다변량데이터분석

Ensemble

과제 4

산업경영공학부

2019170815

신건우

Table of Contents

[사전 작업 사항]

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

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

[Q1] 다음과 같이 세 가지 단일 모형에 대하여 분류 모델을 구축하고 Accuracy와 BCR 관점에서 분류 정확도를 비교해 보시오. CART와 ANN의 경우 hyperparameter 후보 값들을 명시하고 Validation dataset을 통해서 최적의 값을 찾아서 Test에 사용하시오.

먼저, 과제를 진행하기 이전에 간단한 EDA를 진행했다.

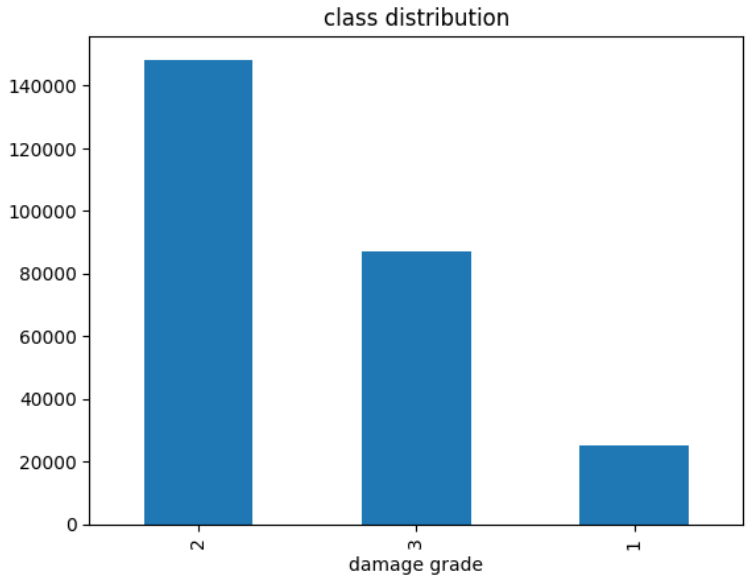
각 건물의 일련 번호인 building_id를 제외하여 입력 변수는 총 38개고, 결측치는 존재하지 않았다.

각 feature의 분포를 시각화하여 Appendix에 첨부했다.

사전 작업 사항[1]에 따라 8개의 명목형 변수들에 1-hot encoding을 적용하여 dummy variable을 생성했다. 여기서 범주의 개수만큼이 아닌 (범주의 개수 - 1)개의 dummy variable을 생성했다.
기저 범주에 대한 변수 중요도를 알 수 없다는 단점이 있지만, 데이터에 담긴 정보는 잃지 않으면서 feature의 수를 줄일 수 있기 때문에 computational cost 관점에 있어 보다 효율적일 것이라고 생각하여 해당 방식을 선택했다.

1-hot encoding을 적용한 결과 입력 변수의 개수가 60개가 되었다.

사전 작업 사항[2]에 따라 260601개의 원본 데이터에서 10000개의 데이터를 샘플링해야한다.



샘플링 이전에 classification을 수행할 종속 변수 damage_grade의 분포를 확인해본 결과, 다음과 같은 imbalance가 존재함을 확인했다. 따라서 sampling 과정에서 stratify 옵션을 사용해 해당 분포를 반영하도록 하여 모델의 학습 과정에서 일반화 성능을 어느정도 보장하고자 했다.

ANN모델을 사용할 예정이므로, 학습 데이터에 MinMaxScaler를 fit하여 학습, 검증, 테스트 데이터 모두를 정규화했다. MinMaxScaler를 사용한 이유는 여러 활성화함수에 적합한 0과 1 사이의 출력 범위를 지니며, 특정 feature의 단위에 관계없이 학습할 수 있기 때문이다.

MinMaxScaler는 이상치가 존재하면 데이터 분포에 왜곡이 생길 수 있다는 단점이 존재한다. 분포를 확인한 결과 눈에 띄는 이상치가 없다고 생각했기 때문에, MinMaxScaler를 사용해도 괜찮다고 판단했다.

실험에 앞서 재현성 보장을 위해 난수 시드들을 고정했다.

⬤Multinomial logistic regression

모델 학습에 Sklearn의 LogisticRegression을 사용했으며, Solver로 saga를 사용했다. SAGA는 확률적 경사 하강법(SGD)의 변형으로, 대규모 데이터셋을 처리하는 데 적합하다. 특히 희소 데이터와 같은 대규모 문제에서 효율적이라고 알려져 있는데, 현재 1-hot encoding으로 인해 보다 sparse해졌으므로 적합하다고 생각했다. 일반적으로 사용하는 lbfgs를 solver로 사용한 결과와 비교해 봤을 때 성능이 동일한 것을 확인할 수 있었다.

Lbfgs : Logistic Regression - Accuracy: 0.5950, BCR: 0.4729

Saga : Logistic Regression - Accuracy: 0.5950, BCR: 0.4729

⬤Classification and Regression Tree (CART)

Hyperparameter로 다음의 요소들을 고려했다. 해당 parameter들은 모델의 일반화 성능(overfitting)에 큰 영향을 주며, 검증 데이터로 모델의 일반화 성능을 평가하여 파라미터 조합을 선정했다.

- max_depth : 트리의 최대 깊이, 트리의 복잡도와 관련있다.
    - 3,5,7,10,None(제한없음)
- min_samples_split : 내부 노드를 분할하는 데 필요한 최소 샘플 수, 마찬가지로 트리의 복잡도와 관련있다.
    - 2,5,10
- min_samples_leaf : 리프 노드에 있어야 하는 최소 샘플 수, 마찬가지로 트리의 복잡도와 관련있다
    - 1,2,4

검증데이터에 대해 가장 좋은 BCR 값을 보이는 hyperparameter 조합을 최적 파라미터로 선정했다.

