# Multivariate Data Analysis Assignment #4

## Ensemble

**산업경영공학부 2020170831 민찬홍**

### [Q0] 사전 작업 사항

[1] 입력 변수의 속성이 numeric 이 아닌 변수들에 대해 1-of-C coding (1-hot encoding) 방식을 통해 명목형(요인형) 변수를 범주의 개수만큼의 이진형(binary) 변수들로 구성되는 dummy variable 을 생성하시오.

본 데이터셋은 'Earthquake_Damage dataset'으로 종속변수인 'damage_grade'와 독립변수 38개, building id를 합한 총 40개의 column으로 구성되어 있으며 데이터의 개수는 260601개로 이루어져있다.

명목형 변수를 제외한 변수들에 대해 Scaling작업을 수행하였고, building_id의 경우 본 데이터셋에서 단순 건물을 식별하는 식별자 역할 밖에 하지 않아 제외시켰다. 명목형 변수에 대해서는 아래 코드를 적용하여 범주의 개수만큼의 이진형(binary) 변수들로 구성되는 dummy variable를 생성하였다. 특별히 밀집행렬을 이용하여 모든 요소를 저장하였다.

```
from sklearn.preprocessing import OneHotEncoder

independent_vars=earthquake_df.drop(['building_id', 'damage_grade'], axis=1)
dependent_var=earthquake_df['damage_grade']

categorical_vars=independent_vars.select_dtypes(include=['object'])
ohe=OneHotEncoder(sparse_output=False) #밀집행렬로 생성
encoded_categorical_vars=pd.DataFrame(ohe.fit_transform(categorical_vars))
encoded_categorical_vars.columns=ohe.get_feature_names_out(categorical_vars.columns)
numerical_vars=independent_vars.select_dtypes(include=['int64', 'float64'])
encoded_vars=pd.concat([numerical_vars, encoded_categorical_vars, dependent_var], axis=1)
```

[2] 전체 데이터셋에서 10,000 개의 instance 를 샘플링한 뒤 학습/검증/테스트 데이터셋을 다시 6,000 개/2,000 개/2,000 개로 분할하고 다음 각 물음에 답하시오. 분류 성능을 평가/비교할 때는 3-class classification 의 Accuracy 와 Balanced Correction Rate (BCR)을 이용하시오.

앞서 [1] 문항에서 가공한 dataset에서 임의로 10000개의 instance를 샘플링을 하였다. 이후 sklearn의 train_test_split 함수를 통해 학습/검증/테스트 데이터셋을 다시 6,000 개/2,000 개/2,000 개로 분할하였다.

분류 성능을 평가/비교하기 위한 함수는 아래 코드와 같이 작성하였다.

```
#성능평가
from sklearn.metrics import accuracy_score

def calculate_matrix(y_true, y_pred):
  cm=confusion_matrix(y_true, y_pred)

  #Accuracy
  accuracy=accuracy_score(y_true, y_pred)

  #BCR
  BCRs=[]
  for i in range(cm.shape[0]):
    TP=cm[i,i]
    FN=cm[i,:].sum()-TP
    FP=cm[:,i].sum()-TP
    TN=cm.sum()-TP-FN-FP

    sensitivity=TP/(TP+FN) if TP+FN!=0 else 0
    specificity=TN/(TN+FP) if TN+FP!=0 else 0

    BCR=np.sqrt(sensitivity*specificity)
    BCRs.append(BCR)

  avg_BCR=np.mean(BCRs)

  return accuracy, avg_BCR
```

## [Q1] MLR, CART, ANN 분류 모델 구축

다음과 같이 세 가지 단일 모형에 대하여 분류 모델을 구축하고 Accuracy 와 BCR 관점에서 분류 정확도를 비교해보시오. CART 와 ANN 의 경우 hyperparameter 후보 값들을 명시하고 Validation dataset 을 통해서 최적의 값을 찾아서 Test 에 사용하시오.

우선 3개의 단일모델에 관한 초모수 값 후보를 다음과 같이 선정하였다.

```
#1.MLR
param_grid_logistic={'solver': ['lbfgs', 'sag'] ,'C':
[0.1,1,100] ,'max_iter': [100, 200, 500, 1000, 3000]}

#2. DT
param_grid_tree={'criterion': ['gini', 'entropy'] , 'max_depth': [None,
10, 20] , 'min_samples_split': [2, 10, 50]}

#ANN using MLPClassifier
param_grid_ann={'hidden_layer_sizes': [(50,), (100,)] ,'activation':
['relu', 'tanh', 'logistic'] , 'solver': ['adam', 'sgd', 'lbfgs'] ,
'early_stopping':[True]}
```

### 1-1.MLR

우선 MLR 모델의 최적의 하이퍼파라미터를 찾기 위해 아래 코드를 사용하였다.

```
#MLR 최적의 초모수 값 찾기
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import ParameterGrid

logistic_model=LogisticRegression(multi_class='multinomial', random_state=42)
logistic_model.fit(x_train, y_train)

best_bcr=0.0
best_model_logi=None

for params in ParameterGrid(param_grid_logistic):
  model=LogisticRegression(**params, random_state=42)
  model.fit(x_train, y_train)
  y_val_pred=model.predict(x_val)

  accuracy, bcr=calculate_matrix(y_val, y_val_pred)

  if bcr>best_bcr:
    best_bcr=bcr
    best_model_logi=model

print("Best Model Hyperparameters", best_model_logi.get_params())

y_test_pred=best_model_logi.predict(x_test)
accuracy_logistic, bcr_logistic=calculate_matrix(y_test, y_test_pred)
print("Test Accuracy:", accuracy_logistic, "Test BCR:", bcr_logistic)
```

앞서 설정한 하이퍼파라미터 값을 바꿔가며 검증 데이터셋에 대한 BCR값을 비교하며 최적의 MLR 모델을 찾았다. 이 때의 설정한 하이퍼파라미터 중 최적 값은 다음과 같다.

1. solver: [lbfgs] 2. C: [100] 3. Max_iter [100]

이 때 ACC이 아닌 BCR을 기준으로 최적의 하이퍼파라미터 값을 탐색하였다. ACC의 경우 본 데이터셋과 같이 데이터 class가 불균형인 경우에 분류기가 다수의 class에 편중될 가능성을 배제하지 못한다. 반면 BCR의 경우 이러한 ACC의 단점을 극복한 성능 평가 지표로써 재현율, 정밀도 지표를 종합적으로 고려하여 이를 기하평균하여 산출한 지표이다. 따라서 본 데이터셋이 데이터 class가 불균형을 띄기에 ACC보다 효과적인 성능 평가 지표라고 판단 BCR을 기준으로 최적의 하이퍼파라미터 값을 탐색하였다.

### 1-2. CART

MLR 모델과 동일한 방식으로 최적의 CART 모델을 찾았다. 이 때의 설정한 하이퍼파라미터 중 최적 값은 다음과 같다.

1. criterion: [gini] 2. Max_depth: [None] 3. Min_samples_split: [10]

### 1-3. ANN

마찬가지 방식으로 최적의 ANN모델을 찾았다. 이 때의 설정한 하이퍼파라미터 중 최적 값은 다음과 같다.

