2024 - 1

다변량분석

ENSEMBLE

담당 교수: 강필성

강의명: 다변량분석

제출자: 정동은

전공: 경영학과

학번: 2020120120

제출 날짜: 2024-06-11

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 variable을 생성하시오.

데이터 셋은 260,601개의 행과 40개의 변수로 이루어져있습니다. 독립변수는 39개이고, 그 중 첫 번째 변수인 'building_id'는 순서만을 의미하므로, 학습에는 의미없는 변수입니다. 따라서 이를 삭제하고 학습을 진행하였습니다. 또한 결측치 확인 결과 결측치가 존재하지 않았습니다. 다음으로 pd.get_dummies 함수를 사용하여 명목형 변수들을 이진형 변수로 변환하였다.

최종적으로 68개의 독립변수와 1개의 종속변수로 학습을 진행하였다.

[2] 전체 데이터셋을 임의로 150,000 개의 빌딩이 포함된 Training dataset 과 50,000 개의 Validation dataset, 그리고 60,601 개의 Test dataset으로 구분한 뒤 다음 각 물음에 답하시오. 분류 성능을평가/비교할 때는 3-class classification의 Accuracy 와 Balanced Correction Rate (BCR)을 이용하시오.

```
# Separate the features and target variable
X = df.drop('damage_grade', axis=1)
y = df['damage_grade']

X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=110601, stratify=y, random_state=2023)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=60601, stratify=y_temp, random_state=2023)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
X_val = scaler.transform(X_val)

X_train_val = np.concatenate((X_train, X_val), axis=0)
y_train_val = np.concatenate((y_train, y_val), axis=0)
```

제시된 코드로 데이터셋을 분할하였습니다. 코드의 마지막 두 줄인 X_train_val과 y_train_val은 최적의 하이퍼파라미터 조합을 찾은 후 해당 조합의 성능을 테스트 데이터셋으로 확인할 때 사용했습니다. 즉, 학습과 검증 데이터셋을 결합하여 모델을 학습한 후, 테스트 셋으로 실험을 진행할 때 사용한 코드입니다.

모든 모델의 성능은 언급된 것처럼 3-class classification의 Accuracy와 BCR만을 바탕으로 비교하였습니다. 최적의 하이퍼파라미터는 데이터 불균형 문제로 인해 BCR 성능을 기준으로 선정했습니다. 종속 변수의 클래스는 1, 2, 3의 세 가지로 이루어져 있으며, 클래스 2의 비율이 56.89%로 나타났습니다. 이는 학습을 진행하지 않고 모두를 2로만 예측하더라도 56.89%의 결과를 얻을 수 있음을 의미합니다. 따라서 다른 클래스들도 잘 예측하는지 종합적으로 고려하기 위해 BCR을 기준으로 최적의 하이퍼파라미터를 선정하였습니다.

Q1.

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

Multinomial logistic regression, Decision Tree, Artificial Neural Network를 각각 학습한 다음, 성능을 종합적으로 비교하겠습니다

1. Multinomial logistic regression (MLR)

LogisticRegression(multi_class='multinomial', solver='newton-cg')은 괄호에서 언급한 하이퍼파라미터로 학습을 진행하였다. 이때의 결과는 다음과 같다.

Confusion Matrix on Test data set

Actual / Prediction	1	2	3
1	1339	4450	54
2	977	32191	1308
3	70	18499	1713

Performace on Test data set

Model	Accuracy	BCR
MLR	0.581	0.416
CART	-	-
ANN	-	-

모든 클래스를 2로만 예측해도 0.569의 Accuracy를 얻을 수 있습니다. MLR의 Accuracy는 이와 비슷한 0.582로 약간 높은 수준이지만, 성능이 좋다고 보기 어렵습니다. BCR 또한 0.416으로, 첫 모델이라 비교 대상은 없지만 모든 클래스를 적절히 예측했다고 하기 힘듭니다.

Confusion matrix를 보면 class 1, 2, 3에 대한 예측이 균등하게 이루어지지 않았음을 확인할 수 있습니다. 특히 class 2는 다른 클래스에 비해 정답을 많이 맞췄습니다. 이는 class 2의 데이터가 가장 많아 상대적으로 학습이 잘 이루어진 결과입니다. class 1과 3도 class 2로 잘못 예측한 값이 많은 것은 class 2 데이터의 비중이 높아 학습이 bias되었기 때문으로 보입니다.

결론적으로, class 2를 집중적으로 학습한 결과 Accuracy는 class 2로만 예측했을 때와 큰 차이가 없었고, 이로 인해 BCR 성능도 낮게 나왔습니다.

2. Decision Tree: CART

Decision Tree는 Full Tree로 학습을 한 다음 Pre-pruning을 진행하는 방향으로 학습하였습니다. 먼저 Pre-pruning에서 가장 좋은 성능을 보이는 하이퍼파라미터 조합을 확인해한 결과입니다.

CART hyperparameter used in model

Hyperparameter	1	2	3
max_depth	10	20	30
min_sampels_split	10	30	100
min_samples_leaf	20	40	-

총 18가지의 조합 중 BCR을 기준으로 내림차순 정렬한 결과는 다음과 같다.

	Parameters	ACC	BCR
0	{'max_depth': 30, 'min_samples_leaf': 20, 'min...	0.70814	0.624898
1	{'max_depth': 30, 'min_samples_leaf': 20, 'min...	0.70800	0.624758
2	{'max_depth': 20, 'min_samples_leaf': 20, 'min...	0.70586	0.624473
3	{'max_depth': 20, 'min_samples_leaf': 20, 'min...	0.70576	0.624373
4	{'max_depth': 30, 'min_samples_leaf': 40, 'min...	0.71384	0.623462
5	{'max_depth': 30, 'min_samples_leaf': 40, 'min...	0.71380	0.623422
6	{'max_depth': 30, 'min_samples_leaf': 20, 'min...	0.71222	0.623005
7	{'max_depth': 20, 'min_samples_leaf': 20, 'min...	0.70976	0.622532
8	{'max_depth': 20, 'min_samples_leaf': 40, 'min...	0.71136	0.621738
9	{'max_depth': 20, 'min_samples_leaf': 40, 'min...	0.71134	0.621726
10	{'max_depth': 30, 'min_samples_leaf': 40, 'min...	0.71348	0.618139
11	{'max_depth': 20, 'min_samples_leaf': 40, 'min...	0.71120	0.616664
12	{'max_depth': 10, 'min_samples_leaf': 20, 'min...	0.68292	0.574842
13	{'max_depth': 10, 'min_samples_leaf': 40, 'min...	0.68530	0.574800
14	{'max_depth': 10, 'min_samples_leaf': 40, 'min...	0.68530	0.574800
15	{'max_depth': 10, 'min_samples_leaf': 20, 'min...	0.68288	0.574704
16	{'max_depth': 10, 'min_samples_leaf': 40, 'min...	0.68518	0.574189
17	{'max_depth': 10, 'min_samples_leaf': 20, 'min...	0.68336	0.572499

BCR을 기준으로 최적 하이퍼파라미터를 선택한 이유는 세 개의 클래스에 대해 균형 있는 예측 성능을 확인하기 위해서입니다. 클래스 불균형이 존재할 때는 F1-Score가 더 적합하지만, 이번 문제에서는 정확도와 BCR만을 제시하였기 때문에 클래스 균형을 고려한 BCR을 중점적인 지표로

사용하였습니다. Optimal Hyper parameter는 아래와 같습니다.

'max_depth': 30, 'min_samples_leaf': 20, 'min_samples_split': 10

이 조합이 최적의 성능을 보인 이유는 다음과 같을 것 같습니다. max_depth를 30으로 설정함으로써 트리가 적절히 깊어지도록 하였습니다. 이를 통해 데이터의 복잡성을 잘 반영할 수 있기때문입니다. 하지만 min_samples_leaf: 20, min_samples_split: 10은 과적합의 위험을 높일 수 있습니다.

하지만 그럼에도 선택된 가장 큰 이유는 원-핫 인코딩을 통해 독립변수의 수가 증가했기 때문이라고 생각됩니다. 원-핫 인코딩으로 인해 독립변수의 수가 많아지면서 데이터의 복잡성이 증가하였고, 이를 처리하기 위해 트리의 깊이를 충분히 깊게 설정하는 것이 필요했습니다. 그렇기에 낮은 높은 max depth, 낮은 min sample leaf, min sample split을 지닌 모델이 높은 BCR과 Accuracy를 달성할 수 있었던 거 같습니다.

Confusion Matrix on Test data set

Actual / Prediction	1	2	3
1	2833	2863	147
2	1646	28092	4738
3	171	7687	12424

Performace on Test data set

Model	Accuracy	BCR
MLR	0.582	0.416
CART	0.716	0.637
ANN	-	-

MLR과 비교하여 Accuracy, BCR 모두 성능이 개선되었습다. 특히, BCR의 성능이 더 많이 개선되었는데 이는 class 2를 잘 예측한 것만이 아니라, class 1, class 3의 성능을 잘 예측하는 경향도 증가하여 이루어진 성과라고 생각됩니다. Accuracy는 class 2만 잘 예측하여도 어느정도 보장되기 때문입니다. 그리고 Class 2로 잘못 예측하는 비율이 감소했기 때문입니다. MLR 모델에서는 Class 1이 Class 2로 잘못 예측되는 경우가 많았지만, CART 모델에서는 이러한 오류가 줄어들었습니다. 이는 Decision Tree가 데이터의 구조를 더 잘 학습하여 클래스 간의 경계를 명확히 구분할 수 있었기 때문입니다. 결론적으로, CART 모델은 Class 1과 Class 3을 정확하게 예측하여 전체적인 성능이 향상되었습니다. 이는 Accuracy와 BCR 모두에서 개선된 결과로 나타났습니다. MLR 모델에서는 Class 2로 잘못 예측되는 비율이 높았으나, CART 모델에서는 이 비율이 크게 감소하여 보다 균형잡힌 예측 성능을 보여주었습니다. 이를 통해 CART 모델이 MLR 모델에 비해 데이터의 패턴을 더 잘 학습하고, 클래스 간의 차이를 효과적으로 구분할 수 있음을 확인할 수 있습니다.

3. Artificial Neural Network (ANN)

ANN에서의 하이퍼파라미터는 다양하지만, 컴퓨팅 파워 및 시간적 제약으로 Learning rates, # of hidden layers, # of hidden nodes만을 고려하였습니다.

- # of Hidden nodes는 데이터에 따라 좋은 성능을 발휘하는 개수가 달라지기에, K-fold를 통해서 hidden nodes의 수에 따른 성능을 먼저 확인하였습니다.