Best Parameters: {'max_depth': 10, 'min_samples_leaf': 4, 'min_samples_split': 10}

테스트 데이터에 대한 모델 성능은 다음과 같다.

CART - Accuracy: 0.6520, BCR: 0.5832

⬤Artificial Neural Network (ANN)

Pytorch와 gpu를 이용하여 하이퍼파라미터 튜닝 및 학습에 소요되는 시간을 단축하고자 했다. gpu로는 colab의 T4를 사용했다.

```python
class NeuralNet(nn.Module):
    def __init__(self, input_size, hidden_size, output_size, layer_num, activation_func):
        super(NeuralNet, self).__init__()
        layers = [nn.Linear(input_size, hidden_size)]

        # 활성화 함수 선택
        if activation_func == 'ReLU':
            activation = nn.ReLU()
        elif activation_func == 'Sigmoid':
            activation = nn.Sigmoid()
        elif activation_func == 'Tanh':
            activation = nn.Tanh()
        elif activation_func == 'LeakyReLU':
            activation = nn.LeakyReLU()
        elif activation_func == 'ELU':
            activation = nn.ELU()
        else:
            raise ValueError("지원되지 않는 활성화 함수입니다.")

        for _ in range(layer_num - 2):
            layers.append(nn.Linear(hidden_size, hidden_size))
            layers.append(activation)

        layers.append(nn.Linear(hidden_size, output_size))
        layers.append(nn.Softmax(dim=1))

        self.layers = nn.Sequential(*layers)

    def forward(self, x):
        return self.layers(x)
```

모델의 마지막에 softmax 활성화 함수를 적용하여 output으로 multiclass classification에서의 각 class 별 확률을 출력하도록 설계했다.

Batch size는 여러 번의 간단한 실험을 통해 hyperparameter search에 적정하다고 판단된 128로 설정했다.

학습 과정에는 epoch를 100으로 설정하고 overfitting을 방지하기 위해 patience 3의 early stop을 적용했다.

탐색할 ANN의 Hyperparameter로는 모델 성능의 큰 영향을 주는 요소들을 고려했으며, 다음과 같다.

- Hidden_size: 은닉층 노드 수
    - 32, 64, 128, 256
- Layer_num: layer의 수 (은닉층의 수와 연관)
    - 3, 5, 7
- Activation_func: 활성화 함수
    - ELU, ReLU, Tanh, Sigmoid
- Learning_rate: 학습률
    - 0.0001, 0.001, 0.005, 0.01

마찬가지로 검증데이터에 대해 가장 좋은 BCR 값을 보이는 hyperparameter 조합을 최적 파라미터로 선정했다.

Best Parameters: {'activation_func': 'ReLU', 'hidden_size': 128, 'layer_num': 3, 'learning_rate': 0.005}

해당 하이퍼파라미터를 가지는 모델은 은닉층이 1개인 모델로, 상대적으로 단순한 모델이다. 과적합 가능성은 낮아지지만, 복잡한 패턴을 학습하는데에는 한계가 있다.

해당 모델의 검증 데이터와 테스트 데이터에 대한 성능은 다음과 같다.

검증 Accuracy: 0.5983, BCR: 0.5092

테스트 Accuracy: 0.5690, BCR: 0.3333

ANN에서 검증 데이터보다 테스트 데이터에서의 BCR이 상당히 낮은 것으로 나타났다. 이 현상은 과적합이나 데이터를 샘플링하고, split하는 과정에서 학습/검증 데이터와 테스트 데이터의 분포가 달라졌을 가능성이 있음을 시사한다.

아래 질문들에 대해서는 Base Learner가 CART와 ANN인 경우 [Q1]에서 선택된 hyperparameter를 사용하여 실행하고 그 결과를 이용하여 답하시오.

[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?

n_estimators	Accuracy	BCR
1 (test)	0.6520	0.5835
10 (이하 val)	0.6165	0.5418
30	0.6365	0.5539
50	0.6455	0.5643
100	**0.6530**	**0.5679**
200	0.6525	0.5625
300	0.6475	0.5543

bootstrap 개수 또한 hyperparameter이므로 검증 데이터에 대해 성능을 비교했다. 각 Bootstrap 개수 별로, 검증 데이터에 대한 성능은 위와 같다. 100개를 사용한 경우가 Accuracy와 BCR 관점에서 가장 좋은 성능을 나타냈다.

n_estimators	Accuracy	BCR
1 (test)	0.6520	0.5835
100 (test)	0.6800	0.6052

Test dataset에 대한 단일 모형의 성능은 Accuracy 0.6520, BCR 0.5835로 나타났다. CART Bagging (100) 모델은 Accuracy가 0.6800, BCR이 0.6052로 나타나 단일 모델에 비해 성능이 향상되었음을 확인할 수 있었다. 이론적으로 단일 CART 모델은 높은 분산을 가지기 때문에, 훈련 데이터의 작은 변화에도 예측 결과가 크게 달라질 수 있다. Bagging을 통해 여러 CART 모델을 훈련시키고 이들의 예측을 결합하여 분산을 줄이므로. 개별 모델의 예측이 가지는 불안정성(높은 분산)을 상쇄시켜 더 안정적이고 일반화된 예측이 이루어졌다고 해석할 수 있다.

[Q3] 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.

Random Forest(랜덤 포레스트)는 앙상블 학습의 한 형태로, 여러 개의 의사 결정 트리(Decision Tree)를 만들어 합침으로써 작동하는 방법이다. 각각의 트리는 데이터의 부분 집합을 사용하여 학습되며, 분기 기준으로 랜덤한 feature를 사용하여 분할을 결정한다. 이론적으로 이를 통해 각 트리가 서로 다른 특성들에 대해 학습함으로써 트리 간의 상관관계를 줄이고, 모델의 일반화 성능을 향상시킨다.