1. Hidden_layer_sizes: [50,] 2. activation: [relu] 3. solver: [lbfgs] 4. Early_stopping: [True]

### 1-4. 종합 성능지표

| Model | ACC | BCR |
|---|---|---|
| MLR | 0.59100 | 0.49911 |
| CART | 0.56500 | 0.59721 |
| ANN | 0.54400 | 0.57036 |

세 모델의 종합적인 성능 지표를 살펴보면 MLR, CART, ANN 순으로 ACC 값이 높게 산출되었다. 즉, 단순히 ACC 측면에서 살펴본다면 MLR, CART, ANN 순으로 성능이 우수하다고 판단할 수 있다. 하지만, 본 데이터셋이 클래스 간 불균형이 존재하기에 ACC 지표보다는 클래스 간 불균형을 고려한 지표인 BCR 지표로 평가하는 것이 합리적이라고 판단하였다.

BCR 기준으로 CART, ANN, MLR 순으로 성능이 좋다고 해석할 수 있다. 우선 MLR의 경우 독립변수와 종속 변수 간 선형 관계를 가정한다. 하지만, CART, ANN의 성능이 MLR보다 우수한 것으로 미뤄보았을 때 데이터 간 비선형 관계가 존재함을 유추할 수 있다. 이 요인 이외에도 본 데이터셋이 고차원 데이터셋이라는 점, 클래스 간 불균형이 존재하는 데이터셋이라는 점 등의 요인이 작용한 것으로 해석된다.

CART, ANN 모두 데이터 간 비선형 관계가 존재할 때 좋은 분류 모델이다. 하지만, CART의 경우 ANN보다 성능이 우수하게 나왔다. CART의 경우 데이터 셋의 피처 간 상호작용 고려할 수 있고, 데이터셋에 불균형이 존재하는 경우 클래스 가중치를 설정하여 불균형 문제를 해결할 수 있다. 또한 본 데이터셋을 분석할 때 이상치 제거 작업을 따로 수행하지 않았는데 CART는 이상치에 대해서도 robust한 성질을 가지고 있는 분류 모델이게 CART가 ANN보다 성능이 좋게 나왔다고 해석할 수 있다.

## [Q2] MLR, CART, ANN 분류 모델 구축

[Q2] CART의 Bagging 모델을 Bootstrap의 수를 (10, 30, 50, 100, 200, 300)의 순으로 증가시키면서 분류 정확도를 평가해보시오. 최적의 Bootstrap 수는 몇으로 확인되는가? 이 모델은 단일 모형과 비교했을 때 성능의 향상이 있는가?

| # of Bootstrap | ACC | BCR |
|---|---|---|
| 10 | 0.65275 | 0.62530 |
| 30 | 0.66450 | 0.62877 |
| 50 | 0.67125 | 0.62780 |
| 100 | 0.67250 | 0.62989 |
| 200 | 0.67075 | 0.62269 |
| 300 | 0.66975 | 0.62380 |

부트스트랩의 개수를 증가시켜 모델을 학습시켜도 ACC와 BCR값이 거의 일정한 양상을 보였다. 그 이유로 CART Bagging이 안정적인 성능 수준에 이미 도달하여 더 이상 반복 횟수를 증가시켜도 성능에 유의미한 영향을 주지 않기 때문이라고 해석된다.

최적의 부트스트랩 개수를 선정 시 앞선 문항과 마찬가지로 데이터셋의 불균형으로 최적의 부트스트랩 개수 선정 시 BCR을 ACC보다 우선적인 기준으로 사용하겠다. 부트스트랩 개수가 100일 때 BCR 값이 0.62989로 가장 높게 나왔다. 뿐만 아니라 부트스트랩 개수가 100일 때 ACC값이 0.67250로 가장 높게 나왔다. 따라서 <u>최적의 부트스트랩 개수는 100개이다.</u>

| Model | ACC | BCR |
|---|---|---|
| CART BAGGING | 0.65200 | 0.61657 |
| CART | 0.56500 | 0.59721 |

위 표를 보면 알 수 있듯이 CART bagging모델이 CART 단일 모델과 비교하였을 때 ACC, BCR 값이 모두 높기에 성능이 향상되었다는 사실을 확인할 수 있다.

우선 CART bagging 모델은 여러 개의 CART 모델의 예측을 종합한 것이기에 단일 모델보다 뛰어난 성능을 보인 것으로 해석된다. 또한 부트스트랩을 생성하여 데이터의 다양성을 확보하여 다양한 패턴을 확보함과 동시에 기존 데이터셋만 사용 시 발생하는 학습 데이터에 오버피팅되는 상황을 방지할 수 있다. 부트스트랩을 통한 다양한 데이터셋 형성은 클래스 불균형 문제를 해결하는 효과를 가진다.

## [Q3] Random Forest

Random Forest 모델의 Tree의 수를 (10, 30, 50, 100, 200, 300)의 순으로 증가시키면서 분류 정확도를 평가하고 다음 물음에 답하시오. 학습 과정에서는 변수의 중요도가 산출되도록 학습하시오.

[Q3-1] 최적의 Bootstrap 수는 몇으로 확인되는가?

| # of trees | ACC | BCR |
|---|---|---|
| 10 | 0.63200 | 0.59885 |
| 30 | 0.65075 | 0.59929 |
| 50 | 0.65525 | 0.59986 |
| 100 | 0.66375 | 0.59931 |
| 200 | 0.65975 | 0.59666 |
| 300 | 0.66025 | 0.59887 |

앞선 문항과 동일한 이유로 데이터셋에 불균형이 존재하기에 BCR 기준으로 최적의 부트스트랩 개수는 50으로 확인되었다.

부트스트랩 개수 50을 기점으로 ACC, BCR이 증가하다가 이후 감소하는 추세를 보인다. 이는 랜덤 포레스트가 안정적인 성능 수준에 도달하여 부트스트랩 개수가 증가하더라도 성능에 영향을 미치지 못한 것으로 해석된다. 50 이후에 부트스트랩 개수가 증가함에도 성능 지표가 오히려 감소하는 추세를 보이는 것은 분기 변수의 임의 추출 방식을 통해 개별 Tree들을 생성 했음에도 개별 Tree들의 다양성 확보로 인한 효과보다 개별 Tree의 성능 감소 효과가 더 크기 때문이라고 해석된다. 또한 랜덤 포레스트의 부트스트랩 생성, 분기 변수 추출 방식에는 모두 확률적 요소가 관여하기에 이 또한 영향을 미쳤을 것이라 해석된다.