Grid Search로 다른 하이퍼파라미터와 함께 탐색하는 것도 가능하지만, hidden nodes의 범위가 늘어날수록 Grid Serarch의 조합은 기하급수적으로 늘어나고, 이는 학습 시간의 엄청난 증가를 의미하기에 5 ~50까지 5단위로 개수를 늘려가며 # of Hidden nodes의 성능을 먼저 확인해보았다. 결과는 다음과 같다.

	hidden	ACC	BCR
2	15.0	0.442427	0.431740
6	35.0	0.505720	0.401430
1	10.0	0.523440	0.401210
7	40.0	0.433433	0.395161
0	5.0	0.417760	0.391791
8	45.0	0.425953	0.365546
3	20.0	0.497833	0.358999
4	25.0	0.567740	0.346714
9	50.0	0.499507	0.343014
5	30.0	0.493493	0.341156

표의 결과를 바탕으로 hidden nodes의 수를 15, 25, 35로 설정하였습니다. 먼저, BCR 기준으로 크게 차이가 나는 구간은 첫 번째에서 두 번째로 넘어갈 때와 다섯 번째에서 여섯 번째로 넘어갈 때입니다. 이 두 구간 모두 약 0.03 정도의 BCR 성능 차이를 보였습니다. BCR을 중점적으로 고려하기로 하였지만, BCR 기준 상위 node의 Accuracy가 0.44로 낮은 편이었습니다

따라서, BCR 기준 상위 2개 node와 Accuracy 기준 상위 1개 node를 선정하여 최적의 hidden nodes 수를 결정하였습니다. 최종적으로 hidden nodes 수를 15, 25, 35로 설정하였으며, 이는 각각 BCR 기준 상위 2개와 Accuracy 기준 상위 node를 반영한 결과입니다.

ANN hyperparameter

Hyperparameter	1	2	3

Learning rates	0.01	0.05	0.1
# of hidden layers	3	5	7
# of hidden nodes	15	25	35

위의 하이퍼파라미터를 조합한 27가지 조합에 대해 batch size = 32, epochs=50으로 총 27가지 조합 중 BCR을 기준으로 내림차순 정렬한 결과는 다음과 같다.

Sorted Results DataFrame:

	learning_rate	hidden_layers	hidden_nodes	ACC	BCR
0	0.01	3	15	0.700000	0.696429
21	0.10	5	15	0.700000	0.687500
9	0.05	3	15	0.666667	0.665179
3	0.01	5	15	0.633333	0.638393
14	0.05	5	35	0.633333	0.616071
12	0.05	5	15	0.600000	0.602679
8	0.01	7	35	0.600000	0.593750
5	0.01	5	35	0.600000	0.593750
6	0.01	7	15	0.566667	0.571429
17	0.05	7	35	0.566667	0.566964
19	0.10	3	25	0.566667	0.566964
15	0.05	7	15	0.566667	0.566964
2	0.01	3	35	0.566667	0.553571
1	0.01	3	25	0.533333	0.526786
16	0.05	7	25	0.500000	0.513393
10	0.05	3	25	0.500000	0.508929
24	0.10	7	15	0.466667	0.500000
26	0.10	7	35	0.533333	0.500000
25	0.10	7	25	0.533333	0.500000
4	0.01	5	25	0.500000	0.486607
20	0.10	3	35	0.466667	0.473214
13	0.05	5	25	0.466667	0.464286

27가지 조합 중, 가장 좋은 성능을 보이는 Optimal hyper parameter는 다음과 같다.

Learning Rate: 0.01, # of Hidden Layers: 3.0, # of Hidden Nodes: 15.0

가장 좋은 성능을 보이는 하이퍼파라미터 조합을 기반으로 Train + Validation 세트에서 학습을 진행한 후, Test 데이터로 평가를 수행했습니다. 참고로, 이 과정에서는 추후 비교 대상이 될 Q4와 Q5의 ANN 모델과 동일한 기준을 설정하기 위해 epoch를 10으로 사용했습니다. epoch를 10으로 설정한 이유는, Q5에서 epoch가 작으면 학습이 제대로 이루어지지 않았고, 반대로 너무 크면 오랜 시간 동안 결과가 나오지 않았기 때문입니다. 따라서, 최적의 성능을 유지하면서도 실험을 효율적으로 수행하기 위해 epoch를 10으로 설정하였습니다.

이러한 이유로 Validation 세트를 사용했을 때보다 Test 세트를 사용하여 학습한 ANN 모델의 BCR 성능이 저하되었습니다. 그러나 Accuracy는 오히려 향상되었는데, 이는 모델이 Class 2에 집중적으로 학습한 결과로 보입니다.

학습 과정 중의 랜덤 요인도 일부 작용했겠지만, epoch 수가 50에서 10으로 줄어든 것이 가장 큰 영향을 미친 것으로 판단됩니다. 이러한 문제는 추후 Q4와 Q5의 결과와 비교할 때 동일한 기준을 설정하였기 때문에 큰 문제는 없을 것으로 판단됩니다.

Confusion Matrix on Test data set

Actual / Prediction	1	2	3
1	1723	4037	83
2	904	29705	3867
3	44	11573	8665

Performace on Test data set

Model	Accuracy	BCR
MLR	0.581	0.416
CART	0.708	0.632
ANN	0.662	0.528

ANN의 Accuracy, BCR 성능은 MLR보다는 좋고, Decision Tree의 CART보다는 좋지 못하다. MLR은 단순한 분류 기준을 설정한 반면, CART는 복잡한 Rule을 생성한다. ANN은 단순한 분류 기준보다는 좋은 성능을 보였지만, 원핫인코딩으로 데이터의 복잡성이 증가해서인지, CART보다는 좋지 못한 성능을 보였다.

Confusion Matrix 분석 결과, MLR과 CART 사이의 특성을 보여주었습니다. 즉, MLR과 CART와 마찬가지로 ANN 모델도 Class 1을 Class 3으로 잘못 예측한 경우는 거의 없었고, Class 3을 Class 1로 잘못 예측한 경우도 드물었습니다. 그러나 Class 1과 Class 3을 Class 2로 잘못 예측한 경우가 성능 차이에 영향을 미쳤습니다. ANN 모델은 MLR보다는 Class 2로 잘못 예측하는 경우가 적었지만, CART보다는 많았습니다.

4. 종합 설명

MLR(다중 선형 회귀)은 독립 변수와 종속 변수 간의 선형 관계를 가정하기 때문에 데이터 내에서 복잡한 비선형 패턴을 포착하는 데 한계가 있습니다. 이로 인해 세 가지 모델 중 가장 저조한 성능을 보였습니다. 반면, CART(분류 및 회귀 트리)는 데이터를 분할(splitting)하여 복잡한 비선형 관

계를 효과적으로 포착할 수 있습니다. 이러한 이유로 CART는 가장 우수한 성능을 보였습니다.

ANN(인공 신경망)은 데이터 내의 복잡한 패턴을 잘 식별하지만 ANN 모델이 CART보다 상대적으로 낮은 성능을 보인것은 계산 시간의 제약으로 인해 배치 크기(batch size)와 epoch 수를 적게 설정했기 때문인 것으로 보입니다. 하지만 epoch수를 높게 할 경우 CART보다 높은 성능을 보일 것으로 보이지만, 정확한 판별을 위해서는 epoch 수를 실험하여 그 성능을 판별해야 합니다.

Q2.

CART의 Bagging 모델을 Bootstrap의 수를 30 부터 30 단위로 300까지 증가시키면서 분류 정확도를 평가해보시오. 최적의 Bootstrap 수는 몇으로 확인되는가? 이 모델은 단일 모형과 비교했을 때 성능의 향상이 있는가?

1. Bagging을 활용한 분류 정확도 평가

하이퍼파라미터 값은 Q1의 CART에서 가장 좋은 성능을 보였던 다음 조합을 선택하였다.

'max_depth': 30, 'min_samples_leaf': 20, 'min_samples_split': 10

Bootstrap의 수를 30~300까지, 30단위로 증가시키면 총 10번의 반복이 이루어진다. 10번의 반복에 대한 성능을 BCR을 기준으로 내림차순 정렬하면 다음 표와 같다.

# of Bootstraps	Accuracy	BCR
150	0.612	0.633
120	0.612	0.633
270	0.612	0.633
240	0.612	0.633
180	0.612	0.633
210	0.612	0.633
300	0.612	0.633
60	0.545	0.567
30	0.540	0.567
90	0.540	0.567

실험 결과,

1. Accuracy와 BCR의 순서 일치:

Accuracy와 BCR의 순서는 완벽하게 일치합니다. 이는 모델이 일관된 예측 성능을 보여주고 있음을 의미합니다.

2. Bootstrap 수와 성능:

Bootstrap 수에 관계없이 결과는 거의 일정합니다. 특히 Bootstrap 수가 120, 150, 180, 210, 240, 270, 300일 때 Accuracy와 BCR 값이 동일하게 나옵니다. 이는 Bootstrap 수를 늘려도 모델의 성능에 큰 영향을 미치지 않음을 보여줍니다.

3. Optimal Bootstrap 수:

BCR 기준으로 가장 좋은 성능을 보인 Bootstrap 수는 150, 120, 270, 240, 180, 210, 300으로, 모든 경우에 동일한 성능을 보였습니다. 이는 Bootstrap 수가 모델 성능에 미치는 영향이 적다는 것을 의미합니다.

CART의 Bagging 모델에서 Bootstrap 수를 늘린다고 성능이 향상되지는 않는다는 것을 확인할 수 있습니다. BCR과 Accuracy 모두 일정한 값을 보이며, Bootstrap 수가 60 이하일 때는 약간의 성능 저하가 나타났지만, 120 이상에서는 성능이 동일하게 유지되었습니다. 이는 개별 트리들이 다양성이 부족하여, Bagging이 variance를 줄이는 데 적합하지 않은 문제에서 발생한 것으로 생각됩니다.

따라서, Bootstrap 수를 150, 120, 270, 240, 180, 210, 300으로 설정하면 모델의 최적 성능을 유지할 수 있으며, 이는 반복횟수를 증가시키는 것이 성능 향상에 크게 기여하지 않음을 의미합니다.

2. 단일 모형과 비교

Optimal Bootstrap인 150으로 학습한 뒤, test set을 바탕으로 실험한 결과는 다음과 같습니다. 단일 모형의 결과는 Q1에서 얻은 CART의 test set을 바탕으로 실험한 결과입니다.