마찬가지로 bootstrap 개수 또한 hyperparameter이므로 검증 데이터에 대해 성능을 비교했다. 각 Bootstrap 개수 별로, 검증 데이터에 대한 성능은 다음과 같다.

n_estimators	Accuracy	BCR
10	0.5955	0.5155
30	0.6210	0.5117
50	0.6375	0.5297
100	0.6445	0.5351
200	0.6435	0.5300
300	**0.6505**	**0.5400**

변수 중요도는 다음과 같다. 해당 도표에 대한 해석은 Q7-3에서 같이 하겠다.

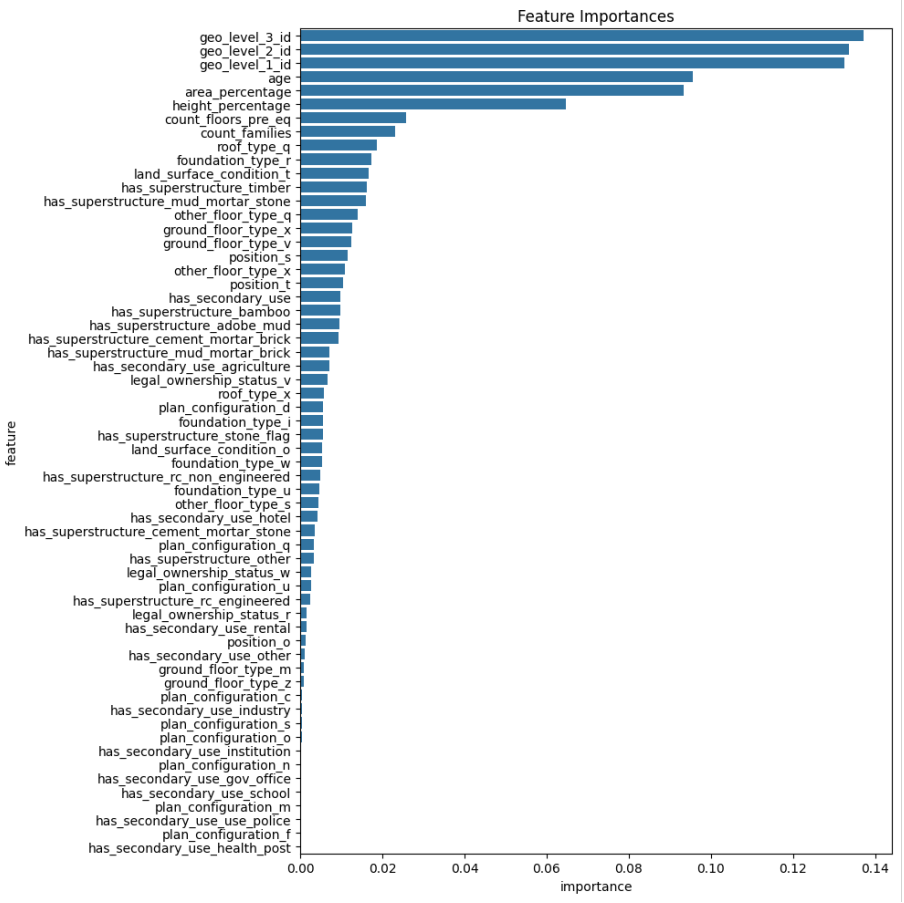


Figure 1 random forest feature importance

[Q3-1] 최적의 Bootstrap 수는 몇으로 확인되는가?

300개를 사용한 경우가 Accuracy와 BCR 관점에서 가장 좋은 성능을 나타냈다.

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

Random Forest는 Test 데이터에 대해서 Accuracy: 0.6795, BCR: 0.5672를 나타냈다.

n_estimators	Accuracy	BCR
CART Bagging(100)	**0.6800**	**0.6052**
Random Forest(300)	0.6795	0.5672

Accuracy 측면에서는 거의 비슷하지만, BCR에서 CART Bagging이 비교적 크게 뛰어난 것을 확인할 수 있었다.

[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 미친다고 볼 수 있는가?