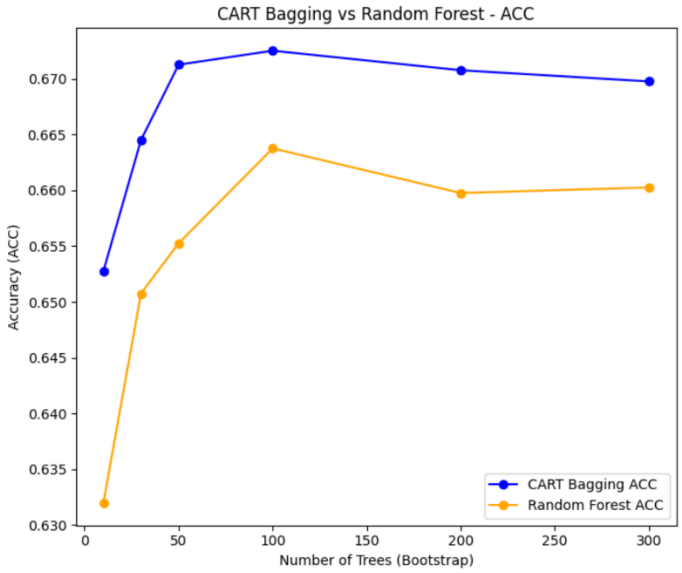
[Q3-2] 최적의 Tree 수를 기준으로 이 데이터셋에 대해서는 CART Bagging과 Random Forest 중에서 더 높은 분류 정확도를 나타내는 모형은 무엇인가?

| Model | ACC | BCR |
|---|---|---|
| CART BAGGING | 0.65200 | 0.61657 |
| RF | 0.64500 | 0.58483 |

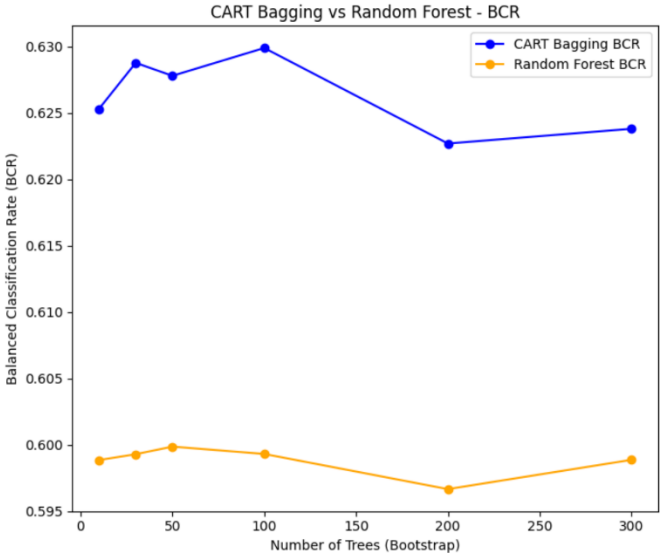
위 표를 보면 알 수 있듯이 CART Bagging이 Random Forest보다 ACC, BCR값이 높아 더 높은 분류 정확도를 나타낸다는 것을 알 수 있다. 이는 RF가 CART Bagging보다 성능이 좋을 것이라는 예상과는 상반된 결과이다.

우선 RF는 개별 부트스트랩에서 트리 생성 시 분기를 하는 변수 임의 추출함으로써 CART Bagging보다 다양성을 확보할 수 있다는 장점이 있다. 하지만, 전체 변수를 가지고 분기점을 탐색하는 CART Bagging보다 엄연히 적은 변수를 가지고 분기점을 탐색하기에 개별 트리의 성능은 확연히 떨어진다. 위 결과에서 알 수 있듯이 개별 트리의 성능 감소 효과가 다양성 확보 효과보다 크게 작용하여 RF의 성능이 CART Bagging보다 낮게 나온 것으로 해석된다. 이외에도 랜덤 포레스트의 하이퍼파라미터 값이 적절하지 않게 설정되지 않았을 가능성도 존재한다.

[Q3-3] 각 Tree의 수(Bootstrap의 수)마다 CART Bagging 모형과의 분류 정확도를 비교할 수 있는 그래프를 도시하시오. Tree의 수는 CART Bagging과 Random Forest는 성능 차이에 영향을 미친다 고 볼 수 있는가?

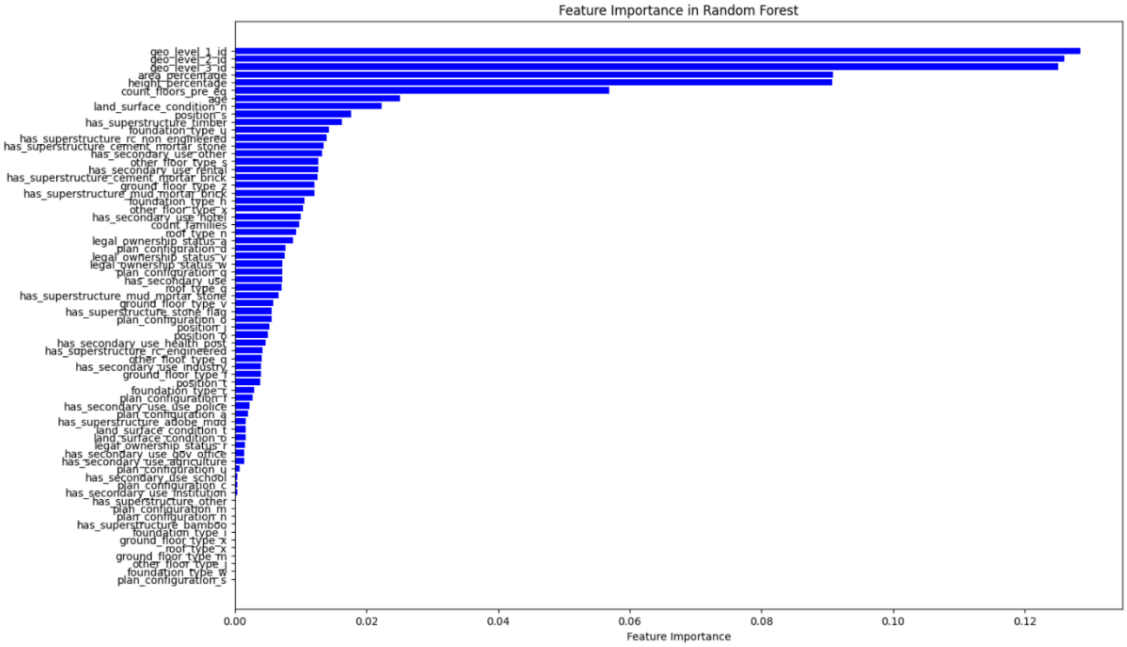



부트스트랩 개수가 100을 기점으로 두 모델 모두 ACC값이 증가하다가 이후 약간의 감소하는 추세를 보인다. 또 모든 부트스트랩에 대해 CART Bagging이 RF보다 높은 ACC값을 보였다. 이는 부트스트랩 개수의 증가가 어느 정도 다양성 확보에 기여하여 성능 향상에 영향을 끼쳤고, 특정 시점 이후로 안정적 성능에 도달했다고 해석할 수 있다. 그러나 부트스트랩 샘플링 자체의 확률적 요소, 부트스트랩 수가 증가함에 따른 훈련 데이터에 과적합 가능성 존재 등의 이유로 ACC값이 약간의 감소 추세를 보였다고 해석할 수 있다. 특히 CART Bagging에서 한 단계 더 나아간 RF의 경우 분기 변수 임의 추출에서 또 한 번 확률적 요소가 영향을 끼친다고 할 수 있다.