Confusion Matrix on Test data set

Actual / Prediction	1	2	3
1	2553	3225	65
2	1000	29605	3870
3	91	7697	12494

Performace on Test data set

Model	Accuracy	BCR
단일 모형	0.708	0.632
Bagging 모형	0.737	0.638

단일 모형에서는 Accuracy: 0.708, BCR: 0.632의 성능을 보였다. 반면 Optimal Bagging 모형 에서는 Accuracy: 0.737, BCR: 0.638의 성능을 보였습니다. 신뢰구간을 고려하지 않았지만, 성능 차이가 확연히 발생한다고 말하기에 애매한 수치이지만, variance 감소를 기저로 하는 Bagging 모형이기에, 단일 모형보다는 좋은 성능을 발휘하였습니다.

하지만 실험 결과에서도 보다시피, Bootstrap 수가 120 이상일 때 Bagging의 Bootstrap 수와 성능이 비례하지 않았습니다. 이는 CART 모델의 특징과 관련이 있다고 생각합니다. CART 모델은 greedy 알고리즘을 사용하여 error를 줄이는 방향으로 split 변수를 선택합니다. greedy로 인해 CART 모델을 Bagging하더라도 각 CART들은 높은 확률로 구조적 동일성을 지닐 것이기에, 이는 높은 상관관계를 지닌 채, voting을 바탕으로 예측을 수행할 것입니다. 따라서, Sub-model들간에 다양성이 존재하여야 성능 향상이 이루어질 수 있는데, 상관관계가 크니 다양성이 작다는 의미로 이는 별다른 성능 차이를 만들지 못합니다. 그렇기에 단일 모형과 Bagging 모형은 사실 비슷한 모형으로 학습을 진행하였기에 결과에 별다른 차이가 없을 수 있습니다. 물론 Bagging 모형에서 variance를 최소화하였기에 약간의 성능 개선은 있다고 볼 수도 있다.

Q3.

Random Forest 모델의 Tree의 수를 30 부터 30 단위로 300까지 증가시키면서 분류 정확도를 평가하고 다음 물음에 답하시오. 학습 과정에서는 변수의 중요도가 산출되도록 학습하시오.

1. 최적의 Bootstrap 수는 몇으로 확인되는가?

Tree의 수를 30~300까지, 30단위로 증가시키면 총 10번의 반복이 이루어진다. 10번의 반복에 대한 성능을 BCR을 기준으로 내림차순 정렬하면 다음 표와 같습니다.

# of Trees	Accuracy	BCR
150	0.633	0.620
120	0.633	0.620
270	0.633	0.620
300	0.600	0.589
240	0.600	0.585
180	0.600	0.585
210	0.567	0.554

30	0.567	0.550
90	0.567	0.550
60	0.500	0.487

Optimal Bootstrap인 150으로 학습한 뒤, test set을 바탕으로 실험한 결과는 다음과 같습니다.

Confusion Matrix on Test data set

Actual / Prediction	1	2	3
1	2686	2944	213
2	1657	27270	5549
3	211	8517	11554

Performace on Test data set

Model	Accuracy	BCR
Bagging 모형	0.685	0.608

최적의 Bootstrap 수는 150으로 확인되었습니다. 이는 Q2에서도 동일하게 최적의 Bootstrap 수로 선택된 값입니다. BCR 기준 상위 3개 항목은 Accuracy에서도 동일하게 상위 3개 항목이었으며, 모두 Tree 수가 많은 편에 속합니다(150, 120, 270). 이는 Tree 수가 증가할수록 전반적으로 ACC와 BCR이 증가하는 경향이 있다고 볼 수 있습니다.

그러나 Bootstrap 수가 300일 때 성능이 오히려 낮아지는 경향을 보였습니다. 이는 다음과 같은 이유 때문입니다. 첫째, 모델의 복잡도가 지나치게 증가하면서 과적합(overfitting) 현상이 발생할 수 있습니다. 과도하게 많은 트리를 사용하면 모델이 학습 데이터의 노이즈까지 학습하여 테스트 데이터에 대한 일반화 성능이 떨어질 수 있습니다. 둘째, 너무 많은 Bootstrap 샘플을 사용하면 생성된 트리들 간의 상관성이 높아져, 다양한 트리들로부터 얻을 수 있는 예측의 이점을 감소시키고, 결과적으로 모델의 성능을 제한할 수 있기 때문입니다.

결론적으로 해당 데이터셋과 모델에서 bootstrap이 너무 많거나 적은 경우보다는 중간 정도의 150이 최적의 성능을 보인다고 판단할 수 있습니다.

2. 최적의 Tree 수를 기준으로 이 데이터셋에 대해서는 CART Bagging 과 Random Forest 중에서 더 높은 분류 정확도를 나타내는 모형은 무엇인가?

CART와 RF에서의 최적의 Tree를 기준으로 가장 좋은 성능을 보인 결과는 다음과 같다.

Model	Accuracy	BCR
CART(Q2)	0.737	0.638
RF(Q3)	0.685	0.608

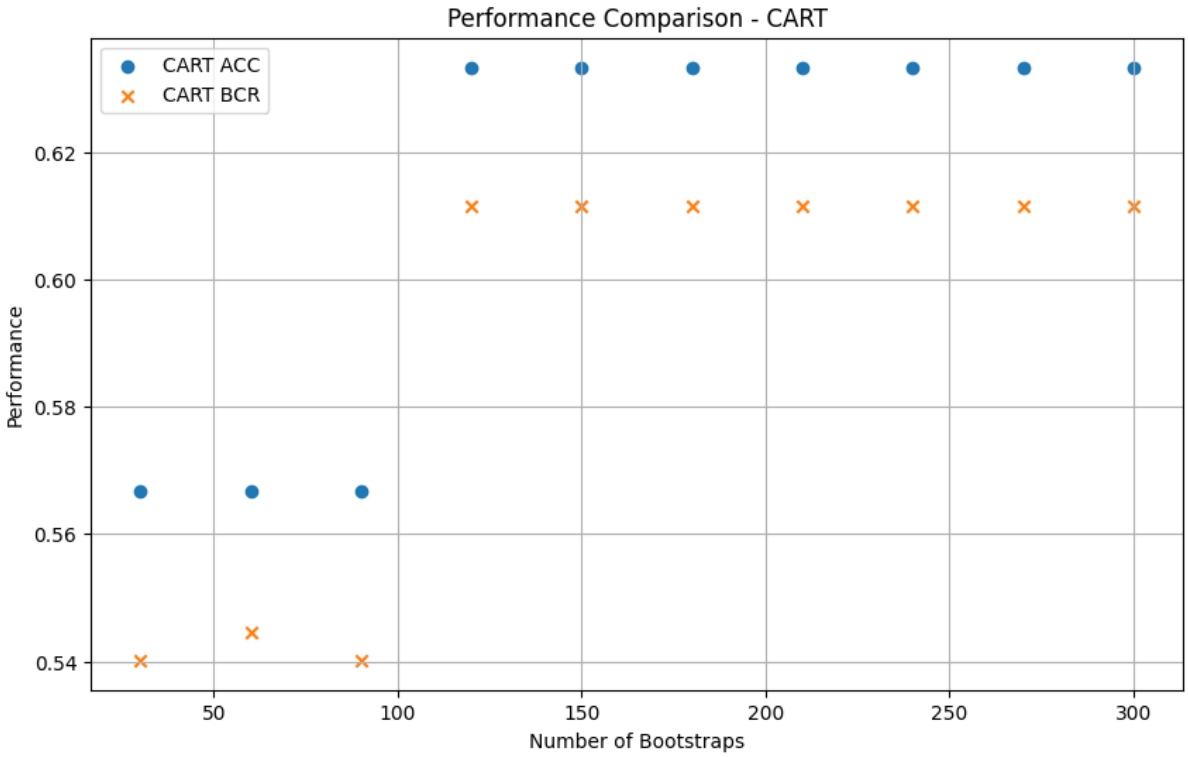
실험 결과를 얻기 전에는 Random Forest(RF)가 더 좋은 성능을 발휘할 것이라고 예상했습니다. Ensemble 기법은 다양성을 추가하여 성능을 향상시키는데, RF는 CART보다 더 높은 다양성을 가지고 있기 때문입니다. 하지만 실험 결과는 예상과 달랐습니다. 하이퍼파라미터 설정의 차이가 예상과 다른 결과를 초래했다고 생각됩니다.

RF에서 다양성을 추가하기 위해서는 Tree의 수를 늘려야 하지만, 컴퓨터 메모리 문제로 인해 n_estimator를 3 이상으로 설정할 수 없었기에, 이로 인해 모델의 다양성을 충분히 확보하지 못한 것으로 보입니다. 반대로 CART에서는 max_depth 등 3가지 하이퍼파라미터를 조정하여 Full Tree에 제약을 걸었습니다. 이는 CART 모델에 다양성을 추가하여, RF보다 더 좋은 성능을 발휘하게 만들었습니다.

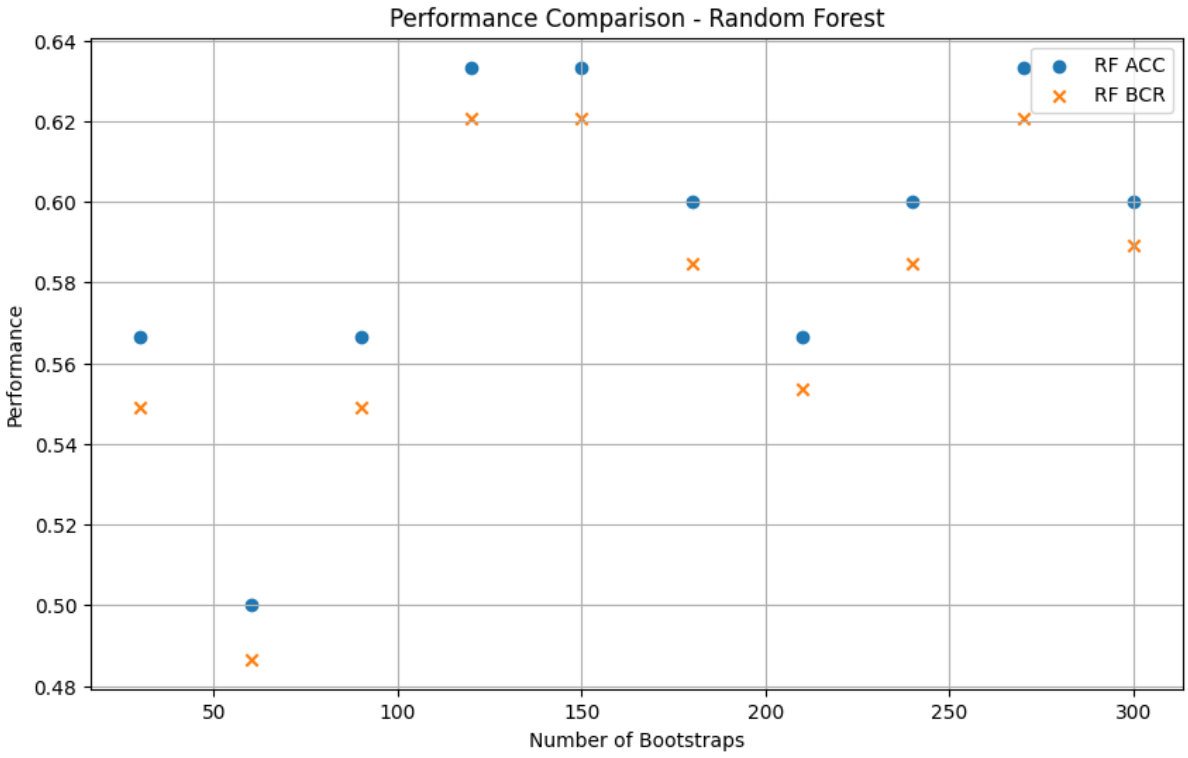
그렇기에 결과는 최적의 결과라고 할 수 없습니다. 두 모델의 하이퍼파라미터 설정에 따라 결과는 충분히 달라질 수 있기 때문입니다. 요약하자면, 해당 실험에서 Accuracy와 BCR 측면에서 더 좋은 성능을 발휘한 모델은 CART였습니다. 그 이유는 RF와 달리, CART에서의 다양성 추가로 Class 1과 3을 더 잘 학습하게 하여, Accuracy와 BCR 성능 향상을 이끌었다고 생각합니다.