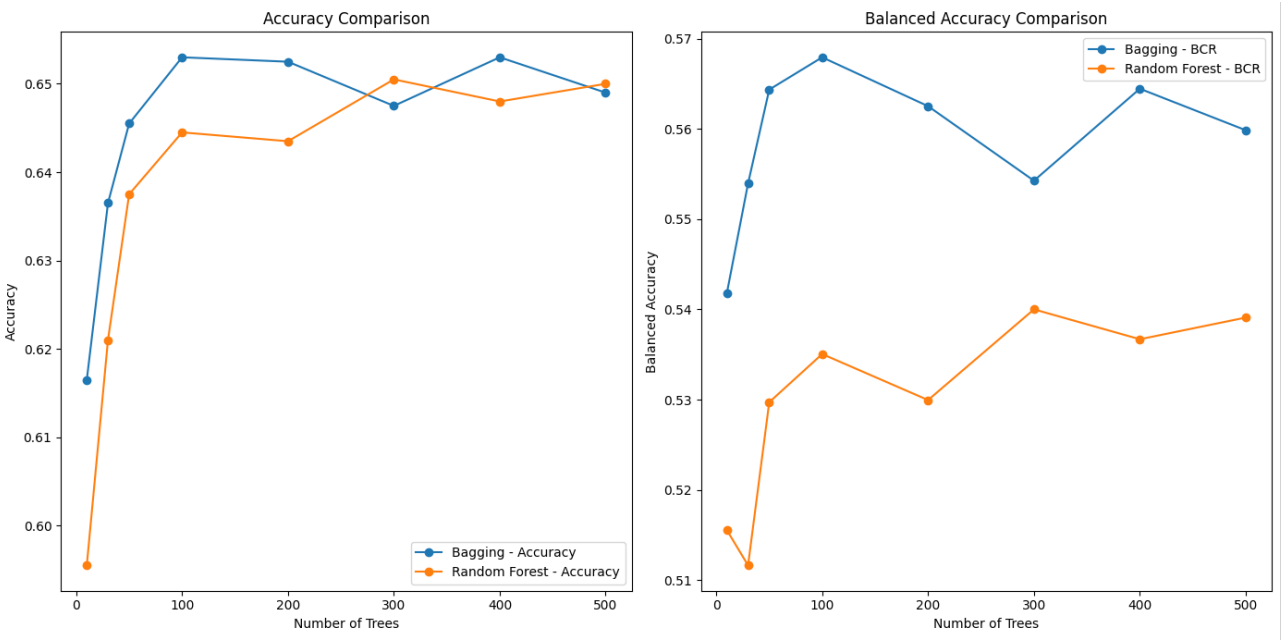


Figure 2 Performance by number of trees

위의 그래프를 통해 검증데이터에 대해서 약 100개까지는 두 모델 모두 성능이 좋아지나, 그 이후로는 비교적 비슷한 성능을 나타낸다. 특히 BCR에서는 CART bagging이 Random Forest보다 항상 높은 성능을 보임을 확인할 수 있었다.

Tree의 수는 모델의 성능에 어느정도 영향을 주는 것으로 결론지을 수 있다.

[Q4] [Q1]에서 찾은 최적의 hyperparameter를 이용하여 ANN 단일모형을 10번 반복하여 테스트 정확도를 평가해보시오. Accuracy와 BCR의 평균 및 표준편차를 기록하시오.

ANN은 초기 가중치 설정에 따라 성능이 크게 달라질 수 있다. 이는 학습 과정에서 최적의 가중치를 찾아가는 경로가 초기 가중치에 따라 달라지기 때문이다. 따라서 여러 번의 모델 학습을 통해 다양한 초기 가중치를 시도하고, 그에 따른 성능을 평가하는 것이 보다 일반화된 성능을 측정할 수 있는 방법이다.

10번의 모델 학습과 각 학습 모델에 대한 테스트 데이터 성능 지표를 표로 다음과 같이 나타냈다.

Run	Accuracy	BCR
1	0.5695	0.3338
2	0.6180	0.5028
3	0.5690	0.3333
4	0.6055	0.4594
5	0.5805	0.3579
6	0.5690	0.3333
7	0.6100	0.4299
8	0.5995	0.4190
9	0.6215	0.5042
10	0.6300	0.5029
Mean	**0.5973**	**0.4177**
Std	**0.0223**	**0.0698**

두 지표 Accuracy와 BCR 모두 표준편차가 낮게 나타나지만, Accuracy에 비해 BCR의 표준편차가 더 크게 나타났다.

BCR은 모든 클래스의 예측 성능을 평균한 값이므로, 소수 클래스에 대한 성능 변화에 민감하다. Class imbalance 데이터셋에서 소수 클래스에 대한 성능이 불안정할 때, BCR 지표의 변동폭이 커지게 된다. 반면에 Accuracy는 주로 다수 클래스의 예측 성능에 의존하므로, 상대적으로 안정적인 값을 보이게 된다고 해석했다.

[Q5] ANN Bagging 모델에 대해 다음 물음에 답하시오.

다음과 같은 bootstrap_loader 함수를 만들어 bootstrap을 구현했다. numpy.random.choice의 replace 매개변수를 True로 설정하여 복원 추출을 하도록 설정했다.

```
8 def bootstrap_loader(dataset, batch_size):
9     n_samples = len(dataset)
10    indices = np.random.choice(n_samples, n_samples, replace=True)
11    subset = Subset(dataset, indices)
12    return DataLoader(subset, batch_size=batch_size, shuffle=True)
```

Figure 3 Bootstrap data loader function

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

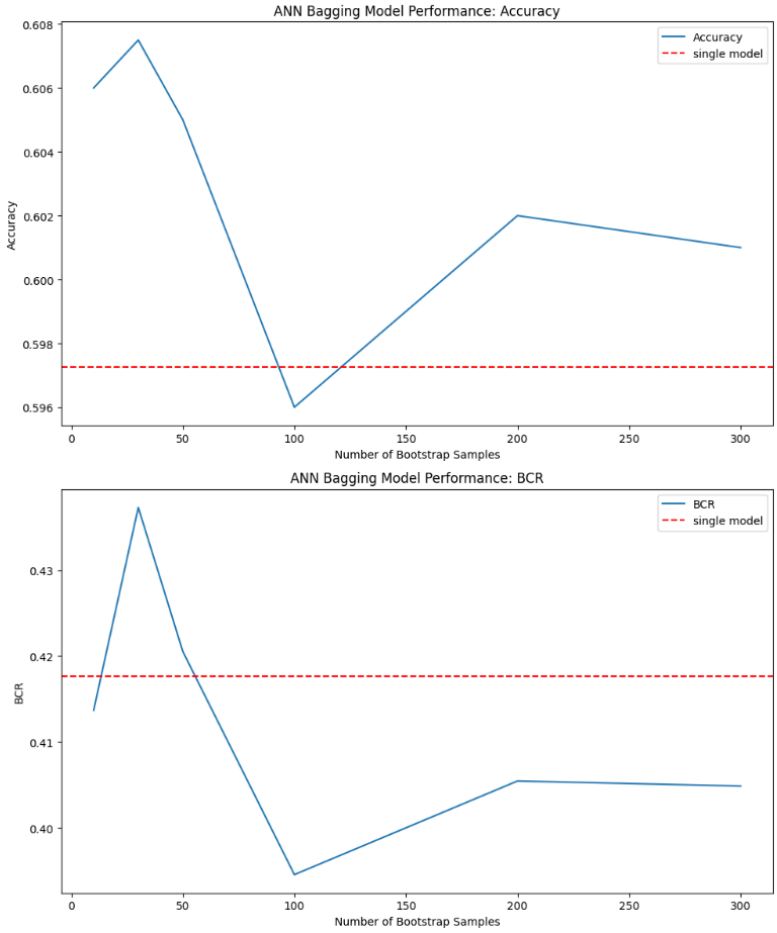
Bootstrap 수	Accuracy	BCR
10	0.6060	0.4137
30	**0.6075**	**0.4373**
50	0.6050	0.4205
100	0.5960	0.3945
200	0.6020	0.4054
300	0.6010	0.4049

개별 모델 학습에는 이전과 같이 patience 3의 early stop을 적용하여 과적합을 방지했다. 위의 표는 각 Bootstrap 개수 별로, 검증 데이터에 대한 성능이다.