위 그래프를 보면 알 수 있듯이 부트스트랩 수에 따른 BCR 값의 변동폭이 큰 것을 관찰할 수 있다. 특히 CART Bagging의 경우 RF보다 변동폭이 큰데 이를 통해 부트스트랩 개수의 변화가 CART Bagging에 더 많은 영향을 준다고 해석할 수 있다. ACC와 마찬가지로 CART Bagging이 RF보다 모든 부트스트랩에 대해 높은 값을 지니는 것을 확인할 수 있다. 이를 통해 CART Bagging이 RF보다 뛰어난 분류 성능을 지닌다고 해석할 수 있다. CART Bagging, RF의 BCR 최대값은 각각 100, 50일 때 나타난다. 이를 통해 부트스트랩 수 증가가 특정 시점까지는 성능 향상에 영향을 미치나 이후에는 영향을 미치지 않는다는 것을 확인할 수 있다. 이는 앞서 ACC에서 살펴봤듯이 확률적 요소, 과적합 문제, 안정적 분류 성능 도달 등의 요인이 작용한 것으로 해석된다.

*추가적으로 최적의 RF 모델에 대한 변수 중요도는 아래와 같다.



변수의 수가 많은 관계로 변수 중요도를 한 눈에 알아보기 어려워 아래와 같이 표를 만들었다.

| | Feature Name | Feature importance |
|---|---|---|
| 0 | geo_level_1_id | 0.128396 |
| 1 | geo_level_2_id | 0.125961 |
| 2 | geo_level_3_id | 0.125015 |
| 3 | area_percentage | 0.090794 |
| 4 | height_percentage | 0.090759 |
| .. | ... | ... |
| 63 | roof_type_x | 0.000043 |
| 64 | ground_floor_type_m | 0.000015 |
| 65 | other_floor_type_j | 0.000000 |
| 66 | foundation_type_w | 0.000000 |
| 67 | plan_configuration_s | 0.000000 |

[68 rows x 2 columns]

변수 중요도가 높은 상위 5개의 변수를 살펴보도록 하겠다.

geo_level_1_id: 지리적인 위치의 최상위 단계, 지역의 광범위한 지리적인 위치가 건물 파괴 정도와 관련이 있는 것으로 해석할 수 있다.

geo_level_2_id: 지리적인 위치의 중간 단계, 지역의 일반적인 지리적인 위치가 건물 파괴 정도와 관련이 있는 것으로 해석할 수 있다.

geo_level_3_id: 지리적인 위치의 세부 단계, 지역의 세부 위치가 건물의 파괴 정도를 예측하는 데에 큰 영향을 미친다.

area_percentage: 건물이 차지하는 지역의 비율, 건물의 크기나 지역의 차지 비율이 파괴 정도와 관련이 있을 수 있기 때문에 중요한 변수임을 파악할 수 있다.

height_percentage: 건물이 차지하는 높이의 비율, 건물의 높이가 파괴 정도와 관련이 있을 수 있기 때문에 중요한 변수임을 파악할 수 있다.

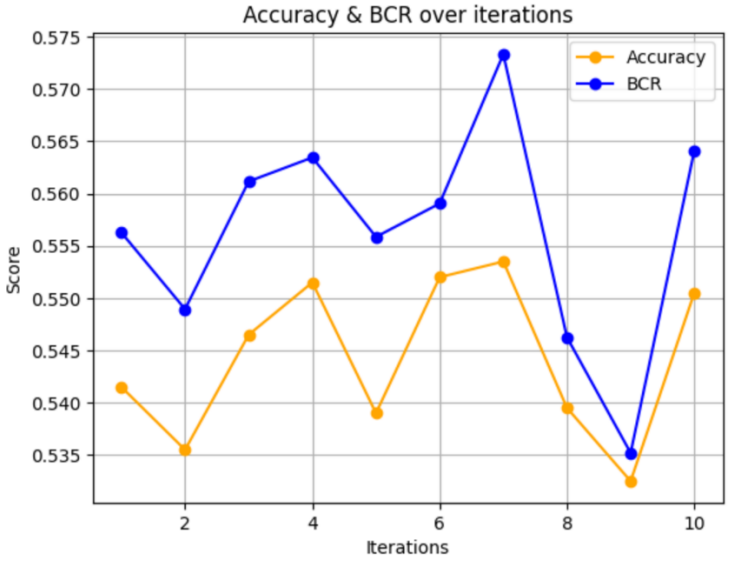
## [Q4] ANN

[Q1]에서 찾은 최적의 hyperparameter를 이용하여 ANN 단일모형을 10번 반복하여 테스트 정확도를 평가해보시오. Accuracy와 BCR의 평균 및 표준편차를 기록하시오.

| Model | ACC | BCR |
|---|---|---|
| ANN(단일) | 0.54400 | 0.57036 |
| ANN(10회 반복) | 0.54420<br>(±0.00717) | 0.55636<br>(±0.01017) |

( )는 표준편차를 나타낸다. 단일 ANN 모델이 10회 반복 ANN 모델보다 ACC 값은 작고, BCR값은 크게 나타났다. 클래스 간 불균형이 존재하고, ACC의 차이는 BCR보다 미미하기에 BCR을 기준으로 중점적으로 해석을 진행하겠다. 우선 10회 반복 ANN 모델의 BCR이 더 낮은 것을 통해 하이퍼파라미터 값이 잘못 설정되었을 가능성, 데이터에 적합하지 않은 모델일 수도 있다는 사실을 확인할 수 있었다.

특히 BCR의 표준편차가 ACC보다 높은 것을 확인할 수 있다. 이는 클래스 불균형으로 인해 하나의 다수 클래스로 예측하는 경우가 발생하여 클래스 불균형을 고려하는 성능 평가 지표인 BCR의 표준편차가 크게 나왔다고 해석된다.




위 표는 반복 수에 따른 ACC, BCR을 시각화한 그래프이다. ACC, BCR 비슷한 변화 추이 양상을 보이며, 모든 반복 수에 대해 BCR 값이 ACC보다 높게 나타난 것을 확인할 수 있다. 비슷한 변화 추이 양상을 보이는 것은 모든 클래스에 대해 모델의 성능이 일관되게 변화되기 때문이라고 해석

된다.

특이한 점은 반복을 진행함에 있어서 성능 평가 지표가 갑자기 아래로 튀는 부분이 존재한다는 것이었다. 이는 ANN 모델의 특성상 지역 최소값에 빠질 가능성이 존재하여 전역 최소값을 찾지 못하는 즉, 성능 개선이 이뤄지지 않았을 가능성 때문이라고 생각된다. 이는 하이퍼파라미터 및 적절한 종료 조건 설정을 통해 해결할 수 있을 것이라 생각한다.

