_data, y_axis_data,
marker=marker, label=i)

    plt.legend()
    plt.xlabel('pca_component_1')
    plt.ylabel('pca_component_2')
    plt.show()
```

2. 기본 모델 성능 비교

① 수업자료 및 개인공부에 사용했던 모델 사용

- 지금까지 수업시간 실습에 사용했던 모델과 개인공부를 하며 사용해봤던 모델(파라미터 포함)들을 사용해 전처리는 진행하지 않은 상태에서 성능 확인

- KNN/DecisionTree/SVM/Random Forest/ AdaBoos/ CatBoost/Xgboost/DNN/LGBM/ GradientBoost/Bagging/Stacking/Voting(가장 마지막에 확인)

② 1차 모델 선택 과정 – Stacking Model(LGBM 모델)

- ①의 과정을 통해 LGBM 모델에서 다음과 같은 조합의 파라미터에서 꽤 좋은 성능을 보인다고 판단(train 데이터 예측결과 90.2%, test데이터 예측결과 89%)

```
LGBMClassifier(boosting_type='dart',n_estimators=1000,
num_leaves=64, random_state=123)
```

- Stacking 모델에서 LGBM 모델을 final 모델로 설정하고 base 모델 4가지를 다음과 같이 설정했을 때 train 데이터에서 조금 더 나은 성능을 확인(train 데이터 예측결과 90.98%)

```
estimators = [('rf',
RandomForestClassifier(n_estimators=300,
random_state=123)),  # 90.98
            ('svm', svm.SVC(kernel='rbf',
random_state=123)),
            ('lr', LogisticRegression(max_iter=500,
random_state=123)),
            ('ridge', RidgeClassifier(random_state=123))]
```

- Stacking 모델이 가장 좋은 모델이라고 생각하여 해당 모델을 기본 모델로 선택하고 base 모델을 아래와 같이 더 추가하여 train 데이터 예측결과 91% 까지 성능 향상을 확인했으나 실제 test 데이터 예측결과 88%로 대폭 하락

```
estimators4 = [('rf',
RandomForestClassifier(n_estimators=300, max_depth=5,
criterion='gini', random_state=123)),
             # 90.98  #
             ('svm', svm.SVC(kernel='rbf',
random_state=123)),
             ('lr', LogisticRegression(max_iter=500,
random_state=123)),
             ('ridge',
RidgeClassifier(random_state=123)),
             ('etc',
ExtraTreesClassifier(n_estimators=1900, criterion='gini',
random_state=123)),
             # hyper parameter tuning
             ('dt', DecisionTreeClassifier(max_depth=5,
criterion='gini', random_state=123)),
             ('gausi',
GaussianProcessClassifier(random_state=123))]
```

③ 2차 모델 선택 과정 – Xgboost basic Model

- Stacking 모델 선정에 문제가 있다고 판단하여 다시 단일 모델 확인한 결과 default 파라미터에서 xgboost 가 test 데이터 예측결과 90%라는 높은 성능 보임(데이터 전처리X)

⇨ stacking 모델에서는 데이터 전처리, feature selection, 파라미터 조정 등을 통해 train데이터의 성능이 눈에 띄게 향상되었지만 결정적으로 test 데이터에는 반영되지 못하고 오히려 train 데이터 결과와 test 데이터 결과의

차이가 4~5% 로 벌어졌음 (xgboost 모델에서는 1~2%)

⇨ base 모델이 추가되면서 모델 자체가 달라질 수 있다는 것을 간과

⇨ 최종 선정 모델은 xgboost

⇨ 모든 모델 평가는 cross_val_score()를 사용해 cv=5로 설정하고 평균 정확도로 판단

3. 데이터 전처리

① 데이터 정규화 및 표준화

- MinMaxScaler() 를 사용해 데이터 정규화
- StandardScaler() 를 사용해 데이터 표준화 진행

⇨ 성능 차이 없거나 더 떨어짐