[Q5-2] 최적의 Bootstrap 수는 몇으로 확인되는가?

Bootstrap이 30개일 때, Accuracy와 BCR 관점에서 가장 높은 성능을 보였다.

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



BCR 관점에서 약 30개, 50개인 경우 단일 모델에 비해 성능이 뛰어나고, 나머지의 경우에선 그렇지 않다는 것을 위의 그래프에서 확인 할 수 있다.

[Q6] Adaptive Boosting(AdaBoost)에 대해 다음 물음에 답하시오.

AdaBoost(Adaptive Boosting)는 앙상블 학습의 한 유형(boosting)으로, 약한 학습기(weak learner)를 이용하여 강력한 분류기를 만드는 방법이다. 각각의 약한 학습기는 이전 학습기의 오분류된 샘플에 더 많은 가중치를 부여하여 학습한다. 따라서 이전에 잘못 분류된 샘플들에 집중하여 새로운 학습기를 학습하게 된다. 이러한 반복적인 과정을 통해 각각의 약한 학습기는 데이터셋에 대해 점차 더 나은 성능을 보이게 된다. 마지막으로, 이러한 약한 학습기들을 결합하여 강력한 분류기를 만들어낸다. AdaBoost는 이론적으로 이상치(outliers)에 민감할 수 있지만, 일반적으로 좋은 성능을 보인다.

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

Adaptive Boosting의 Hyperparameter로는 다음의 요소들을 고려했다.

- N_estimators: 모델 수
    - 50, 100, 200, 300, 400, 500
- Learning_rate: 학습률
    - 0.0001, 0.001, 0.01, 0.1, 1, 10
- Algorithm: 학습 알고리즘
    - 'SAMME', 'SAMME.R'

학습 알고리즘 SAMME는 다중 클래스 문제를 다루기 위한 AdaBoost의 확장된 버전으로, 학습기의 오류율을 사용하여 가중치를 업데이트하고, SAMME.R은 예측 확률을 사용하여 보다 세밀한 가중치 조정을 가능하게 하는 SAMME의 변형이다.

검증 데이터로 찾은 hyperparameter 조합과 이 조합을 사용하여 테스트 데이터에 대해 예측한 결과는 다음과 같다.

Best hyperparameters: {'algorithm': 'SAMME.R', 'learning_rate': 1, 'n_estimators': 300}

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

Accuracy: 0.6515, BCR: 0.5807 로 기존 모델들의 성능과 비교해보면 다음과 같다.

모델	Accuracy (Mean)	BCR (Mean)
Multinomial Logistic Regression	0.5950	0.4729
CART	0.6520	0.5832
ANN	0.5973	0.4177
CART Bagging (100)	**0.6800**	**0.6052**
ANN Bagging (30)	0.6075	0.4373
Random Forest (300)	0.6795	0.5672
AdaBoost (300)	0.6515	0.5807

종합적으로 중간 정도의 성능을 보였다.

[Q7] Gradient Boosting Machine(GBM)에 대해 다음 물음에 답하시오.

Gradient Boosting Machine(GBM)은 트리 기반의 앙상블 학습 방법 중 하나다. 이 방법은 여러 개의 결정 트리를 순차적으로 학습시켜 이전 트리의 오차를 보완하는 방식으로 작동한다. 각 트리는 이전 트리에서의 예측값과 실제값 간의 차이, 즉 잔차(residual)에 대해 학습한다. 이전 트리에서 잘못 예측한 샘플들에 더 많은 가중치를 두고 새로운 트리를 학습함으로써 모델을 개선해 나간다. 이러한 방식으로 GBM은 이전 모델들의 오차를 줄이는 방향으로 학습하여, 최종적으로 강력한 앙상블 모델을 형성한다. GBM은 일반적으로 다른 앙상블 방법보다 높은 성능을 보이며, 대용량 데이터셋에서도 효과적으로 작동한다는 특징이 있다.

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

GBM의 Hyperparameter로는 다음의 요소들을 고려했다.

- n_estimators:
    - 50, 100, 200, 300, 400, 500
- Learning_rate: 학습률
    - 0.0001, 0.001, 0.01, 0.1, 1, 10
- Max_depth:
    - 3,4,5

검증 데이터로 찾은 hyperparameter 조합은 다음과 같다.

Best hyperparameters: {'learning_rate': 0.1, 'max_depth': 4, 'n_estimators': 300}

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

모델	Accuracy (Mean)	BCR (Mean)
Multinomial Logistic Regression	0.5950	0.4729
CART	0.6520	0.5832
ANN	0.5973	0.4177
CART Bagging (100)	**0.6800**	**0.6052**
ANN Bagging (30)	0.6075	0.4373
Random Forest (300)	0.6795	0.5672
AdaBoost (300)	0.6515	0.5807
GBM (300)	0.6850	0.6050

GBM은 테스트 데이터에 대해 Accuracy: 0.6850, BCR: 0.6050을 보였다. Accuracy와 BCR 관점에서 기존의 가장 뛰어난 모델인 CART Bagging (100)과 거의 비등한 성능을 보인다. 대신, 300개의 bootstrap을 사용한다는 점에서 더 적은 bootstrap을 사용하는 CART Bagging (100)이 computational cost관점에서 훨씬 좋은 모델이라고 생각한다.