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?

CART Bagging의 Bootstrap 별로 Accuracy, BCR 성능이 기록된 그림은 다음과 같습니다.



Random Forest의 Bootstrap 별로 Accuracy, BCR 성능이 기록된 그림은 다음과 같습니다.

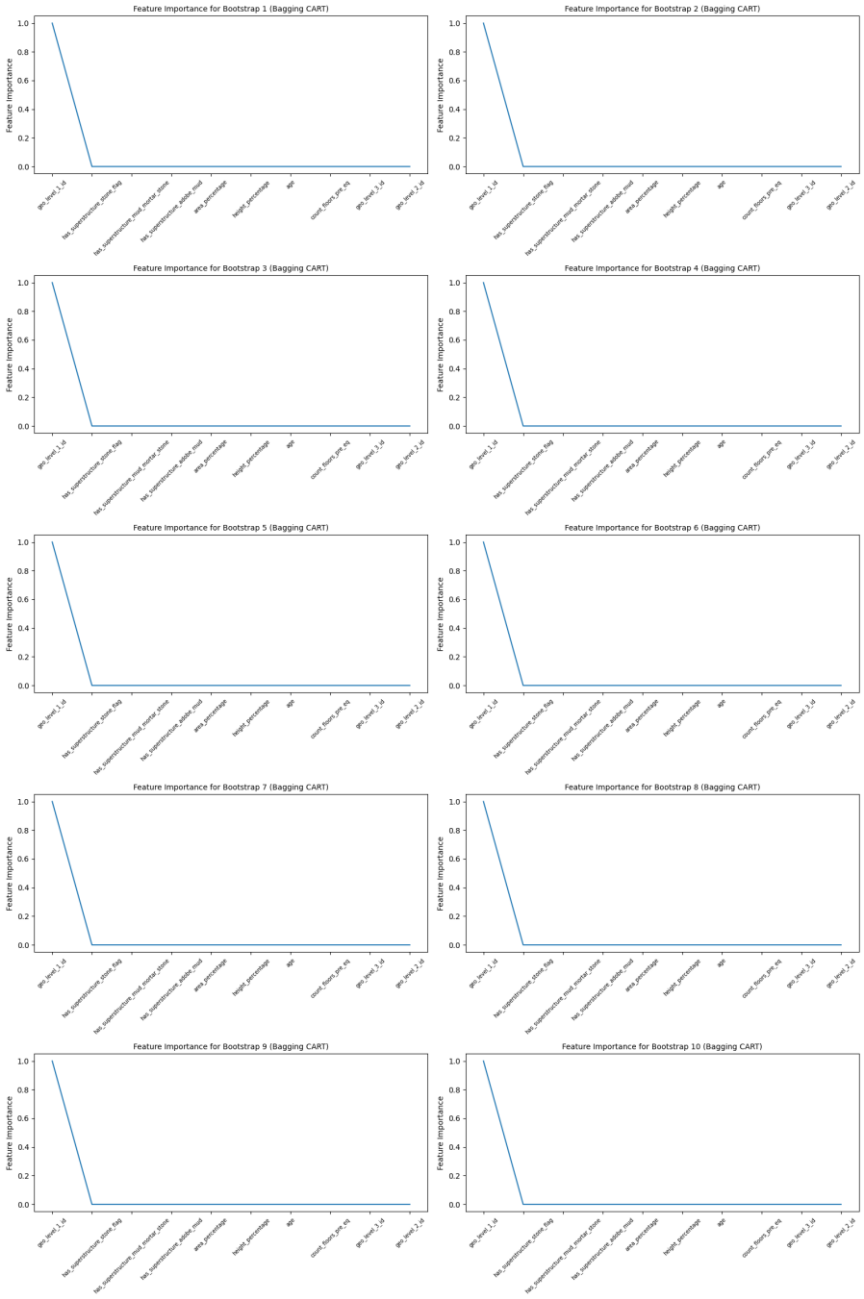


CART Bagging 모델에서는 Bootstrap 수가 증가함에 따라 성능이 안정적으로 유지되지만, Bootstrap 수가 일정 수준 이상에서는 큰 성능 향상이 없었습니다. 이는 CART 모델이 데이터의 구조를 잘 학습하고 있어, 추가적인 Bootstrap 샘플이 큰 변화를 주지 않는다는 것을 의미합니다.

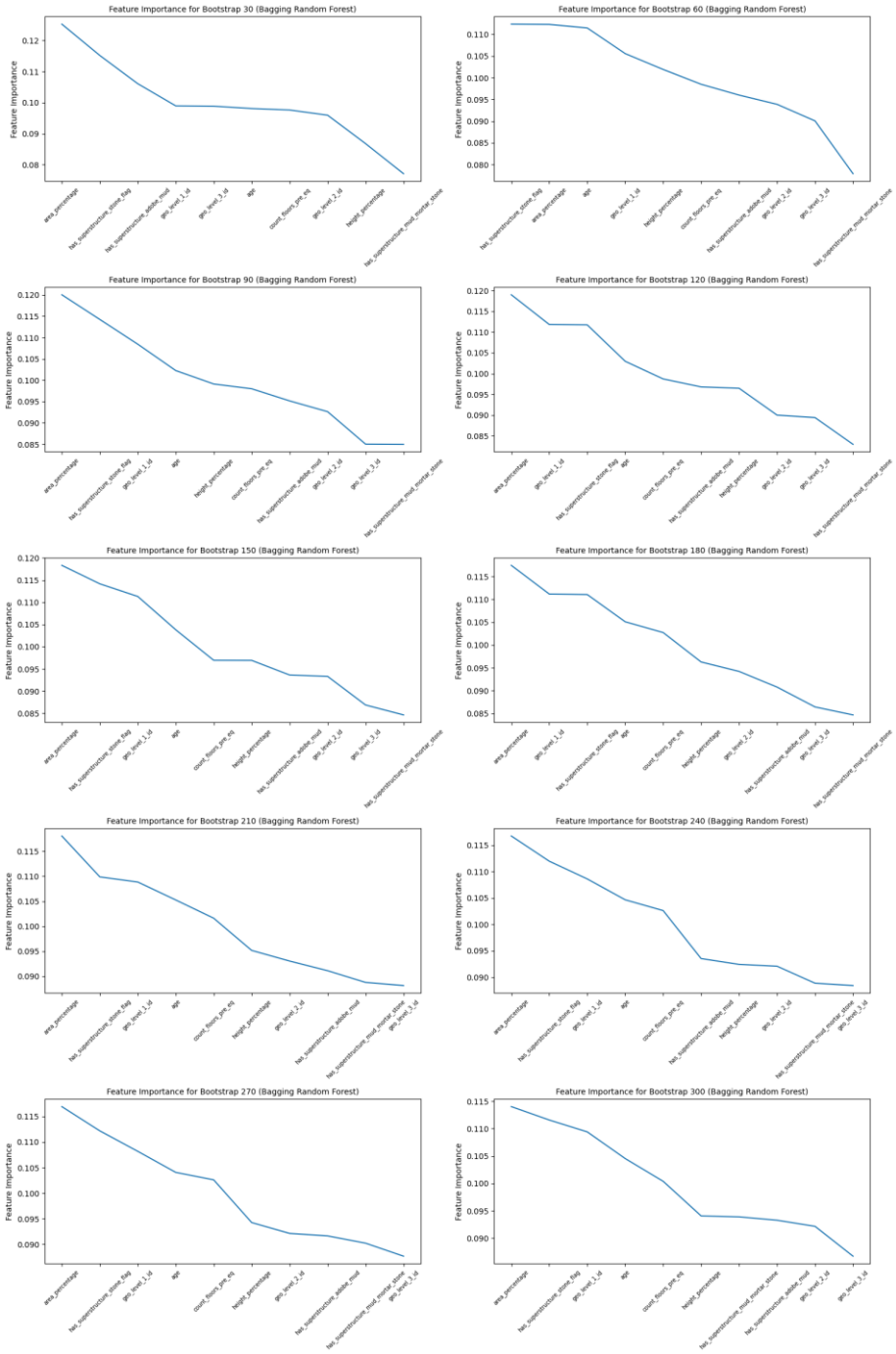
Random Forest 모델에서는 Tree 수가 일정 수준 이상일 때 성능이 증가하지만, 너무 많아지면 오히려 성능이 저하되는 경향이 있습니다. 이는 너무 많은 트리가 과적합을 초래하여 모델의 일반화 성능이 떨어지는 것으로 보입니다.

정리하자면, Cart의 경우 Bootstrap 또는 Tree 수가 일정 수준 이상일 때부터는 성능이 더 이상 크게 향상되지 않으며, RF의 경우 너무 많은 Tree 또는 Bootstrap 수는 오히려 성능 저하를 가져올 수 있지만 전반적으로는 우상향의 양상을 띄고 있습니다. 즉, tree의 수는 일정 수준 이상에서 성능에 영향을 미치지 못하거나 성능에 변동성이 있을 수 있지만, 그럼에도 tree의 수는 성능에 유의미한 향상을 가져다주는 경향성이 있다고 말할 수 있습니다.