```
# 표준화
data_test[:] =
StandardScaler().fit_transform(data_test[:])
# 정규화
sacler = MinMaxScaler()
data_test[:] = sacler.fit_transform(data_test[:])
```

② 이상치 제거 및 조정

- 이상치를 아예 제거하거나 feature별 boxplot의 중앙값과 이상치값의 차이를 더하거나 빼서 중앙값에 가까워지도록 조정
- 차이값을 2~10으로 나눠서 다양하게 미세 조정해봤으나 성능 향상 X

③ 소수점 아래 n 자리 반올림(n=1,2,3)

- Stacking 모델에서는 이 방법으로 0.2% 성능 향상 되었으나 xgboost 모델에서는 향상 X

4. Feature Selection

① filter method 사용

- filter method를 통해 얻어낸 feature 조합
- selection을 진행 과정에서 한 feature가 추가될 때 갑자기 성능이 떨어지거나 다시 갑자기 좋아지는 경우, 해당 feature를 조합에 추가하거나 제거
- 최종 18개의 feature 선정 ('f7', 'f16', 'f20', 'f24', 'f11', 'f17', 'f19', 'f14', 'f8', 'f13', 'f4', 'f29', 'f10', 'f2', 'f5', 'f22', 'f1','f3)
- 상관관계 분석에서 좋지 않았던 feature들이 포함된 경우 제거해보며 성능 확인해봤지만 그대로 두는게 가장 좋았음
- backward는 별로 성능이 좋지 않았고, forward는 오류가 나서 시도해보지 못함

⇨ xgboost deafult 모델에서 test 데이터 예측결과로 89.8%을 반올림하여 90% 성능을 냈다면 feature selection을 통해 90.002 성능 확인 (최고 성능)

```
def Filter_Method(df_X, df_y):
    test = SelectKBest(score_func=chi2, k=df_X.shape[1])
    fit = test.fit(df_X, df_y)

    # summarize evaluation scores
    print(np.round(fit.scores_, 3))
    f_order = np.argsort(-fit.scores_)
    # sort index by decreasing order
    sorted_columns = df_X.columns[f_order]

    return sorted_columns
```

5. 파라미터 튜닝

① xgboost model hyper parameter

- gridsearch 를 사용해 파라미터 튜닝을 진행했지만 n_estimators = 48 에서 train 데이터 예측결과가 0.2%가량 좋아짐

⇨ 하지만 test 데이터 예측결과 성능은 좋아지지 않음

⇨ 파라미터 튜닝 효과가 별로 없음

6. 오분류 데이터 분석

① 오분류 데이터의 feature/실제값/예측값 파악

- 어떤 label에서 오분류가 많이 일어나는지 파악
- 주로 lable 3(D),5(F),4(E) 에서 오분류 많이 발생
- 3(D)는 2(C)또는 5(F)로 오분류, 5(F)는 3(D)으로 오분류, 4(E)는 0(A) 또는 5(F)으로 오분류 하는 경우가 다수
- 이전에 제출용 csv 파일에 문제가 있는지 확인하기 위해 csv 파일 몇개를 비교해본 적이 있는데 그 당시에도 F 나 D 클래스에 변화가 많았던 것으로 기억(test 데이터를 사용한 예측값에도 비슷한 문제가 있다고 예측할 수 있음)

② 오분류 데이터의 feature를 넣어 차원축소 산점도 그림

6. 합성 데이터 추가

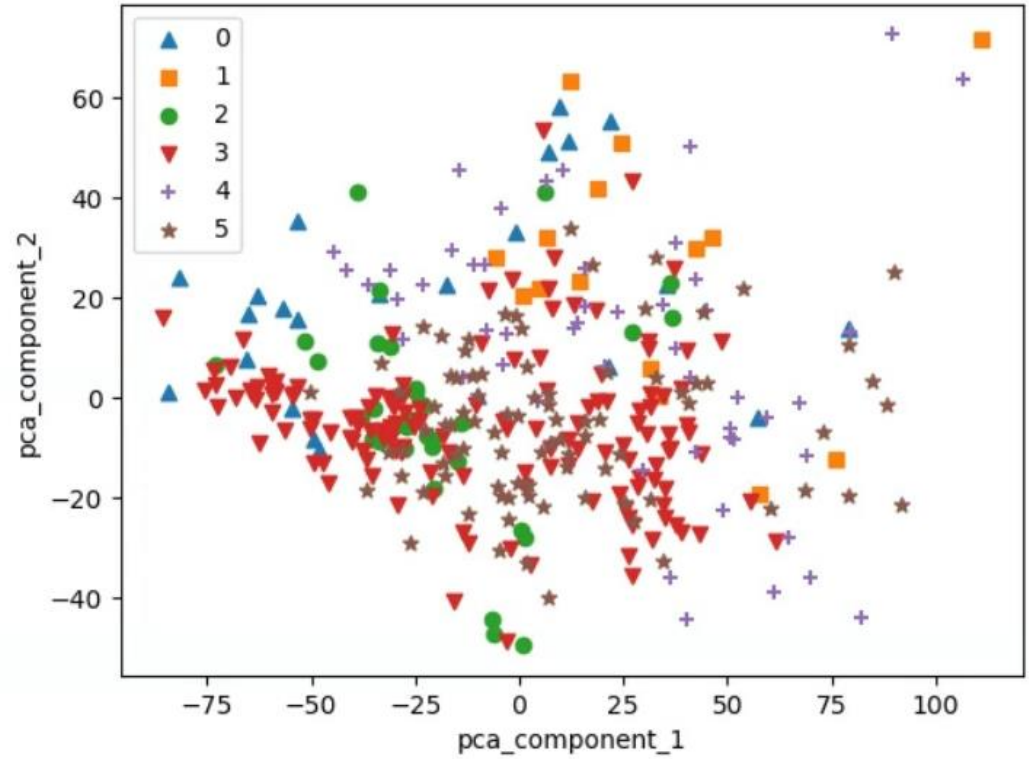
① 상관관계 높은 feature 합성

- 기존의 feature만으로는 모델의 성능을 높이기 힘들 것이라고 판단하여 상관관계가 가장 높은 f7과f16 합성하여 새로운 feature를 만들어 추가
- 두 값을 빼고 절대값을 새로운 feature로 추가

⇨ 성능 그대로

⇨ 하지만 두 차이값이 0~15 정도의 값으로 이 값을 사용해 weight를 주는

등의 아이디어를 사용해볼 수 있지 않을까 생각함(시간 관계상 더 이상 진행 X)



- 오분류 데이터의 규칙을 찾아보려고 했으나 데이터끼리 겹쳐 있는 부분이 너무 많아 군집화나 미세 조정이 힘듬
- 예를 들어 2라고 예측을 했는데 산점도가 x축이 -25이하, y축이 10(8)이하면 3으로 재분류 시도해보았으나 성능 차이 없음

7. Voting 모델

① '최고 성능 모델 + Default 파라미터 기본 모델' 로 구성된 voting 모델

- Voting 모델을 구성하는 각 모델의 성능을 최대한 높인 후 진행해야 하는데 주 모델인 xgboost만 튜닝 후 나머지는 기본 모델을 사용
- Xgboost 외 다른 모델들을 파라미터 튜닝하면서 train 데이터 예측결과

90.98%까지 성능 향상

- 하지만 실제 test 데이터 예측결과는 향상 X