## [Q5] ANN Bagging

[Q5-1] Bootstrap의 수를 (10, 30, 50, 100, 200, 300)의 순으로 증가시키면서 Accuracy와 BCR을 구해 보시오.

| # of bootstrap | ACC | BCR |
|---|---|---|
| 10 | 0.58850 | 0.58468 |
| 30 | 0.60025 | 0.58328 |
| 50 | 0.60075 | 0.57998 |
| 100 | 0.60300 | 0.58086 |
| 200 | 0.60225 | 0.57575 |
| 300 | 0.57575 | 0.58056 |

부트스트랩 수가 증가함에 따라 ACC, BCR의 변화 추이는 크게 없는 것을 확인할 수 있다. 이는 모델이 안정적인 성능에 도달하여 부트스트랩 수 증가가 성능 향상에 큰 영향을 미치지 못했기 때문이라고 해석된다. 다만, ACC의 경우 BCR의 경우보다 변화 추이 폭이 큰 것을 확인할 수 있는데 이는 ACC가 다수 클래스의 성능 변화에 더 민감하게 반응하기 때문이라고 해석된다. 즉, 클래스 불균형이 존재하는 데이터셋에서는 다수 클래스의 성능이 주로 평가되므로, 다수 클래스의 성능 영향을 많이 받는 ACC의 변화 폭이 BCR보다 크게 나타났다고 해석된다.

[Q5-2] 최적의 Bootstrap 수는 몇으로 확인되는가?

앞선 문항들과 동일한 이유로 BCR 값을 기준으로 최적의 부트스트랩 수를 산출한 결과 10인 것을 확인할 수 있었다.

| # of bootstrap | ACC | BCR |
|---|---|---|
| 10 | 0.58850 | 0.58468 |

다시 말해 최적의 부트스트랩 수는 제일 작은 값인 10이 나왔다. 이는 개별 ANN모델들에 다양성이 존재하지 않기에 Bagging variance를 줄이는데 적합하지 않기에 나온 현상이라고 해석된다.

또한 부트스트랩 생성 시 확률적 요소가 관여한다는 사실도 영향을 미친 것으로 해석된다.

[Q5-3] 이 모델은 최적의 단일 모형과 비교했을 때 어떠한 성능의 차이가 있는가?

| Model | ACC | BCR |
|---|---|---|
| ANN | 0.54420<br>(±0.00717) | 0.55636<br>(±0.01017) |
| ANN Bagging | 0.57400 | 0.55538 |

우선 ACC의 경우 ANN 모델이 ANN Bagging모델보다 값이 작게 나타났다. 이를 통해 ACC 측면에서 ANN Bagging이 어느 정도의 성능 개선을 이뤘다고 볼 수 있다.

그러나 BCR의 경우 ANN 모델이 ANN Bagging모델보다 더 크게 나타났다. ANN Bagging 모델이 ANN모델보다 성능이 좋을 것이라는 일반적인 예상과는 상이한 결과이다. Bagging 전략은 부트스트랩 생성을 통한 데이터셋의 다양성 확보가 전제되어야 한다. 그러나 ANN Bagging 모델의 성능이 더 낮은 것을 통해 다양성 확보가 제대로 이뤄지지 않았다고 해석된다. 다양성을 확보하지 못했던 이유로는 부트스트랩 생성 시 확률적 요소의 영향, 하이퍼파라미터 값이 잘못 설정되었을 가능성이라고 생각된다. 특히 앞서 최적의 부트스트랩 개수가 10개로 가장 적은 개수가 나왔는데 ANN과 ANN Bagging 성능 지표 비교를 통해 이 값 또한 잘못 설정되었을 가능성을 배제할 수 없을 것이라 생각한다. 이는 최적의 부트스트랩 개수 탐색 시 각 부트스트랩에 대해 1번이 아닌 여러 번의 반복을 통해 모델의 일반화 성능을 산출하여 비교하면 더 정확한 결과를 기대할 수 있을 것이라 생각한다.

ACC, BCR을 기준으로 종합적으로 평가하면 ACC Bagging 모델의 성능 향상은 없다고 생각한다. ACC 지표의 향상이 있었으나 본 데이터셋이 클래스 간 불균형이 존재하여 BCR을 우선적으로 고려해야 한다고 판단하였기 때문이다.

## [Q6] Adaboost

[Q6-1] Hyperparameter 후보 값들을 명시하고, Validation dataset을 통해 최적의 hyperparameter 값을 찾아보시오.

Adaboost의 탐색할 하이퍼파라미터 후보값과 그 범위는 다음과 같다.

1. Learning_rate: [1, 0.1,0.01,0.001]

최종 combination에서 각 weak learner의 기여도를 제어하고 각 분류기에 적용되는 가중치를 조정한다. 학습률이 낮을수록 각 weak learner의 기여도가 줄어들어 보다 안정적인 수렴이 가능하나 모델 학습 시 많은 시간이 소요된다.

2. N_estimator: [50, 100, 150, 200]

순차적으로 학습할 최대 weak learner의 수를 결정한다. weak learner의 개수를 늘리면 잠재적으로 모델의 성능이 향상될 수 있지만, 과적합의 위험과 더불어 계산 시간도 증가한다.

이렇게 각 모델에 대해 설정한 하이퍼파라미터 후보값들을 적절히 조합하여 validation dataset에 대해 ACC, BCR 값을 산출하였고 그 중 BCR 값을 기준으로 최적의 하이퍼파라미터 조합을 찾았다(Q1-1과 동일한 방식으로 진행하였다). 그 결과 찾아낸 최적의 하이퍼파라미터 조합은 아래와 같다.

'learning rate': 0.001, 'n estimators': 200

[Q6-2] 최적의 hyperparameter 값을 이용하여 AdaBoost 모델을 학습한 뒤, Test dataset에 적용하여 먼저 구축된 모델들과 분류 성능을 비교해보시오.

| Model | MLR | CART | ANN | CART Bagging | RF | ANN Bagging | ADA Boost |
|---|---|---|---|---|---|---|---|
| ACC | 0.59100 | 0.56500 | 0.54400 | 0.65200 | 0.64500 | 0.57400 | 0.63850 |
| BCR | 0.49911 | 0.59721 | 0.57036 | 0.61657 | 0.58483 | 0.55538 | 0.61113 |

비교에 앞서 ANN의 경우 테스트 데이터 셋에 대해 10회 반복하여 산출한 ACC, BCR 지표 대신 동일한 테스트 데이터셋에 대해 1회 측정한 분류 성능 지표를 사용하였다. 이유는 나머지 모델들의 성능 지표가 모두 테스트 데이터셋에 대해 1회 측정한 지표이기에 공정성을 보장하고자 했기 때문이다.

우선 단일 모델인 MLR, CART, ANN보다 앙상블 기법을 적용한 나머지 모델들의 BCR 지표가 모두 높게 나온 것을 확인할 수 있다. 2016 MLConf SF의 10가지 메인 포인트 중 하나인 'Ensembles almost always work better'를 본 실습을 통해 다시 한번 확인할 수 있었다. 링크는 아래에 첨부하겠다.

https://tryolabs.com/blog/2016/11/18/10-main-takeaways-from-mlconf

서로 다른 모델은 서로 다른 분류 경계면과 예측 곡선을 생성하고 서로 다른 사고방식 체계를 가지고 있어 상호 보완이 가능하다. 이 때문에 여러 알고리즘을 결합하는 앙상블이 단일 모델보다 뛰어난 성능을 보인 것이라 해석된다.