다음으로 CART Bagging과 Random Forest Bagging의 변수 중요도입니다. 각 그래프는 Tree의 개수별로 변수 중요도가 표기된 그래프로, 우하향할 수록 Tree가 증가한 그래프입니다. 먼저 CART Bagging의 Tree 개수별 변수 중요도가 표기된 그래프입니다.



다음으로 Random Forest Bagging의 Tree 개수별 변수 중요도가 표기된 그래프입니다.



Cart의 경우, 변수 중요도와 변수 중요도 순서 변화 측면에서, Tree의 개수에 상관 없이 일정한 패턴을 보이고 있습니다. 그에 반해 RF의 경우 모델의 변수 중요도가 Cart와 달리 Tree 개수에 따라서 변수 중요도가 변하고 있으며, 그리고 특정변수들이 지배적인 위치를 차지하기 보다는 비교적 고르게 중요도를 가지고 있습니다. 이는 RF가 여러 개의 Tree를 조합하여 다양한 변수를 고르게 고려하려하는 특성때 문인 것으로 보입니다. 이를 통해, Bagging CART 모델에서는 주요 변수에 집중하고, Random Forest 모델에서는 다양한 변수를 고려하여 모델을 최적화함을 알 수 있습니다.

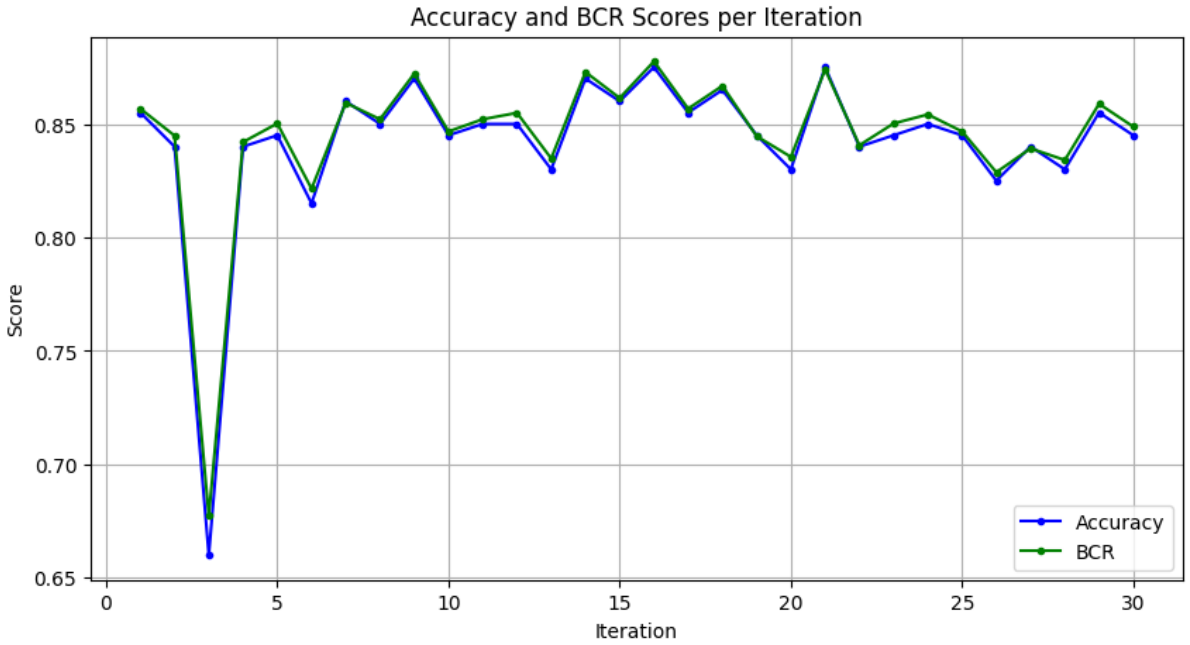
Q4.

Q1]에서 찾은 최적의 hyperparameter를 이용하여 ANN 단일모형을 30번 반복하여 테스트 정확도를 평가해보시오. Accuracy 와 BCR의 평균 및 표준편차를 기록하시오.

Q1에서 찾은 ANN의 최적의 하이퍼파라미터는 다음과 같다.

Learning Rate: 0.01, # of Hidden Layers: 3.0, # of Hidden Nodes: 15.0

Train+validation set으로 학습한 후에, test set으로 Iteration 별로 Accuracy(blue line)와 BCR(green line)을 평가하여 plot한 결과는 다음과 같습니다.



Model	Accuracy	BCR
ANN (단일 모형)	0.662	0.528
ANN (30회 반복 모형)	0.842 (0.037)	0.845 (0.034)

초기에는 정확도와 BCR 점수가 높게 시작하지만, 4번째 반복에서 크게 떨어집니다. 이 하락 이후, 점수는 다시 상승하여 0.80에서 0.85 사이에서 안정화되며, 약간의 변동을 보입니다. 이러한 변동의 이유는 초기 학습 불안정성에 있을 것으로 보입니다. 초기 몇 번의 반복 동안 모델이 안정적으로 학습되지 않을 수 있으며, 특히 4번째 반복에서 발생한 큰 하락은 이후의 결과에서 보듯 일시적인 불안정성으로 보입니다.

결과적으로는 30번 반복한 ANN 모델은 단일 모델보다 더 높은 정확도와 BCR을 달성하였습니다. 초기 학습 불안정성과 데이터 분할에 따른 변동이 있었지만, 반복을 통해 이를 극복하고 최적의 성능을 달성한 것으로 보입니다.

Q5.

ANN Bagging 모델에 대해 다음 물음에 답하시오

1. Bootstrap의 수를 30 부터 30 단위로 300까지 증가시키면서 각 Boostrap 수마다 30 회 반복수행을 실시하여 Accuracy 와 BCR의 평균 및 표준편차를 각각 기록하시오.

	Bootstrap	Accuracy Mean	Accuracy Std	BCR Mean	BCR Std
0	30	0.58908	0.0	0.407861	0.0
1	60	0.59018	0.0	0.412443	0.0
2	90	0.58958	0.0	0.412174	0.0
3	120	0.58990	0.0	0.413764	0.0
4	150	0.58920	0.0	0.412034	0.0
5	180	0.58878	0.0	0.412362	0.0
6	210	0.58902	0.0	0.412954	0.0
7	240	0.58886	0.0	0.413098	0.0
8	270	0.58856	0.0	0.413439	0.0
9	300	0.58864	0.0	0.413298	0.0

Bootstrap	Accuracy Mean	Accuracy Std	BCR Mean	BCR Std
120	0.5899	0	0.413764	0
270	0.58856	0	0.413439	0
300	0.58864	0	0.413298	0
240	0.58886	0	0.413098	0
210	0.58902	0	0.412954	0
60	0.59018	0	0.412443	0
180	0.58878	0	0.412362	0
90	0.58958	0	0.412174	0
150	0.5892	0	0.412034	0

30	0.58908	0	0.407861	0

2. 최적의 Bootstrap 수는 몇으로 확인되는가?

```
Optimal Bootstrap Count: 120
Performance on Test Data:
Accuracy Score: 0.5880926057325787
BCR Score: 0.41495984675074543
Confusion Matrix for Optimal Bootstrap Count:
[[ 1219  4597    27]
 [  805 32554  1117]
 [   29 18387  1866]]
```

Optimal Bootstrap은 120으로 판별되었지만, Bootstrap 수에 상관없이 결과들은 일정한 경향성을 보이고 있습니다. Bootstrap 30을 제외하고는 Accuracy, BCR 모두 0.001 ~ 2의 차이만 보이며 비슷한 성능을 보이고 있습니다. 하지만 Bootstrap이 120일 때 Accuracy, BCR 모두 가장 높기에, 이를 Optimal Bootstrap으로 선정하였습니다.

3. 이 모델은 단일 모형의 30 회 반복 결과와 비교했을 때 분류 정확도 및 성능의 편차 측면에서 어떤 차이가 있는가?

Optimal Bootstrap인 120으로 test set으로 실험한 결과를 ANN Bagging 모형의 결과로 사용하였다. 3가지 ANN의 결과가 아래의 표에 기술하였습니다.

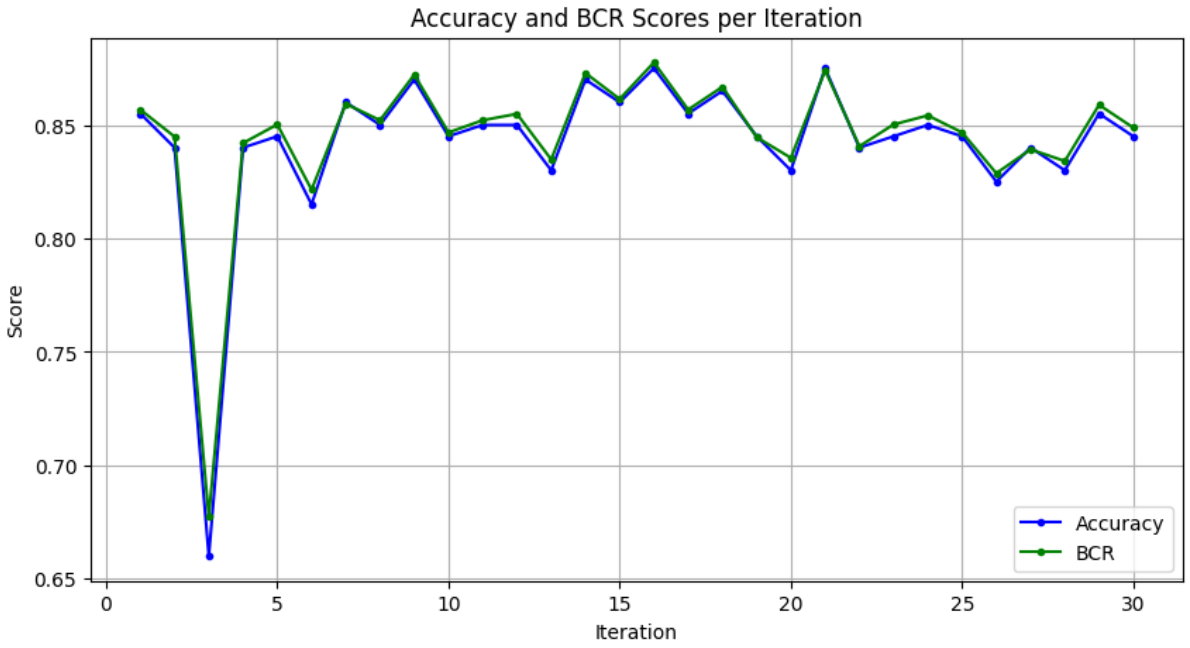
Model	Accuracy	BCR
ANN (단일 모형)	0.662	0.528
ANN (30회 반복 모형)	0.842 (0.037)	0.845 (0.034)
ANN (Bagging 모형)	0.588	0.415

실험 결과, 단일 모형과 반복 모형에서는 높은 성능을 보였으나, Bagging 모형의 성능은 예상보다 좋지 않았습니다. 이는 Ann Bagging 모형에서 추가된 tolerance 변수로 인한 것으로 보입니다. Bagging 모형에서 tolerance는 학습을 조기에 중단시키는 역할을 합니다. 이 실험에서는 tolerance를 0.01로 설정했는데, 이는 손실함수의 값 변화가 설정값보다 작을 때 학습을 멈추도록 하는 파라미터입니다. tolerance 값이 작을수록 학습이 엄격하게 멈추고, 클수록 빨리 멈추게 됩니다. Bagging 모형에서 tolerance를 0.01로 설정한 결과, 모델이 최적값에 도달하기 전에 학습이 멈추어 성능이 저하된 것으로 판단됩니다.

