다변량데이터분석

과제 4

2021150456 이예지

목차

[다변량데이터분석][과제4]

[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.

속성이 'object'인 변수들은 get_dummies 함수를 통해 1-of-C coding을 진행해주었다. 학습에 필요하지 않은 'building_id' 변수는 제거하여 총 68개의 독립변수와 1개의 종속변수로 이후 작업을 진행할 것이다.

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.

전체 관측치는 260,601개이고, 결측치는 존재하지 않는다.

sample 함수를 통해 10,000개의 instance를 샘플링하고, sklearn의 train_test_split 함수를 이용하여 학습/검증/테스트 데이터셋을 6:2:2의 비율로 분할하였다. 이때 샘플링 과정부터 분할 과정까지 모두 종속변수 내 범주 비율이 동일하게 유지될 수 있도록 하였다. 추가적으로 StandardScaler 함수를 통해 독립변수에 대한 scaling을 진행하였다.

분류 성능을 평가/비교할 때는 3-class classification의 Accuracy와 Balanced Correction Rate(BCR)을 사용할 것이나, 종속변수 내 범주 불균형[1]이 존재하고 범주 내 중요도에 차이가 없으므로 Accuracy보다는 BCR을 중점으로 판단할 것이다.

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

- **Multinomial logistic regression**
- **Classification and Regression Tree (CART)**
- **Artificial Neural Network (ANN)**

1) Multinomial logistic regression (MLR)

[1] 각 범주의 비율은 약 0.096:0.569:0.335로, 심한 불균형을 보이고 있다.

MLR의 경우, multi_class='multinomial', solver='newton-cg'로 hyperparameter를 설정한 후 test 데이터셋에 대한 성능을 측정하였다. 하이퍼파라미터 후보를 따로 설정하지 않았으므로, 검증 데이터셋은 사용하지 않았다.

2) Classification and Regression Tree (CART)

CART의 최적 하이퍼파라미터 조합을 찾기 위해 아래와 같이 하이퍼파라미터 후보를 설정하였다. Full Tree의 경우 depth는 34, leaf node의 개수는 1714이므로 과적합을 방지하기 위해 max_depth는 34보다 더 작은 값들을, min_samples_laef는 1보다 큰 값들을 후보로 사용하였다.

[Table 1] (CART) 사용된 하이퍼파라미터 종류와 범위

종류	범위	설명
criterion	**["gini", "entropy"]**	**분기 시 불순도를 측정하는 함수. "gini"는 Gini impurity를, "entropy"는 Information gain을 나타낸다.**
max_depth	**[10, 15, 20, None]**	**Tree의 최대 깊이. None으로 설정할 경우 모든 leaf node가 pure하거나 모든 leaf node가 min_samples_split sample보다 적을 때까지 팽창한다.**
min_samples_leaf	**[5, 10, 20, 30, 50]**	**leaf node에 요구되는 최소 샘플 수.**

각 하이퍼파라미터의 범위 선정 이유는 다음과 같다.

1) criterion
   DecisionTreeClassifier에서 사용 가능한 criterion 중 수업 시간에 배운 "gini"와 "entropy"를 후보로 선정하였다.
2) max_depth
   Full Tree의 depth가 34이므로, best model은 Full Tree보다 depth가 작아야 하기 때문에 위와 같이 범위를 설정해주었다. max_depth를 설정하지 않았을 때 최적의 모델이 만들어질 가능성 역시 존재하기 때문에 범위에 'None'도 추가하였다.
3) min_samples_leaf
   min_samples_leaf가 너무 작으면 모델이 과적합될 가능성이 높다. 현재 데이터 개수는 6000개이므로, 위와 같이 범위를 설정하는 것이 적당하다고 판단했다.

학습 데이터로 각 경우를 학습하고, 검증 데이터를 통해 BCR이 가장 높은 경우를 찾아보았다. 그 결과 얻어진 최적의 하이퍼파라미터 조합은 다음과 같다.

[Table 2] (CART) 최적의 하이퍼파라미터 조합

종류	최적 값
criterion	**entropy**
max_depth	**15**

min_samples_leaf	10

depth는 15, leaf node의 개수는 382로 Full Tree에 비해 어느정도 과적합이 해소되었음을 알 수 있다.

3) Artificial Neural Network (ANN)

ANN의 최적 하이퍼파라미터 조합을 찾기 위해 아래와 같이 하이퍼파라미터 후보를 설정하였다.