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

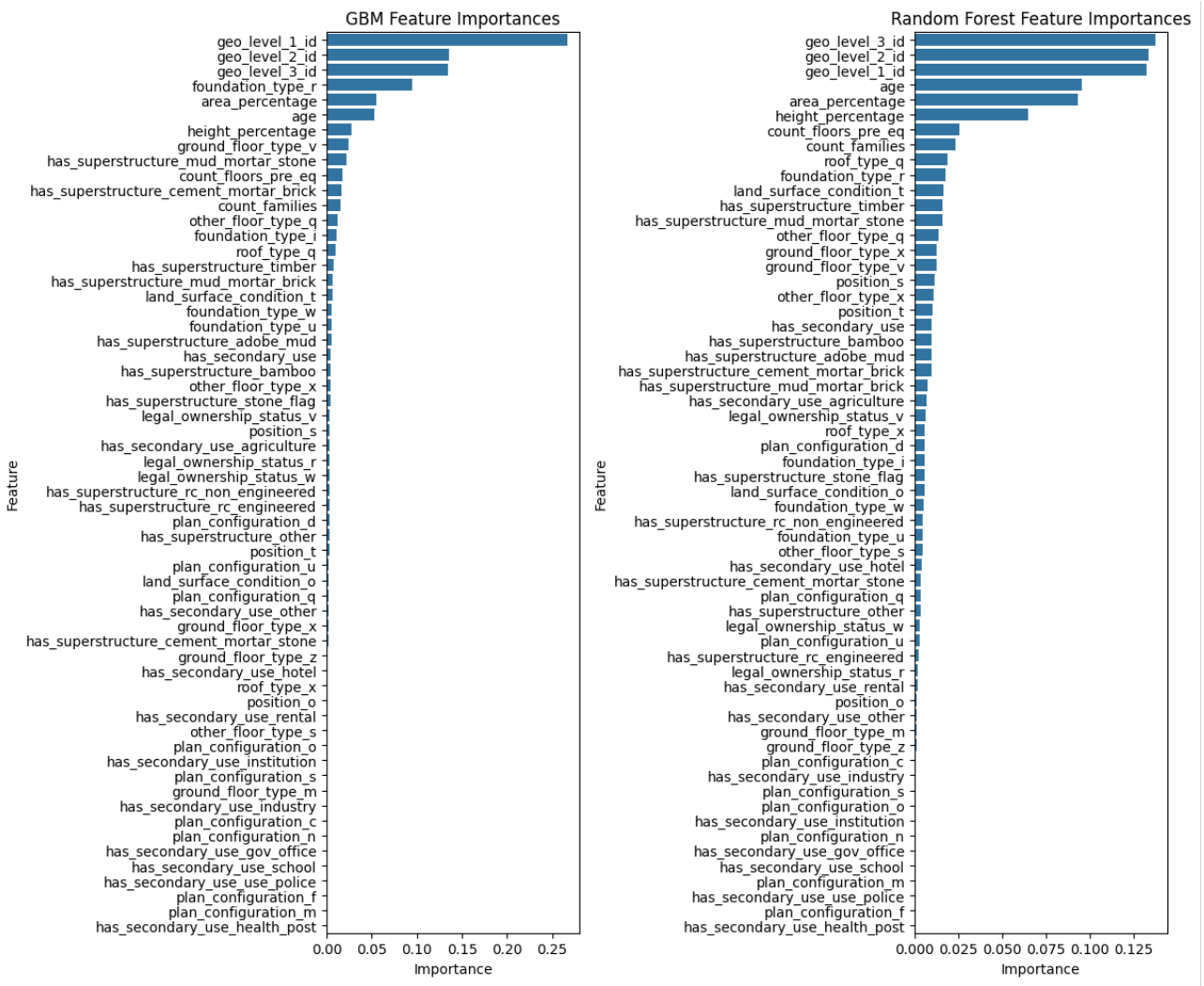


Figure 4 Feature Importance of GBM&RF

가장 눈에 띄는 특징은 다음과 같다.

1. **지리적 변수 (geo_level_1_id, geo_level_2_id, geo_level_3_id)**:

   두 모델 모두 지리적 변수를 가장 중요한 변수로 강조하고 있다. 이는 지리적 위치가 건물 손상 예측에 중요한 요인임을 시사한다.

2. **건물의 물리적 특성 (age, area_percentage, height_percentage):**

   age, area_percentage, height_percentage 등 건물의 물리적 특성도 두 모델에서 중요한 변수로 나타났다. 이는 건물의 구조적 특성이 예측에 중요한 영향을 미친다는 것을 보여준다.
   foundation_type_r와 같은 변수는 GBM에서 더 중요하게 나타났으며, 이는 특정 기초 유형이 건물의 내구성에 중요한 역할을 할 수 있음을 시사한다.

다만 모델 별로 각 변수의 중요도가 다르고, GBM이 더 좋은 성능을 보이는 것으로 보아, GBM의 변수 중요도가 좀 더 유의미하다고 판단할 수 있다.

[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?

모델	Accuracy (Mean)	BCR (Mean)
Multinomial Logistic Regression	0.5950	0.4729
CART	0.6520	0.5832
ANN	0.5973	0.4177
CART Bagging (100)	**0.6800**	**0.6052**
ANN Bagging (30)	0.6075	0.4373
Random Forest (300)	0.6795	0.5672
AdaBoost (300)	0.6515	0.5807
GBM (300)	0.6850	0.6050

지금까지의 결과를 요약하자면 다음과 같다.