```
## Voting Model
models = [
    ('knn', KNeighborsClassifier(n_neighbors=7)),
    ('xgb',
     XGBClassifier(verbosity=0, n_estimators=48,
learning_rate=0.3, use_label_encoder=False,
objective='multi:softmax',
                   random_state=123)),
    ('rfc', RandomForestClassifier()),  # n_estimators=300
    ('lgbm', LGBMClassifier(n_estimators=1000,
num_leaves=64)),  # n_estimators=1900
    ('etc', ExtraTreesClassifier(n_estimators=1900)),
    ('gbc', GradientBoostingClassifier())  #
n_estimators=200
]


clf_voting = VotingClassifier(estimators=models,
voting='soft')
voting_scores = cross_val_score(clf_voting, X, Y, cv=5)
print('Voting accuracy', np.mean(voting_scores))
```

8. Get_Test 코드

① 최종 결과 도출에 사용된 코드

```
def Get_Test(train_X, train_y):
    data_test =
pd.read_csv("C://Users//DKU//Desktop//경진대회_dataset_2022//tes
t_open.csv")

    data_test = data_test.loc[:,
              ['f7', 'f16', 'f20', 'f24', 'f11', 'f17', 'f19',
'f14', 'f8', 'f13', 'f4', 'f29', 'f10', 'f2', 'f5',
               'f22', 'f1', 'f3']]  # test 기준 이 조합에서
최고(파라미터 튜닝도 믿을게 x)
    # 이상치 조정
```