[Table 3] (ANN) 사용된 하이퍼파라미터 종류와 범위

종류	범위	설명
hidden_layer_sizes	[(10,), (30,), (50,), (100,), (10,10), (30,30), (50,50), (100,100)]	hidden layer 내에 있는 hidden neuron의 개수. i번째 요소는 i번째 hidden layer를 나타낸다.
activation	['logistic', 'tanh', 'relu']	hidden layer에 적용되는 activation function.
learning_rate_init	[0.001, 0.01, 0.1]	초기 학습률. 가중치 업데이트 시 step-size를 제어한다.

각 하이퍼파라미터의 범위 선정 이유는 다음과 같다.

1) hidden_layer_sizes
   입력변수의 차원과 동일하거나 더 낮게 설정함으로써 과적합을 방지하고자 하였다. 일반적으로 hidden layer는 3개 이상 사용하여도 성능이 높아지지 않으며, 각 hidden layer의 node 수는 동일하게 설정해주므로 범위를 위와 같이 설정하였다.

2) activation
   MLPClassifier에서 사용 가능한 activation function 중 비선형변환이 이루어지지 않는 'identity'를 제외한 전부를 후보로 선정하였다.

3) learning_rate_init
   learning_rate_init이 너무 작으면 학습이 너무 느려 수렴까지 오래 걸린다는 단점이 존재하고, learning_rate_init이 너무 크면 gradient가 발산할 수 있다는 단점이 존재한다. 따라서 일반적으로 많이 사용하는 0.001부터 조금 더 큰 값까지 후보로 선정하였다.

추가적으로 max_iter=500 & early_stopping=True 설정을 통해 초기 학습률이 낮아 수렴하지 못하거나, 과적합되는 경우를 방지하고자 하였다.

학습 데이터로 각 경우를 학습하고, 검증 데이터를 통해 BCR이 가장 높은 경우를 찾아보았다. 그 결과 얻어진 최적의 하이퍼파라미터 조합은 다음과 같다.

[Table 4] (ANN) 최적의 하이퍼파라미터 조합

종류	최적 값
hidden_layer_sizes	(100,)
activation	logistic

learning_rate_init	0.01

4) 모델 간 비교

각 모델의 Confusion Matrix는 다음과 같다.

[Table 5] MLR 의 Confusion Matrix

Confusion Matrix		Predicted		
		1	2	3
Actual	1	53	131	9
	2	38	968	132
	3	5	517	147

[Table 6] CART 의 Confusion Matrix

Confusion Matrix		Predicted		
		1	2	3
Actual	1	76	105	12
	2	96	799	243
	3	12	311	346

[Table 7] ANN 의 Confusion Matrix

Confusion Matrix		Predicted		
		1	2	3
Actual	1	78	92	23
	2	74	744	320
	3	12	300	357

전반적으로 1번 범주나 3번 범주를 2번 범주로 잘못 예측하는 경우가 많은 것으로 나타났다. 이는 2번 범주의 비율이 가장 높기 때문으로 예상된다. MLR에 비해 CART와 ANN은 1번 범주와 3번 범주를 맞힌 경우가 늘어났고, 동시에 2번 범주로 잘못 예측한 경우가 줄어들었다. 그러나 2번 범주를 정확히 예측한 경우도 줄어들었다.

테스트 데이터셋에 대한 세 모델의 성능 평가 결과는 다음과 같다.

[Table 8] MLR & CART & ANN 성능 평가 결과

Model	Accuracy	BCR
MLR	0.584	0.372
CART	0.610	0.523
ANN	0.590	0.520

MLR에 비해 나머지 두 모델의 성능이 높은 것을 확인할 수 있다. BCR은 수가 적은 범주를 틀릴 경우 더 많은 패널티를 얻게 되므로, 2번 범주를 많이 맞혔어도 1번 범주와 3번 범주를 많이 틀린 MLR이 가장 낮은 BCR 값을 가지게 된 것으로 보인다.

MLR은 설명변수와 종속변수의 관계를 선형으로 가정하므로, MLR의 낮은 성능은 현재 데이터셋이

선형관계를 가정할 수 없다는 것을 의미한다고 볼 수 있다. 따라서 데이터셋 내 비선형관계를 식별할 수 있는 CART나 ANN이 더 높은 성능을 보인 것으로 예상된다.

CART가 Accuracy와 BCR 모두 가장 높은 것을 확인할 수 있다. 다만 이는 샘플링된 데이터셋에서 기인한 것일 수 있으므로 반복 실험을 해볼 필요가 있다.

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?

[Q1]에서 구한 최적 하이퍼파라미터 조합을 적용한 CART를 base learner로 사용하여 진행하였다. 최적의 bootstrap 수를 구하기 위해 각 경우의 검증 데이터셋에 대한 성능을 구하였다. 데이터셋에 불균형이 존재하므로 최적의 bootstrap 수를 결정할 때에는 Accuracy보다 BCR의 값을 고려하였다.