유일하게 CART에만 Boosting 기법을 적용하였고 ACC, BCR 값은 CART Bagging>ADA Boost>CART 순으로 나왔다. 우선 Boosting 기법보다 Bagging 기법을 적용하였을 때 더 좋은 성능을 지니는 것은 CART가 분산은 높고, 편향은 낮은 모델이기 때문이다. 구체적으로 Bagging 기법은 모델의 다양성 확보를 통한 분산을 낮추는데 효과적이고, Boosting 기법은 이전 모델의 학습

결과를 기반으로 학습을 반복하기에 편향을 낮추는데 효과적인 기법이다. 따라서 Boosting 기법보다 Bagging 기법을 적용하였을 때 더 좋은 성능을 보인 것이다. 그럼에도 Ada Boost 모델이 단일 CART 모델보다 좋은 성능을 보인 것은 학습 시 잘못 분류한 샘플에 대해 가중치를 부여하여 오류를 줄여 나가는 특성을 지녔기 때문이라고 해석된다.

## [Q7] GBM

[Q7-1] Hyperparameter 후보 값들을 명시하고, Validation dataset을 통해 최적의 hyperparameter 값을 찾아보시오.

GBM에는 다음과 같은 하이퍼파라미터가 사용된다.

| Hyperparameter | 특징 |
|---|---|
| n_estimators | 부스팅 단계 또는 결정 트리 수 결정 |
| subsample | 각 트리를 훈련하는데 사용할 샘플의 비율 |
| loss | 부스팅 과정에서 최적화할 손실 함수 |
| min_samples_split | 노드 분할 시 필요한 최소 샘플 수 |
| min_samples_leaf | 리프 노드에 있어야 하는 최소 샘플 수 |
| max_depth | 의사 결정 트리의 최대 깊이 |

위의 파라미터 중 n_estimators, subsample, loss 3개에 대해 다음과 같은 값을 가지고 탐색을 진행하였다.

| Hyperparameter | 특징 |
|---|---|
| n_estimators | 30, 50, 70 |
| subsample | 0.5, 0.7, 0.9 |
| loss | deviance, log_loss |

사용한 하이퍼파라미터에 대해 추가적인 설명은 아래와 같다.

-n_estimators

GBM의 매우 중요한 하이퍼파라미터 중 하나로 모델에 사용되는 weak learner 즉, 결정 트리의 개수를 지정한다. n_estimators의 수가 클수록 데이터의 복잡한 관계를 확인할 수 있으나 계산 비용 및 과적합의 우려도 존재한다. 따라서 적절한 수의 n_estimators를 찾는 것이 중요하고 본 실험에서는 30, 50, 70 세 개의 값을 가지고 하이퍼파라미터 탐색을 진행하였다.

-Subsample

GBM의 각 트리를 학습하기 위해 무작위로 선택할 샘플의 비율을 제어하는 하이퍼파라미터. 1.0 미만의 값을 설정하면 전체 데이터 중 일부만이 학습에 참여하기에 모델의 과적합을 방지하고, 무작위성을 도입할 수 있다. 실험에서는 0.5, 0.7, 0.9값을 사용하였다.

-loss: 그래디언트 부스팅에서 최적화할 손실 함수를 결정하는 하이퍼파라미터.

실험에서는 'deviance', 'log_loss' 두 가지 손실 함수를 사용하였다. 로그 손실은 예측 확률과 실제 클래스 레이블 간의 차이 평가, deviance는 분류 작업에 일반적으로 사용되는 로지스틱 회귀 loss에 해당한다.

앞서 살펴본 모델들과 동일한 방식으로 최적의 하이퍼파라미터 탐색을 진행하였고, BCR 기준으로 가장 좋은 성능을 발휘한 조합은 아래와 같다.

```
->'loss': 'deviance' 'n_estimators': 70, 'subsample': 0.5
```

[Q7-2] 최적의 hyperparameter 값을 이용하여 GBM 모델을 학습(변수의 중요도가 산출되도록 학습) 한 뒤, Test dataset에 적용하여 먼저 구축된 모델들과 분류 성능을 비교해보시오.

| Model | MLR | CART | ANN | CART Bagging | RF | ANN Bagging | ADA Boost | GBM |
|---|---|---|---|---|---|---|---|---|
| ACC | 0.59100 | 0.56500 | 0.54400 | 0.65200 | 0.64500 | 0.57400 | 0.63850 | 0.66750 |
| BCR | 0.49911 | 0.59721 | 0.57036 | 0.61657 | 0.58483 | 0.55538 | 0.61113 | 0.59948 |

우선 단일 모델인 MLR, CART, ANN보다 GBM을 적용한 모델의 ACC, BCR 지표가 모두 높게 나온 것을 확인할 수 있다. 이는 이전 모델에서 발생한 오류를 줄여 나가는 방식으로 Boosting하는 것이 성능 개선에 기여했다고 해석된다.

다만, GBM은 단일모델과 RF 모델을 제외한 모든 앙상블 모델과 AdaBoost 모델보다 BCR 지표가 낮게 나온 것을 확인할 수 있다. 이는 각각의 학습 단계에서 오류가 최소화하려는 방향으로 모델을 학습하기 때문에 모델의 복잡한 패턴을 학습하기 용이하다는 장점을 가지고 있으나 동시에 그만큼 과적합될 가능성이 높은 GBM의 특성 때문이라고 해석된다.

특히 직전 문항에서 실행한 AdaBoost보다 좀 더 균형 있는 예측을 수행하여 성능 지표가 우수

하게 나올 것으로 예상했다. Learning_rate, subsample과 같은 하이퍼파라미터를 통해 과적합을 방지할 수 있을 것이라 생각했기 때문이다. 그러나 탐색한 하이퍼파라미터 값의 범위 설정 때문인지 결과는 예상과는 반대로 나왔다.

그래서 찾은 최적의 하이퍼파라미터를 살펴보았고, subsample의 최적 값은 탐색한 하이퍼파라미터 값 중 가장 작은 값인 0.5가 나왔다. Subsample의 탐색 범위를 0.5이하로 낮추고, learning_rate도 하이퍼파라미터 탐색 작업을 수행한다면 더 좋은 성능이 나올 것이라 생각한다.

CART 기반의 Bagging 모델이 Cart 기반의 Boosting(Adaboost, GBM)보다 성능이 뛰어남을 위 표를 통해 확인할 수 있다. 이는 CART가 분산은 높고, 편향은 작은 모델이기에 다양성을 통해 분산을 낮추는데 유리한 Bagging 기법이 이전 모델의 성능 개선에 초점을 두어 편향을 낮추는데 유리한 Boosting 기법보다 더 적합한 방식이 때문이다. 모델의 관점에서 나아가 데이터셋의 관점에서 살펴본다면 해당 데이터셋이 Boosting 보다 Bagging에서 더 좋은 효과를 발휘할 수 있는 데이터셋이라고 해석된다.

[Q7-3] 산출된 변수의 중요도를 해석해보고, Random Forest 모델에서 산출된 주요 변수와 비교해보시오.