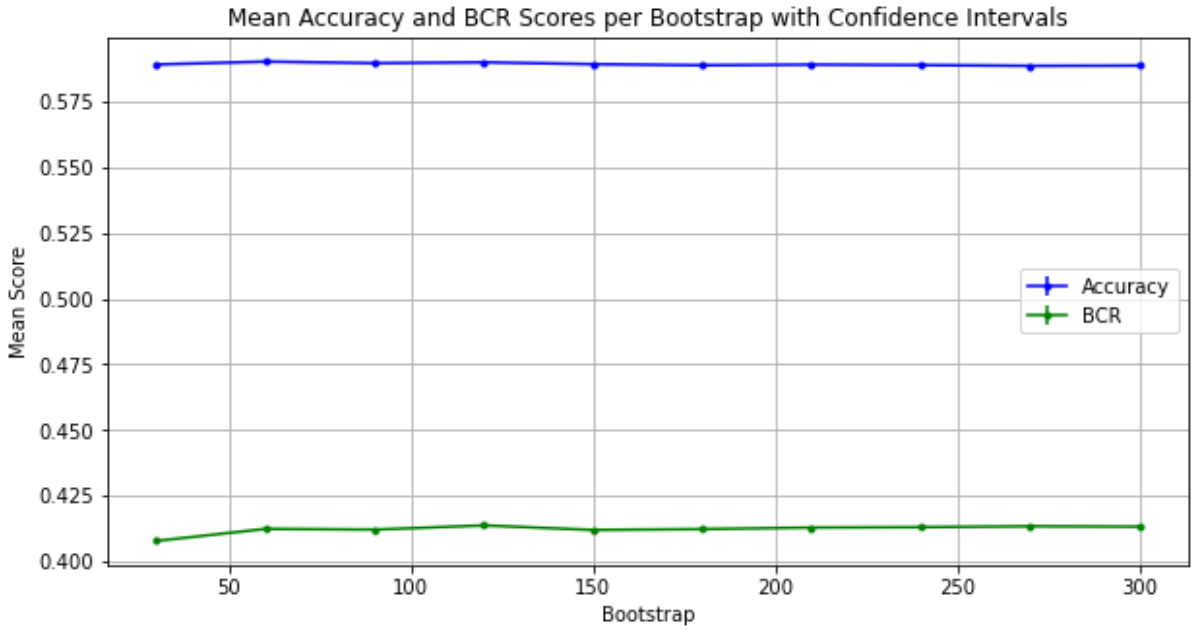
반복 모형의 경우, Accuracy와 BCR 모두 소수 셋째 자리에서 편차를 확인할 수 있었으나, Bagging 모형에서는 그렇지 않았습니다. 그렇다고 Bagging 모형이 반복 모형 보다 안정성이 높은 모델이라고 할 수는 없습니다. Bagging 모형은 모든 반복에서 예측이 일관되게 잘 맞지 않았기 때문에 편차가 거의 없었습니다. 이는 모델이 골고루 잘 맞추지 못했다는 의미입니다. 따라서 Bagging 모형의 낮은 성능과 작은 편차는 모델이 예측을 제대로 수행하지 못한 결과로 볼 수 있습니다.

결론적으로, Bagging 모형의 낮은 성능은 tolerance 설정으로 인해 모델이 최적화되지 못한 결과로 분석됩니다. 또한, Bagging 모형의 작은 편차는 예측 성능이 모든 반복에서 일관되게 낮았기 때문으로 판단됩니다. 향후 실험에서는 tolerance 설정을 포함한 하이퍼파라미터 최적화를 통해 모델 성능을 향상시킬 필요가 있습니다.

반복 모형의 성능 그래프는 다음과 같습니다.



Bagging 모형의 성능 그래프는 다음과 같습니다.



Q6.

Adaptive Boosting(AdaBoost)에 대해 다음 물음에 답하시오.

1. Hyperparameter 후보 값들을 명시하고, Validation dataset을 통해 최적의 hyperparameter 값을 찾아보시오.

base_estimator를 Decision Tree로 Adaboost를 진행하였다. 따라서 Decision Tree의 하이퍼파라미터를 사용하였습니다. 실험에 사용한 하이퍼파라미터는 다음과 같습니다.

Hyperparameter	특징
max_depth	10, 20, 30
'min_samples_split	10, 30, 100
min_samples_lea	20, 40

총 18가지 조합으로 실험을 진행하였고, 하이퍼파라미터와 validation set에 대한 BCR, Accuracy 성능이 표기된 결과는 다음과 같습니다.

	Parameter Settings	BCR Score	Accuracy Score
14	{'max_depth': 30, 'min_samples_split': 30, 'mi...	0.592660	0.69506
12	{'max_depth': 30, 'min_samples_split': 10, 'mi...	0.592660	0.69506
16	{'max_depth': 30, 'min_samples_split': 100, 'm...	0.586721	0.67680
6	{'max_depth': 20, 'min_samples_split': 10, 'mi...	0.585637	0.68572
8	{'max_depth': 20, 'min_samples_split': 30, 'mi...	0.585637	0.68572
15	{'max_depth': 30, 'min_samples_split': 30, 'mi...	0.577582	0.66770
13	{'max_depth': 30, 'min_samples_split': 10, 'mi...	0.577582	0.66770
17	{'max_depth': 30, 'min_samples_split': 100, 'm...	0.570647	0.65418
10	{'max_depth': 20, 'min_samples_split': 100, 'm...	0.568382	0.64722
5	{'max_depth': 10, 'min_samples_split': 100, 'm...	0.565776	0.63812
11	{'max_depth': 20, 'min_samples_split': 100, 'm...	0.564232	0.64288
3	{'max_depth': 10, 'min_samples_split': 30, 'mi...	0.562371	0.63966
1	{'max_depth': 10, 'min_samples_split': 10, 'mi...	0.562371	0.63966
4	{'max_depth': 10, 'min_samples_split': 100, 'm...	0.560182	0.63682
9	{'max_depth': 20, 'min_samples_split': 30, 'mi...	0.559350	0.64176
7	{'max_depth': 20, 'min_samples_split': 10, 'mi...	0.559350	0.64176
2	{'max_depth': 10, 'min_samples_split': 30, 'mi...	0.552366	0.63310
0	{'max_depth': 10, 'min_samples_split': 10, 'mi...	0.552366	0.63310

BCR 기준으로 가장 좋은 성능을 발휘한 조합은 다음과 같다.

'max_depth': 30, 'min_samples_split': 30, 'min_samples_leaf': 20

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

Confusion Matrix on Test data set

Actual / Prediction	1	2	3
1	2388	3272	183
2	1173	28424	4879
3	124	8316	11842

Performace on Test data set

Model	Accuracy	BCR

MLR	0.581	0.416
CART	0.708	0.632
ANN	0.662	0.528
CART Bagging	0.737	0.638
RF Bagging	0.685	0.608
ANN (30회 반복 모형)	0.842 (0.037)	0.845 (0.034)
ANN (Bagging 모형)	0.588	0.415
Adaboost	0.704	0.606

AdaBoost는 이번 실험에서 유일하게 Boosting을 사용한 모델로, 전체 모델 중 중간 정도의 성능을 발휘했습니다. 하지만 Decision Tree를 base estimator로 사용했음에도 불구하고, CART와 비교하여 성능 개선이 이루어지지 않았습니다. 이는 Decision Tree의 높은 분산과 낮은 편향 특성으로 인해 Bagging 기법이 더 적절하다는 점에서 기인합니다. 실제로 CART Bagging 모델의 성능이 이를 뒷받침합니다.

Confusion Matrix를 살펴보면, AdaBoost는 전체적으로 준수한 예측력을 보였으나, class 1, 2, 3 모두에서 예측력이 감소했습니다. 특히 class 2의 예측력 감소가 두드러집니다. 이전 모델들에서는 class 1과 3의 예측력 차이가 있었지만, class 2의 예측력은 크게 변하지 않았습니다. 이는 Boosting의 작동 원리 때문으로 생각됩니다. Boosting은 잘못 예측된 경우에 가중치를 부여하여 다음 라운드에서 정확도를 높이려 시도합니다. 주로 class 1과 3이 잘못 예측되는 경우가 많아, 이들의 정확도를 올리기 위해 class 2의 예측력이 감소한 것으로 판단됩니다.

단일 모형과 비교하자면, AdaBoost는 MLR, ANN보다는 좋은 성능을 보였지만, 동일한 하이퍼파라미터를 사용한 CART보다는 낮은 성능을 발휘했습니다. 이는 class 1과 3에 가중치를 부과하여 학습하면서 class 2의 예측 성능이 떨어진 결과입니다.

Ensemble 모형과 비교하자면 Ensemble 모형 중에서는 ANN, RF을 제외한 Bagging 모형보다 Accuracy와 BCR 성능이 모두 낮았습니다. 하지만 RF의 경우 컴퓨터 메모리 문제로 인해 n_estimator를 3 이상으로 설정할 수 없었기에, ANN의 경우 tolerance 설정이 있어 일반화에 어려움이 있기 때문에 비교 대상에서 제외해야 합니다,

정리하자면 AdaBoost는 Boosting계열의 모델로, 잘못 분류된 경우에 집중하여 새로운 분류 규칙을 만드는 것을 반복합니다. 그러나 AdaBoost는 잘못 분류된 데이터에 모델을 과적합할 위험이 있어 복잡한 모형보다는 단순한 모형이 효과를 가진다. 하지만 해당 데이터 셋은 그렇지 않기에, 따라서 class 1, 3에 과적합되어 class 2의 예측 성능을 떨어뜨려, 전체적은 성능이 저하된 것으로 생각됩니다.