[Table 9] (CART Bagging) 검증 데이터셋 성능 평가 결과

# of bootstrap	Accuracy	BCR
10	0.660	0.505
30	0.674	0.535
50	0.678	0.529
100	0.682	0.534
200	0.676	0.523
300	0.682	0.529

위 표를 통해 bootstrap이 10일 때를 제외하면 성능이 크게 차이나지 않는 것을 알 수 있다. Accuracy 순위와 BCR 순위가 동일하지는 않지만, bootstrap 수에 따라 성능 평가 결과가 크게 바뀌지 않는다는 점은 동일하다. 또한, bootstrap 수를 늘린다고 하더라도 무조건 성능이 높아지지는 않는 것을 볼 수 있다. 이를 통해 bootstrap 수를 높게 설정하지 않더라도, 이미 모델이 안정된 상태에 도달했으며, 오히려 bootstrap을 너무 높게 설정하면 성능이 낮아질 수 있다는 것을 알 수 있다.

위 표를 통해 결정된 최적의 bootstrap 수는 30이다. 이를 기반으로 테스트 데이터셋에 대한 성능 지표를 계산해보았다.

[Table 10] CART Bagging 의 Confusion Matrix

Confusion Matrix	Predicted		
	1	2	3

Actual	1	68	125	0
	2	43	952	143
	3	4	330	335

[Table 6]과 비교했을 때, 1번 범주와 3번 범주를 10개 내외로 더 틀리지만 2번 범주를 약 150개 정도 더 맞히는 것을 확인할 수 있다. 따라서 CART Bagging의 경우 단일 CART보다 Accuracy가 높을 것으로 예상된다. 다만, 범주 간 비율에 차이가 존재하므로 BCR은 차이가 크지 않을 것으로 보인다.

[Table 11] 단일 CART & CART Bagging 성능 평가 결과

Model	Accuracy	BCR
단일 CART	0.610	0.523
CART Bagging	0.678	0.528

Confusion Matrix를 통해 예상했듯이, CART Bagging의 Accuracy가 더 높은 것을 확인할 수 있다. BCR 역시 조금 더 높은 것을 알 수 있다.

CART Bagging은 각기 다른 학습 데이터셋을 통해 학습한 여러 모델들의 예측 결과를 종합하므로 단일 CART 모델보다 높은 성능을 지닐 수 있었던 것으로 보인다. 하지만 성능 차이가 크지 않았던 것은 sub-model들 간의 분산이 높지 않았기 때문으로 보인다. 각 모델의 편향 역시 비슷했을 것으로 예상된다.

다만, CART Bagging은 단일 CART에 비해 과적합 위험이 낮고, voting과 stacking 과정을 거쳐 더 안정적이므로 CART Bagging이 더 선호된다.

[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.

1) 최적의 Bootstrap 수는 몇으로 확인되는가?

동일하게 [Q1]에서 구한 최적 하이퍼파라미터 조합을 적용하여 진행하였다. 최적의 bootstrap 수를 구하기 위해 각 경우의 검증 데이터셋에 대한 성능을 구하였다. 데이터셋에 불균형이 존재하므로 최적의 bootstrap 수를 결정할 때에는 Accuracy보다 BCR의 값을 고려하였다.

[Table 12] (Random Forest) 검증 데이터셋 성능 평가 결과

# of bootstrap	Accuracy	BCR
10	0.618	0.386
30	0.632	0.377
50	0.639	0.379
100	0.638	0.377

200	**0.635**	**0.378**
300	**0.632**	**0.374**

위 표를 통해 결정된 최적의 bootstrap 수는 10이다. 또한, bootstrap 수의 증가가 성능 향상을 보장할 수 없다는 것을 확인할 수 있다.

bootstrap 수가 10일 때보다 이외의 경우들이 Accuracy는 높지만 BCR이 낮은 것으로 보아, bootstrap 수가 커지면서 비율이 낮은 범주를 더 많이 틀리고, 비율이 높은 범주를 더 많이 맞혔을 것이라 예상할 수 있다. 이전 CART Bagging과 동일하게 bootstrap 수에 따른 성능 변화가 크지 않아 이미 모델이 안정적인 상태에 도달했다고 생각할 수 있다.

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

각각의 최적 Tree 수를 기준으로 성능 지표를 계산해본 결과는 다음과 같다.

[Table 13] CART Bagging & Random Forest 성능 평가 결과

Model	Accuracy	BCR
CART Bagging	**0.678**	**0.528**
Random Forest	**0.626**	**0.399**

CART Bagging이 Random Forest보다 Accuracy, BCR 모두 더 높은 것을 확인할 수 있다. 아래는 Random Forest의 Confusion Matrix이다.