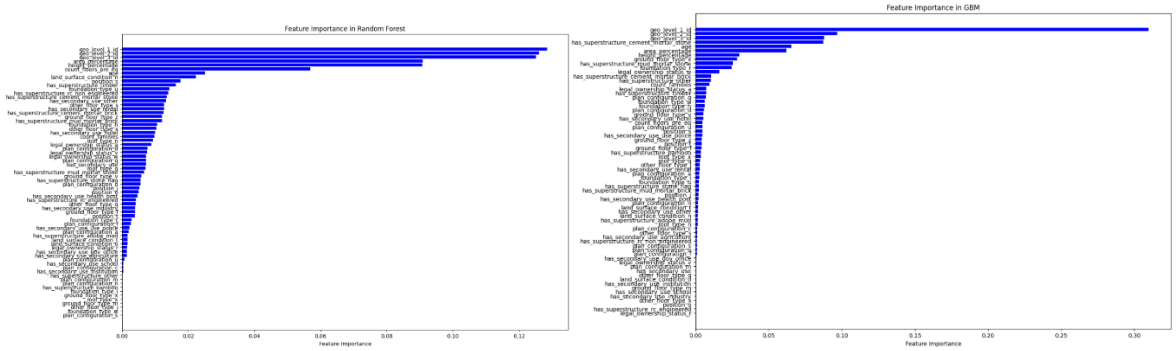
| | Feature Name | Feature importance |
|---|---|---|
| 0 | geo_level_1_id | 0.128396 |
| 1 | geo_level_2_id | 0.125961 |
| 2 | geo_level_3_id | 0.125015 |
| 3 | area_percentage | 0.090794 |
| 4 | height_percentage | 0.090759 |
| .. | ... | ... |
| 63 | roof_type_x | 0.000043 |
| 64 | ground_floor_type_m | 0.000015 |
| 65 | other_floor_type_j | 0.000000 |
| 66 | foundation_type_w | 0.000000 |
| 67 | plan_configuration_s | 0.000000 |

[68 rows x 2 columns]

| | Feature Name | Feature importances |
|---|---|---|
| 0 | geo_level_1_id | 0.309602 |
| 1 | geo_level_2_id | 0.096736 |
| 2 | geo_level_3_id | 0.087952 |
| 3 | has_superstructure_cement_mortar_stone | 0.087483 |
| 4 | age | 0.065684 |
| .. | ... | ... |
| 63 | has_secondary_use_industry | 0.000000 |
| 64 | other_floor_type_x | 0.000000 |
| 65 | position_o | 0.000000 |
| 66 | has_superstructure_rc_engineered | 0.000000 |
| 67 | legal_ownership_status_r | 0.000000 |

[68 rows x 2 columns]

좌측이 RF, 우측이 GBM으로 산출된 변수의 중요도이다.




마찬가지로 좌측이 RF, 우측이 GBM으로 산출된 변수 중요도를 시각화한 그래프이다.

우선 GBM으로 산출된 변수 중요도가 높은 상위 5개의 변수를 살펴보도록 하겠다.

geo_level_1_id: 지리적인 위치의 최상위 단계, 지역의 광범위한 지리적인 위치가 건물 파괴 정도와 관련이 있는 것으로 해석할 수 있다.

geo_level_2_id: 지리적인 위치의 중간 단계, 지역의 일반적인 지리적인 위치가 건물 파괴 정도와 관련이 있는 것으로 해석할 수 있다.

geo_level_3_id: 지리적인 위치의 세부 단계, 지역의 세부 위치가 건물의 파괴 정도를 예측하는 데에 큰 영향을 미친다.

has_superstructure_cement_mortar_stone: 시멘트 모르타와 돌로 만들어진 구조물인지 여부, 이는 건물의 내구성과 안정성과 연관된 지표로 파괴 정도와 관련이 있을 수 있기 때문에 중요한 변수임을 파악할 수 있다.

age: 건물의 연식, 건물의 연식은 곧 건물의 노후화와 관련 있어 파괴 정도와 관련이 있을 수 있기 때문에 중요한 변수임을 파악할 수 있다.

우선 RF, GBM 공통적으로 geo_level_1_id, geo_level_2_id, geo_level_3_id순으로 변수 중요도가 높았다. 즉, 모델에 관계없이 중요하다고 산출되었기에 지리적 특성과 관련된 세 변수는 정말 중요한 변수임을 알 수 있다.

그러나 두 모델의 중요도 상위 5개 변수명이 모두 같지는 않았다. 이는 앙상블 기법의 차이라고 해석된다. 구체적으로 RF의 경우 독립적인 Bootstrap에서 모델을 학습하고 결합하나 GBM의 경우 iteration을 반복하며 update가 진행되기에 독립적이지 않다. 즉, Bagging과 Boosting의 차이가 변수의 차이를 만들었다고 해석된다.

또한 RF와 GBM의 경우 모두 상위 변수에 중요도가 집중된 양상을 보이나 GBM의 경우 상위 변수들의 중요도 비중이 RF보다 높은 것을 시각적으로 확인할 수 있다. 특히 GBM에서

geo_level_1_id의 경우 중요도가 0.309602로 약 31% 가량의 매우 높은 중요도를 차지하는 것을 확인할 수 있다. 이는 GBM 특성 상 이전 모델의 결과를 다음 모델에 반영하여 update를 진행하기에 특정 변수에 대한 가중치가 부여될 가능성이 높기 때문이다. 반면, RF의 경우 생성된 부트스트랩에 대해 모델별로 독립적인 sampling을 고려하기에 RF보다 다양한 변수를 고려할 여지가 많다.

즉 Bagging과 Boosting 기법의 차이가 중요도 상위 5개 변수의 차이와 변수의 중요도 분포의 차이를 만들었다고 해석할 수 있다.

## [Q8] BCR 기준 성능 비교

Q8] 총 여덟 가지의 모델(Multinomial logistic regression, CART, ANN, CART Bagging, ANN Bagging, Random Forest, AdaBoost, GBM) 중 BCR 관점에서 가장 우수한 분류 정확도를 나타내는 모형은 무엇인가?

| Model | MLR | CART | ANN | CART Bagging | RF | ANN Bagging | ADA Boost | GBM |
|---|---|---|---|---|---|---|---|---|
| ACC | 0.59100 | 0.56500 | 0.54400 | 0.65200 | 0.64500 | 0.57400 | 0.63850 | 0.66750 |
| BCR | **0.49911** | **0.59721** | **0.57036** | **0.61657** | **0.58483** | **0.55538** | **0.61113** | **0.59948** |

BCR 기준으로 가장 좋은 성능을 보인 모델은 CART Bagging이다. ACC값 역시 CART Bagging이 제일 높게 나왔다. 뒤를 이어 ADA Boost, GBM 순으로 성능이 좋았다.

ANN 단일 모델의 성능은 오히려 ANN Bagging보다 뛰어났다. 앞선 ANN과 ANN Bagging 비교 문항에서 언급하였듯이 다양성을 제대로 확보하지 못하여 발생한 현상이라고 해석된다.

Bagging 계열의 모델 중에서는 CART Bagging의 성능이 가장 우수하였고, ANN Bagging의 성능이 가장 좋지 못했다. 이를 통해 ANN보다 CART가 본 데이터셋에 대해 Bagging 기법에 더 잘 맞는 것을 알 수 있다. 특히 RF의 경우 예상과는 다르게 CART Bagging보다 성능 지표가 낮게 나왔다. 이는 앞선 문항에서 언급하였듯이 RF의 개별 트리의 성능 감소 효과가 다양성 확보 효과보다 크게 작용하였기 때문이라고 해석된다. 랜덤 포레스트의 하이퍼파라미터 값이 적절하지 않게 설정되지 않았을 가능성도 존재하기에 향후 탐색할 하이퍼파라미터 값을 조정하여 다시 수행한다면 결과가 달라질 수도 있을 것이라 생각한다.