주어진 테스트 데이터에서 BCR(Balanced Classification Rate) 관점에서 가장 우수한 분류 정확도를 나타내는 모델은 CART Bagging (100) 이다. 이 모델의 BCR 은 0.6052로, 다른 모델들과 비교했을 때 가장 높은 값을 보인다. GBM 또한 CART Bagging 모델과 거의 대등한 성능을 낸다.

대체로 CART Bagging, GBM, AdaBoost 등의 앙상블 기법들이 단일 모델보다 BCR이 높게 나타났다. 이는 여러 모델을 결합하는 앙상블 기법을 적용하면 과제에서 주어진 class imbalance 데이터를 단일 모델보다 효과적으로 다룰 수 있음을 보여준다.

ANN 모델의 경우 bagging을 적용했을 때, 크게 성능 향상이 일어나지 않았다. 이를 통해 Bagging의 주된 이점인 분산 감소 효과가 ANN에서는 크게 나타나지 않았다고 판단했다.

한가지 아쉬운 점은 시작할 때, box plot을 그리지 않고 scaler를 선택했다는 점이다. Appendix의 boxplot을 보면, height_percentage, age, area_percentage feature에 이상치가 존재함을 확인할 수 있다. 지금까지 진행한 실험을 robustscaler를 사용하여 다시 진행하면, 이상치에 민감한 AdaBoost의 성능이 개선될 수 있다고 추측해본다. 한편으로는 Robust scaler 또한 데이터의 분포가 정규 분포에 가까워야 왜곡이 일어나지 않는다는 특징이 있으므로, 극적인 개선은 일어나지 않을 것이라고 생각한다.

[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.

	Class 1	Class 2	Class 3
N. of instances	25,124	148,259	87,218
%	9.64%	56.89%	33.49%

파이썬에는 불균형 데이터셋을 다루기 위한 imblearn(imblanced-learn) 패키지가 존재한다. 가장 좋은 성능을 보인 CART Bagging (100)에 여러 방법론을 적용하여 검증 데이터에 대해 성능을 확인해보았다.

그 결과는 다음과 같다.

Method	Accuracy	BCR
SMOTE	0.6155	0.5701
BalancedBaggingClassifier	**0.5895**	**0.6439**
BalancedBaggingClassifier+SMOTE	0.6195	0.5737
RandomUnderSampler	0.5465	0.6392
SMOTEENN	0.5215	0.6258
RandomOverSampler	**0.6280**	**0.5816**
ADASYN	0.6255	0.5805
NearMiss	0.3925	0.5430

사용한 각 방법론에 대한 간략한 설명은 다음과 같다.

1. SMOTE (Synthetic Minority Over-sampling Technique)

SMOTE 는 소수 클래스 샘플을 인위적으로 생성하여 데이터의 균형을 맞추는 방법이다. 이 방법은 소수 클래스 샘플 사이의 선형 간격에 새로운 샘플을 생성하여, 원래 데이터의 분포를 유지하면서도 데이터 양을 늘린다.

2. BalancedBaggingClassifier

BalancedBaggingClassifier 는 부트스트랩 샘플링 과정에서 클래스 균형을 맞추도록 설계된 앙상블 기법이다. 이 방법은 각 부트스트랩 샘플에서 소수 클래스 샘플의 수를 늘리고 다수 클래스 샘플의 수를 줄여, 모델이 균형 잡힌 데이터에서 학습할 수 있도록 한다.

3. RandomUnderSampler

RandomUnderSampler 는 다수 클래스 샘플의 수를 무작위로 줄여서 클래스 균형을 맞추는 방법이다. 이 방법은 다수 클래스 샘플의 일부를 제거하여 소수 클래스 샘플과의 비율을 맞춘다.

4. SMOTEENN

SMOTEENN 은 SMOTE 와 Edited Nearest Neighbors (ENN) 방법을 결합한 조합 샘플링 방법이다. 먼저 SMOTE 로 소수 클래스 샘플을 증가시킨 후, ENN 으로 노이즈와 애매한 샘플을 제거하여 데이터셋을 정제한다.

5. RandomOverSampler

RandomOverSampler 는 소수 클래스 샘플의 수를 무작위로 증가시켜 클래스 균형을 맞추는 방법이다. 이 방법은 소수 클래스의 샘플을 반복적으로 복제하여 전체 데이터셋의 균형을 맞춘다.

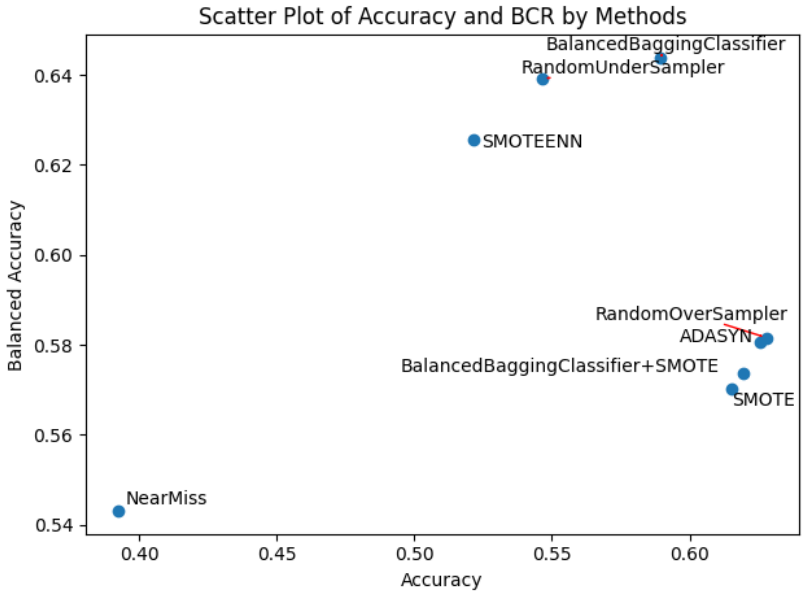
6. ADASYN (Adaptive Synthetic Sampling)

ADASYN 은 SMOTE 의 변형으로, 소수 클래스 샘플의 밀도가 낮은 지역에 더 많은 새로운 샘플을 생성한다. 이를 통해 모델이 학습하기 어려운 영역에서 소수 클래스를 더 잘 학습할 수 있도록 한다.