Q7.

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

1. Hyperparameter 후보 값들을 명시하고, Validation dataset을 통해 최적의 hyperparameter 값을 찾아보시오.

GBM(Gradient Boosting Machine) 모델에서는 다음과 같은 하이퍼파라미터가 사용가능합니다.

Hyperparameter	특징
n_estimators	부스팅 단계 또는 결정 트리의 수를 결정
subsample	각 트리를 훈련하는 데 사용할 샘플의 비율
loss	부스팅 과정에서 최적화할 손실 함수
min_samples_split	드를 분할하는 데 필요한 최소 샘플 수
min_samples_leaf	리프 노드에 있어야 하는 최소 샘플 수
max_depth	의사 결정 트리의 최대 깊이

위의 하이퍼파라미터중 n_estimators, subsample, 그리고 loss를 변경해가며 실험을 진행하였다. 실험에 사용된 하이퍼파라미터 값은 다음과 같습니다.

Hyperparameter	특징
n_estimators	30, 50, 70
subsample	0.0.5, 0.7, 0.9
loss	deviance, log_loss

사용한 하이퍼파라미터는 처음 언급된 것이기에 설명을 추가하면 다음과 같습니다.

n_estimators: GBM 모델에서 구성할 부스팅 단계 또는 결정 트리의 수를 결정한다. 수를 늘림으로써 모델은 데이터에서 더 복잡한 관계를 확인할 수 있다. 실험에서는 n_estimators에 대해 30과 50, 70을 고려하였고, 이는 학습중에 반복적으로 구축될 Decision Tree의 수를 나타낸다.

subsample: GBM의 각 Tree를 학습하기 위해 무작위로 선택할 샘플의 비율을 제어하는 하이퍼파라미터이다. 1.0 미만의 값을 설정하면 훈련 과정에 무작위성이 도입되고 과적합을 방지하는 데 도움이 된다. 실험에서는 subsample에 대해 0.5, 0.7, 0.9을 사용하였다.

loss: 그래디언트 부스팅 과정에서 최적화할 손실 함수를 결정하는 하이퍼파라미터이다. 실험에서는 'deviance'와 'log_loss'의 두 가지 손실 함수를 사용하였다. ' deviance '는 분류 작업에 일반적으

로 사용되는 로지스틱 회귀 loss에 해당된다. 한편, 'log_loss'는 예측 확률과 실제 결과의 차이를 측정하는 log loss를 의미합니다.

총 18가지 조합으로 실험을 진행하였고, 파라미터, validation set에 대한 BCR, Accuracy 성능이 표기된 결과는 다음과 같습니다.

	Parameter Settings	BCR Score	Accuracy Score
15	{'n_estimators': 70, 'subsample': 0.7, 'loss':...	0.553954	0.67608
14	{'n_estimators': 70, 'subsample': 0.7, 'loss':...	0.553954	0.67608
13	{'n_estimators': 70, 'subsample': 0.5, 'loss':...	0.553868	0.67690
12	{'n_estimators': 70, 'subsample': 0.5, 'loss':...	0.553868	0.67690
17	{'n_estimators': 70, 'subsample': 0.9, 'loss':...	0.553448	0.67614
16	{'n_estimators': 70, 'subsample': 0.9, 'loss':...	0.553448	0.67614
10	{'n_estimators': 50, 'subsample': 0.9, 'loss':...	0.536602	0.66654
11	{'n_estimators': 50, 'subsample': 0.9, 'loss':...	0.536602	0.66654
9	{'n_estimators': 50, 'subsample': 0.7, 'loss':...	0.535448	0.66576
8	{'n_estimators': 50, 'subsample': 0.7, 'loss':...	0.535448	0.66576
7	{'n_estimators': 50, 'subsample': 0.5, 'loss':...	0.534936	0.66584
6	{'n_estimators': 50, 'subsample': 0.5, 'loss':...	0.534936	0.66584
5	{'n_estimators': 30, 'subsample': 0.9, 'loss':...	0.508054	0.65530
4	{'n_estimators': 30, 'subsample': 0.9, 'loss':...	0.508054	0.65530
3	{'n_estimators': 30, 'subsample': 0.7, 'loss':...	0.508031	0.65568
2	{'n_estimators': 30, 'subsample': 0.7, 'loss':...	0.508031	0.65568
1	{'n_estimators': 30, 'subsample': 0.5, 'loss':...	0.504813	0.65492
0	{'n_estimators': 30, 'subsample': 0.5, 'loss':...	0.504813	0.65492

BCR 기준으로 가장 좋은 성능을 발휘한 조합은 다음과 같다.

'n_estimators': 70, 'subsample': 0.7, 'loss': deviance or log_loss

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

최적의 하이퍼파라미터를 바탕으로 test set으로 실험한 결과는 다음과 같습니다.

Confusion Matrix on Test data set

Actual / Prediction	1	2	3
1	2156	3645	42
2	1107	30111	3258
3	91	11426	8765

Performace on Test data set

Model	Accuracy	BCR
MLR	0.581	0.416
CART	0.708	0.632
ANN	0.662	0.528
CART Bagging	0.737	0.638
RF Bagging	0.685	0.608
ANN (30회 반복 모형)	0.842 (0.037)	0.845 (0.034)
ANN (Bagging 모형)	0.588	0.415
Adaboost	0.704	0.606
GBM	0.677	0.558

단일 모형과 비교하자면, GBM은 단일 모형인 CART를 제외하고는 모든 모델보다 더 좋은 성능을 보였습니다. 이는 GBM의 Boosting 효과로 Bias를 줄였기 때문입니다.

Ensemble 모형과 비교하면, GBM은 ANN을 제외한 Bagging 모형보다 성능이 좋지 않았습니다. 같은 Boosting 계열인 AdaBoost보다도 낮은 성능을 보였습니다. GBM의 gradient를 찾아가는 과정이 Adaptive하게 찾아가는 AdaBoost보다 noise를 더 학습하여 과적합을 초래한 것이 성능 차이의 원인으로 보입니다.

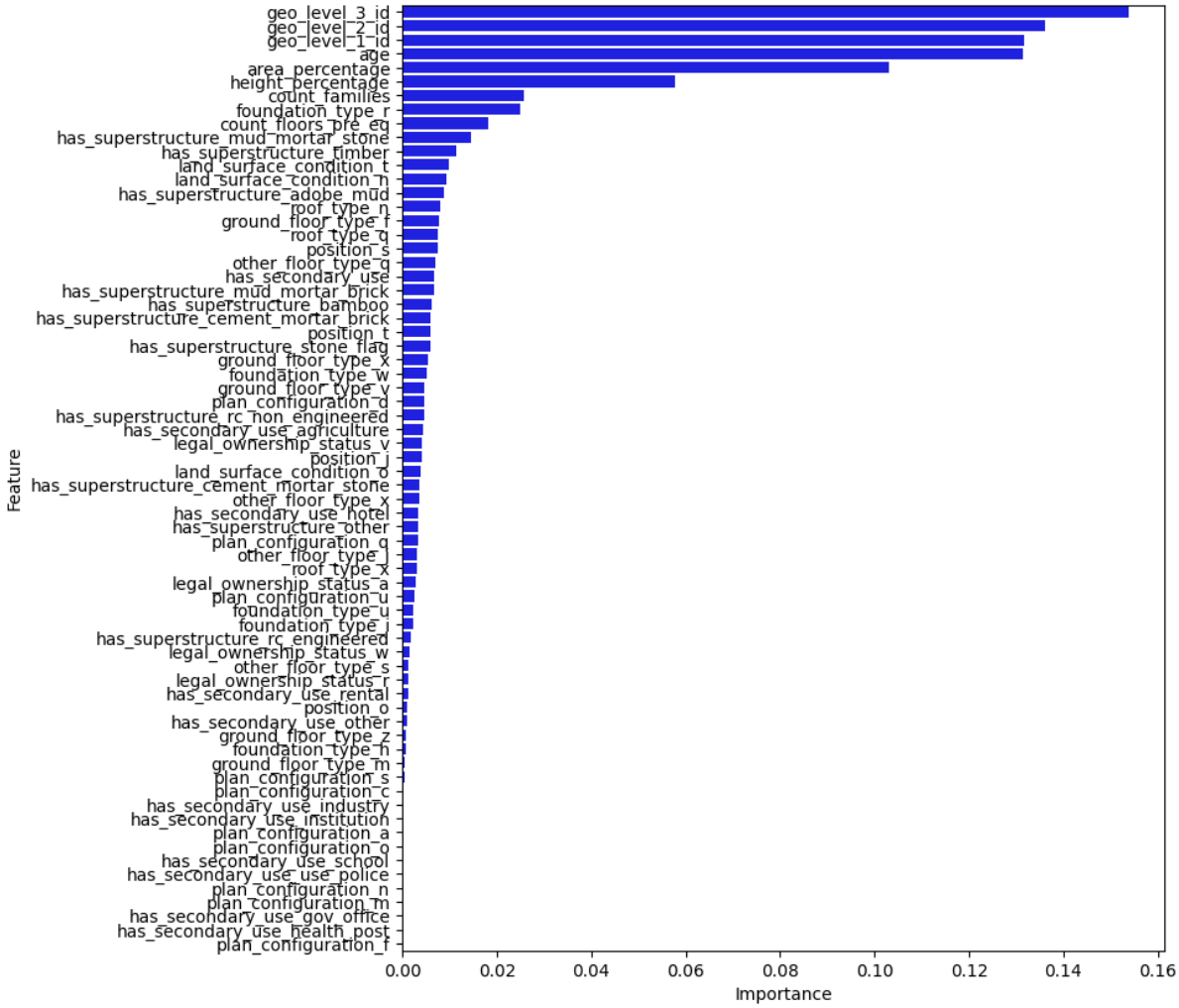
GBM은 class 2를 class 2로 정확하게 예측하는 경우가 30,000건 이상으로 크게 증가한 것으로 보아 class 2에 과적합되어 예측 성능이 떨어진 것으로 보입니다. 이는 하이퍼파라미터 설정 차이와 과적합 방지를 위한 정규화 방법(shrinkage, subsample)의 적용이 충분하지 않았기 때문입니다. 그

렇기에 Boosting보다는 Bagging이 이번 데이터셋에 더 적합한 방법임을 확인할 수 있었습니다.