Boosting 계열의 모델 중에서는 AdaBoost가 GBM보다 성능이 뛰어났다. 이는 이전 모델의 오류를 학습하여 다음 모델을 업데이트하는 GBM의 특성 상 모델의 노이즈까지 학습하여 과적합될 가능성이 있기 때문이라고 해석된다.

ANN Bagging을 제외한 Boosting, Bagging을 적용한 모델들은 전반적으로 단일 모델보다 우수한

성능을 보였다. 단, 단일모델 CART의 경우 성능 지표가 앙상블 모델과 흡사하거나 심지어 높게 나온 경우도 있었다. CART 기반 앙상블 모델 역시 ANN 기반 앙상블 모델보다 성능이 좋은 것을 관찰할 수 있는데 이를 통해 본 데이터셋에 대해 ANN보다 CART가 더 잘 맞는 모델이라고 해석할 수 있다.

## Extra Question

이 데이터셋은 아래 표와 같이 Class 2 > Class 3 > Class 1 순으로 높은 비중을 차지하고 있으며, 범주의 불균형이 상당한 수준이다. [Q8]에서 선정된 가장 우수한 모델(알고리즘 및 hyperparameter)에 대해서 데이터 전처리 관점에서 불균형을 해소하여 분류 성능을 향상시킬 수 있는 아이디어를 제시하고 실험을 통해 검증해보시오.

[Q8]에서 선정된 가장 우수한 모델은 CART Bagging 모델이였다. 데이터 전처리 관점에서 불균형을 해소하여 분류 성능을 향상시킬 수 있는 방법에는 오버샘플링, 언더샘플링, 클래스 가중치 조정의 방법이 있다. 오버샘플링은 데이터셋에서 비중이 작은 클래스의 데이터를 복제하여 클래스의 비율을 맞추는 방법으로 대표적으로 SMOTE (Synthetic Minority Over-sampling Technique)가 있다. 언더샘플링은 데이터셋에서 비중이 큰 클래스의 데이터를 일부 제거하여 클래스의 비율을 맞추는 방법이다. 마지막으로 클래스 가중치 조정은 모델 훈련 시 손실 함수에 클래스 가중치를 부여하여 비중이 작은 클래스에 더 큰 가중치를 부여하는 방법이다.

위 세 가지 방법 중 다운샘플링 방식을 사용하여 분류 성능을 향상시키도록 하겠다. 데이터의 개수가 가장 적은 class 1의 경우 데이터 개수가 25124개로 모델을 학습시키기에 충분한 데이터 양이라고 생각하였고, 오버샘플링 방식보다 시간 복잡도 측면에서 유리할 것이라고 판단하였기 때문이다.

따라서 학습데이터에 대해 다음과 같이 언더샘플링을 하고 10, 30, 50, 100, 200, 300의 부트스트랩 수에 대해 ACC, BCR 값을 산출하여 BCR 값 기준 최적의 부트스트랩 수를 찾았다.

```
#Extra
from imblearn.under_sampling import RandomUnderSampler
# 언더샘플링 적용
undersampler = RandomUnderSampler(random_state=42)
x_train_under, y_train_under = undersampler.fit_resample(x_train,
y_train)

# Bagging Training
bootstrap_nums = [10, 30, 50, 100, 200, 300]
best_bcr = 0.0
best_bootstrap_num = None
```

```
best_bagging_tree = None


for bootstrap_num in bootstrap_nums:
    bagging_tree_model =
BaggingClassifier(base_estimator=best_model_tree,
n_estimators=bootstrap_num, n_jobs=8, bootstrap=True, verbose=1,
random_state=42).fit(x_train_under, y_train_under)
    y_val_pred = bagging_tree_model.predict(x_val)

    accuracy, bcr = calculate_matrix(y_val, y_val_pred)
    print(f"Bootstrap num: {bootstrap_num} - Accuracy: {accuracy:.5f},
BCR: {bcr:.5f}")

    if bcr > best_bcr:
        best_bcr = bcr
        best_bootstrap_num = bootstrap_num
        best_bagging_tree = bagging_tree_model

y_test_pred = best_bagging_tree.predict(x_test)
accuracy_bagging_tree, bcr_bagging_tree = calculate_matrix(y_test,
y_test_pred)
print(f'Best Bootstrap num: {best_bootstrap_num} Test Accuracy:
{accuracy_bagging_tree:.5f} Test BCR: {bcr_bagging_tree:.5f}')
```

Validation dataset을 가지고 최적의 부트스트랩 수를 찾고자 하였고, 부트스트랩 수에 따른 성능 지표는 다음과 같이 산출되었다.

| # of Bootstraps | ACC | BCR |
|---|---|---|
| 10 | 0.55575 | 0.68212 |
| 30 | 0.56075 | 0.69008 |
| 50 | 0.56900 | 0.69575 |
| 100 | 0.57700 | 0.70122 |
| 200 | 0.57800 | 0.69953 |
| 300 | 0.57600 | 0.69803 |

위 표를 보면 모든 부트스트랩 수에 대해 BCR 값이 ACC 값보다 높은 것을 확인할 수 있다. 이는 다운샘플링을 진행하며 나타난 다중 클래스의 영향력 감소, 소수 클래스의 영향력 증가 때문이라고 해석된다. BCR 값 기준 최적의 부트스트랩 수는 100이란 것을 확인하였고, 부트스트랩 수가 100인 CART Bagging 모델을 가지고 테스트 데이터셋을 이용하여 성능 지표를 산출하였다. 다운샘플링 적용 전과 후의 성능 지표는 아래와 같다.

| Model | ACC | BCR |
|---|---|---|
| CART BAGGING | 0.65200 | 0.61657 |
| CART BAGGING(Downsampling) | 0.55350 | 0.68159 |

다운 샘플링 후의 모델 성능 지표는 ACC는 더 낮게 나왔으나 BCR은 더 크게 나왔다. 이는 다음과 같이 설명할 수 있다. 클래스 불균형이 있는 경우 ACC는 주로 다수 클래스에 의해 결정된다. 즉, 소수 클래스에 대한 성능을 잘 반영하지 못한다. 이를 해결하고자 다운샘플링을 진행하여 클래스 간 균형을 맞췄기에 다수 클래스가 더 이상 존재하지 않고, 다운샘플링 이전 다수 클래스의 영향력이 자연스레 떨어지게 된다. 이는 곧 ACC 값에 영향을 미치고 ACC 지표의 감소를 초래했다고 해석된다. 반대로 다운샘플링 이전 소수 클래스였던 클래스는 더 이상 소수 클래스가 아니게 되고 이에 대한 예측 성능이 향상되기에 각 클래스에 대해 더 고른 학습이 가능한 BCR값이 증가한 것으로 해석된다.