7. NearMiss

NearMiss 는 다수 클래스 샘플의 수를 줄이는 언더샘플링 방법 중 하나다. NearMiss 는 소수 클래스와의 거리 기반으로 다수 클래스 샘플을 선택하여, 소수 클래스와 가장 가까운 다수 클래스 샘플을 남기고 나머지를 제거한다.

각 방법론을 검증 데이터 셋에 적용한 결과를 Accuracy 와 BCR 에 대해 시각화하면 다음과 같다.



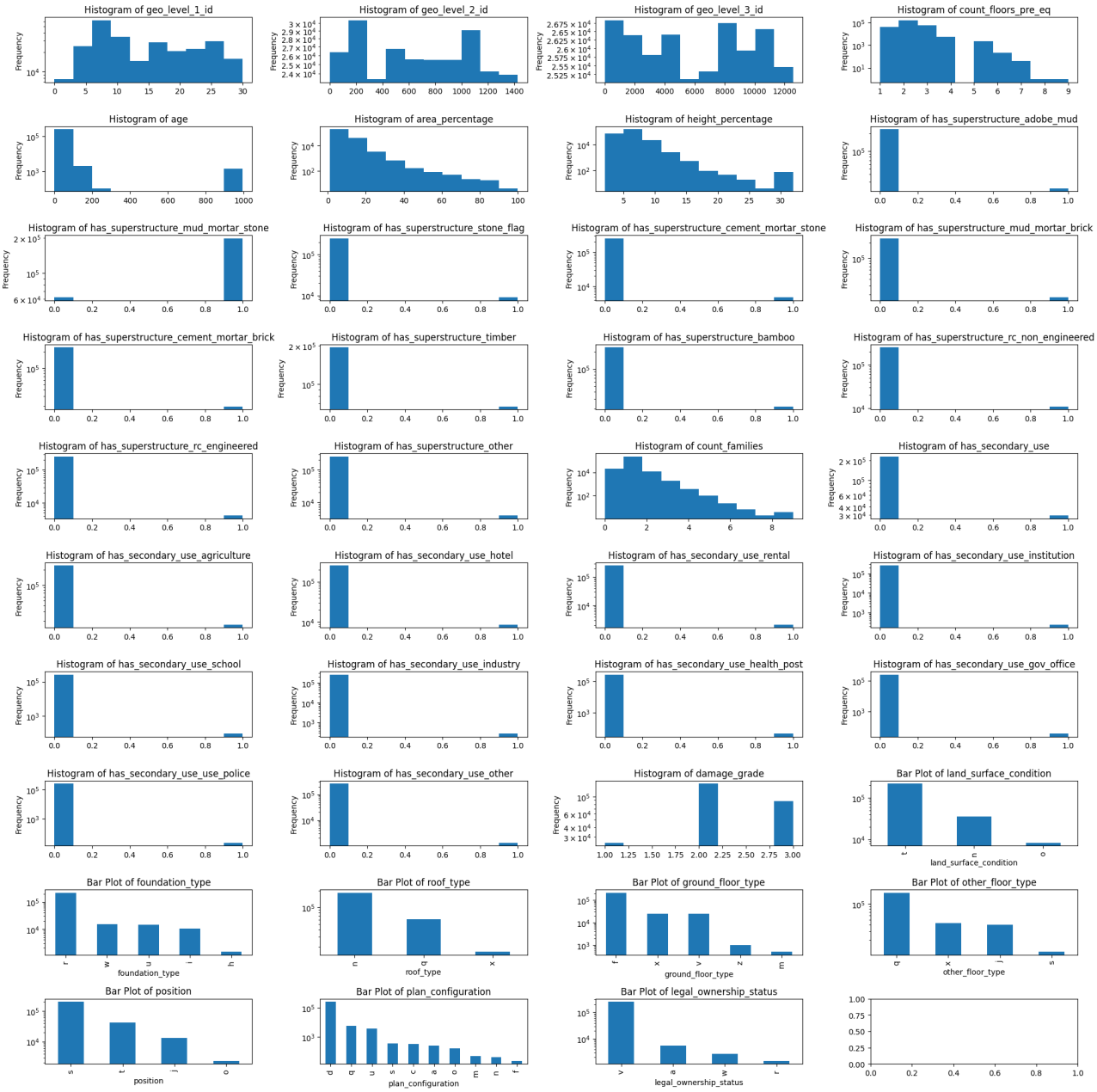
대체로 BalancedBaggingClassifier 가 BCR 과 Accuracy 모두 균형잡힌 성능을 보여줬다. 해당 방법론을 테스트 데이터에 적용한 결과 Accuracy 0.6345, BCR 0.6819 로 BCR 관점에서 압도적인 성능을 보여줬다.

모델	Accuracy (Mean)	BCR (Mean)
Multinomial Logistic Regression	0.5950	0.4729
CART	0.6520	0.5832
ANN	0.5973	0.4177
CART Bagging (100)	0.6800	0.6052
ANN Bagging (30)	0.6075	0.4373
Random Forest (300)	0.6795	0.5672
AdaBoost (300)	0.6515	0.5807
GBM (300)	0.6850	0.6050
CART BalancedBagging (100)	**0.6345**	**0.6819**

전체 모델 중 테스트 데이터에 대해 가장 높은 BCR 을 보이므로 BalancedBagging 을 사용하여 class imbalance 를 고려한 학습이 효과가 있었음을 확인할 수 있다.

Appendix

1. distribution of features



2. boxplot of features