```
    for i in range(len(data_test.columns)):
        q1, q3 = np.percentile(data_test.iloc[:, i], [25, 75])
        iqr = q3 - q1
        lower_bound = q1 - (iqr * 1.5)
        uppper_bound = q3 + (iqr * 1.5)
        for j in range(len(data_test)):
            if ((data_test.iloc[j, i] > uppper_bound) |
(data_test.iloc[j, i] < lower_bound)):
                value = (data_test.iloc[j, i] -
np.percentile(data_test.iloc[:, i], [50])) / 10  # 10
                print(value)
                if (value < 0):
                    data_test.iloc[j, i] = data_test.iloc[j, i] +
abs(value)
                else:
                    data_test.iloc[j, i] = data_test.iloc[j, i] -
value

    model = XGBClassifier(verbosity=0, n_estimators=48,
max_depth=6, use_label_encoder=False, gamma=0,
                          min_child_weight=1,
objective='binary:logistic', random_state=123)
    model.fit(train_X, train_y)
    pred = model.predict(data_test)

    pred_real = []
    for i in range(len(pred)):
        if (pred[i] == 0):
            pred_real.append('A')
        elif (pred[i] == 1):
            pred_real.append('B')
        elif (pred[i] == 2):
            pred_real.append('C')
        elif (pred[i] == 3):
            pred_real.append('D')
        elif (pred[i] == 4):
            pred_real.append('E')
        elif (pred[i] == 5):
            pred_real.append('F')

    print(pred_real)
    with
open('C://Users//DKU//Desktop//경진대회_answer//32200327_김경민.c
sv', 'w', newline='') as f:
        writer = csv.writer(f)
        writer.writerows(pred_real)


def main():
```

```
    data_raw =
pd.read_csv("C://Users//DKU//Desktop//경진대회_dataset_2022//tra
in_open.csv")

    # data_feature = data_raw.drop(columns = ['lable'])
    data_feature = data_raw.loc[:,
                   ['f7', 'f16', 'f20', 'f24', 'f11', 'f17',
'f19', 'f14', 'f8', 'f13', 'f4', 'f29', 'f10', 'f2', 'f5',
                    'f22', 'f1',
                    'f3']]  # #'f7', 'f16', 'f20', 'f24', 'f11',
'f17', 'f19', 'f14', 'f8', 'f13', 'f4', 'f29', 'f10', 'f2',
'f5', 'f22', 'f1','f3'
    data_lable = data_raw.loc[:, "lable"]

    ##get test##
    Get_Test(data_feature, data_lable)


if __name__ == "__main__":
    main()
```

9. 소감

사용해볼 수 있는 모델이 많다는 것에 너무 의지하여 초반에 데이터 분석을 소홀히 한 것 같다. 데이터의 분포나 특징, 편향 등을 더 가시적으로 확인할 수 있도록 시각화하고 분석에 더 시간을 투자 했어야 했다고 생각한다. 특히 train 데이터의 label C,D,F 가 서로 비슷한 분포를 띄고 있는데 이것을 빨리 파악해내고 더 잘 분류할 수 있는 방법을 찾아봤어야 했다는 아쉬움이 남았다.

또한 처음부터 파라미터 튜닝이나 feature selection을 전혀 하지 않은 xgboost 기본 모델로 90%에 달하는 성능이 나오고 다른 학생들과 소수점 차이로 성능 향상을 높이기 위해 고군분투해야 하는 과정이 힘들고 다소 지루했던 것 같다. 특히 챕터 2에서 stacking 모델로 train 데이터 성능을 꽤 높여서 기쁜 마음으로 test 데이터 예측결과를 확인해 봤는데 갑자기 성능이 확 떨어져서 너무 당황했던 것 같다. 모델이 달라지면 train과 test 사이의 성능 차이도 있을 수 있다는 것을 간과하고 있었던 것이 매우 아쉽고, 이로 인해 많은 기회를 날린 것도 아깝게

생각한다. 하지만 이런 과정을 통해 다시 한 번 데이터 분석과 모델 개발에 있어서 기초를 다질 수 있었고 앞으로 다른 경진대회에 부담 없이 도전해볼 수 있겠다는 자신이 생겼다. 또한 경진대회를 통해 모델과 관련된 공식 문서를 찾아보기도 하고 다른 사람들은 어떤 모델을 사용했고 어떻게 최고 성능을 냈는지 찾아보면서 많이 배워갈 수 있었던 것 같고 성능이 오르지 않아 마음이 조급했지만 향상을 위해 새로운 아이디어가 떠오르거나 시도를 해볼 때의 즐거움도 컸던 것 같다.