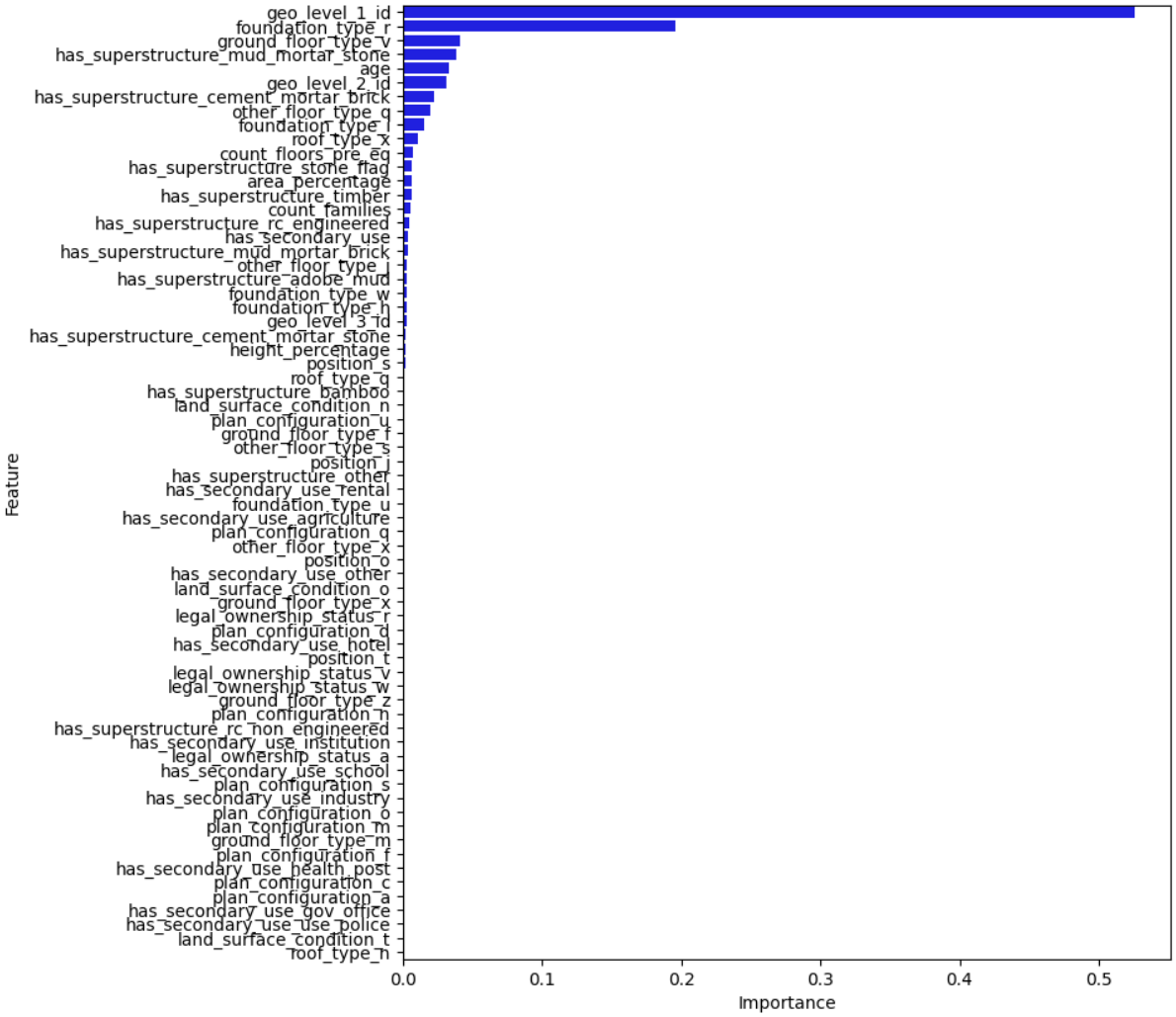
그리고 Bagging과 Boosting 결과를 종합하면, 이번 데이터셋은 Boosting보다 Bagging에서 더 좋은 성능을 발휘할 수 있는 데이터셋이라고 판단됩니다.

3. 산출된 변수의 중요도를 해석해보고, Random Forest 모델에서 산출된 주요 변수와 비교해보시오.

Random Forest 모델에서의 변수 중요도



GBM 모델의 변수 중요도



변수 중요도의 순위 비교

RF(Random Forest)와 GBM(Gradient Boosting Machine) 모델에서 식별된 첫 번째 차이는 중요한 변수의 순위가 다르다는 점입니다. RF에서는 geo_level_3_id가 가장 중요한 변수로 선정된 반면, GBM에서는 이 변수가 상위 5위 안에 들지 않았습니다. 두 모델에서 공통적으로 중요한 변수로 선정된 것은 geo_level_1_id와 age 두 가지뿐입니다. 이는 같은 데이터 셋을 사용하더라도 모델에 따라 중요한 변수가 달라질 수 있음을 보여줍니다.

이 차이는 Bagging과 Boosting의 차이로 설명할 수 있습니다. RF는 독립적인 Bootstrap 샘플에서 각각의 모델을 학습하고 이를 결합하는 반면, GBM은 반복적인 업데이트 과정을 통해 모델을 개선합니다. 따라서 RF는 각 변수의 중요도를 독립적으로 평가하는 반면, GBM은 이전 모델의 결과를 반영하여 특정 변수에 대한 중요도가 누적될 수 있습니다.

변수 중요도의 비율

그래프를 보면 GBM은 상위 6~7개의 변수가 전체 중요도의 대부분을 차지하고 있습니다. 반면, RF는 상위 변수에 중요도가 집중되어 있긴 하지만, GBM에 비해 하위 변수들도 일정한 중요도를 가지고 있습니다.

GBM이 특정 변수에 높은 중요도를 부여하는 이유는 Ensemble 방식의 차이에 있는 것으로 보입니다. GBM은 이전 모델의 결과를 다음 모델에 반영하여 업데이트하기 때문에, 특정 변수의 중요도가 계속 누적될 가능성이 큽니다. 반면 RF는 각 모델이 독립적으로 샘플링되기 때문에 다양한 변수를 고려할 기회가 더 많기에 이런 차이가 비롯된 것으로 보입니다.

즉, RF와 GBM의 변수 중요도 결과 차이는 두 모델의 학습 방식 차이로 설명할 수 있습니다. RF는 독립적인 샘플링과 결합을 통해 다양한 변수를 고려하는 반면, GBM은 반복적인 업데이트 과정을 통해 특정 변수에 집중하게 됩니다. 이로 인해 두 모델은 같은 데이터 셋에서도 다른 변수 중요도를 보일 수 있습니다.

Q8. 총 여덟 가지의 모델(Multi nomial logisticregression, CART, ANN, CART Bagging, ANN Bagging, Random Forest, AdaBoost, GBM) 중 BCR 관점에서 가장 우수한 분류 정확도를 나타내는 모형은 무엇인가?

Model	Accuracy	Accuracy 순위	BCR	BCR 순위
MLR	0.581	9	0.416	8
CART	0.708	3	0.632	3
ANN	0.662	7	0.528	7
CART Bagging	0.737	2	0.638	2
RF Bagging	0.685	5	0.608	4
ANN (30회 반복 모형)	0.842 (0.037)	1	0.845 (0.034)	**1**
ANN (Bagging 모형)	0.588	8	0.415	9
Adaboost	0.704	4	0.606	5
GBM	0.677	6	0.558	6

BCR 관점에서 가장 우수한 분류 정확도를 나타내는 모형은 "ANN (30회 반복 모형)"으로, BCR 값이 0.845로 가장 높았습니다.

- **Bagging 계열 모형 비교**

Bagging 계열에서는 CART Bagging(BCR 0.638, 순위 2위), RF Bagging(BCR 0.608, 4위), ANN Bagging(BCR 0.415, 순위 9위), 로 결과가 나왔습니다. CART Bagging의 경우, 모델이 높은 상관관계를 가진 채로 결합되어 성능이 개선되지 않을 것으로 예상했지만, Random Forest보다 높은 성능을 보였습니다. 이는 하이퍼파라미터 설정의 차이로 인해 CART Bagging이 모델의 분산을 더 효과적으로 줄였기 때문으로 판단됩니다. 따라서, 하이퍼파라미터를 조정한다면 Random Forest가 더 나은 성능을 발휘할 가능성이 있을 것을 보입니다.

- **Boosting 계열 모형 비교**

Boosting 계열에서는 AdaBoost(BCR 0.606, 순위 5위)가 GBM(BCR 0.558, 순위 6위) 보다 높은 순위를 보였습니다. 이는 GBM이 Gradient Boosting 과정에서 노이즈까지 학습하여 과적합한 결과로 보입니다.

Bagging 방식이 Boosting 방식보다 평균적으로 더 좋은 성능을 보였습니다. 이는 해당 데이터 셋이 분산이 큰 데이터셋이었을 가능성으로 인해, 분산을 줄이는 Bagging 방식이 더 효과적이었기 때문인 것을 보입니다. 그 중에서도 CART Bagging이 가장 좋은 성능을 발휘했습니다.

하지만 이번 분석에서 진행한 결과들은 최적이라고 하기 어렵습니다. 왜냐하면 컴퓨팅 한계로 하이퍼 파라미터 조정을 적절하게 수행하지 못했기 때문입니다. 저는 실은 RF(Random Forest) Bagging 모델이 가장 우수한 성능을 보일 것으로 예상했습니다. 그 이유는 실무에서도 RF 모델이 우수한 성능으로 많이 사용되고 있으며, 다양한 변수를 효과적으로 고려하여 과적합을 방지하는 능력이 있기 때문입니다. 또한, RF는 각 트리가 독립적으로 학습되어 결과를 도출하기 때문에 다양한 데이터셋에서도 일관되게 좋은 성능을 보이는 경향이 있습니다. 이러한 점에서 RF Bagging 모델이 다른 모델보다 뛰어난 성능을 발휘할 것이라고 생각했습니다.

하지만 RF Bagging 모델의 성능이 예상보다 낮게 나온 이유 중 하나는 컴퓨터 메모리 문제로 인해 n_estimator를 충분히 설정하지 못한 점이 컸습니다. 이를 통해 충분한 트리를 생성하지 못했고, 결과적으로 모델의 다양성과 성능을 충분히 발휘하지 못했습니다. 향후 실험에서는 이러한 제약을 극복하기 위해 더 많은 컴퓨팅 자원을 활용할 수 있는 환경에서 실험을 진행해 볼 필요가 있습니다.