[Table 14] Random Forest 의 Confusion Matrix

Confusion Matrix		Predicted		
		1	2	3
Actual	1	47	142	4
	2	35	1008	95
	3	4	468	197

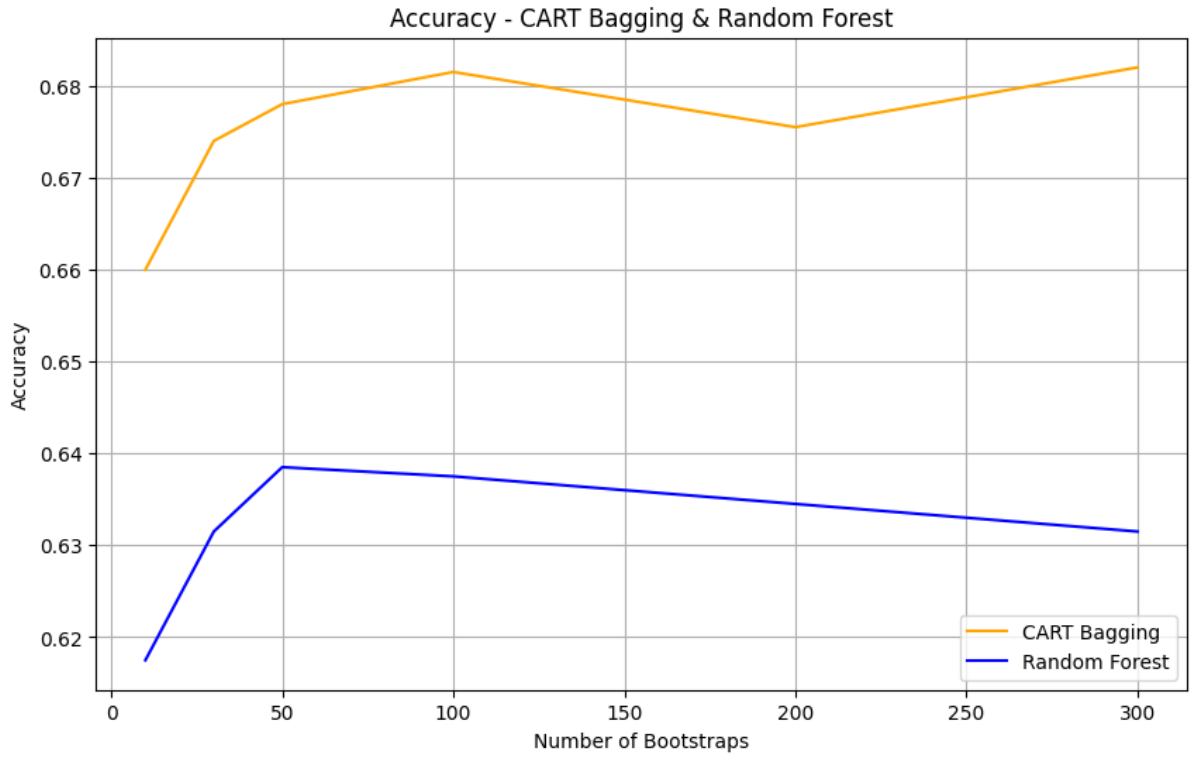
[Table 10]과 비교해보면, 1번 범주는 약 20개, 3번 범주는 약 140개를 더 틀린 것을 확인할 수 있다. 2번 범주는 약 50개를 더 맞힌 것 역시 확인할 수 있다. Random Forest가 비율이 작은 범주를 더 많이 틀렸기에 BCR이 낮아졌고, 범주 무관하게 정답을 맞힌 경우 역시 더 적어 Accuracy도 더 낮은 것을 알 수 있다.

강의 자료에 따르면, Random Forest는 다양성 확보를 통해 시너지 효과를 낸다고 했으나 현재 성능 평가 결과를 보면 단일 CART 모델이 Random Forest보다 좋은 성능을 보이고 있다. Random Forest는 CART Bagging과 다르게, 분기점 탐색 시 원래 변수의 수보다 적은 수의 변수를 임의로 선택하여 해당 변수들만을 고려 대상으로 삼는다. 따라서 예측에 있어 중요하지 않은 변수가 sub-model에 자주 포함되어 정확도가 낮아졌을 수 있다.

반복실험을 하지 않았기 때문에 우연히 bootstrap 수가 적은 경우가 최적으로 선택된 것일 수 있으며, bootstrap 수가 적어 다양성이 충분히 확보되지 않았을 가능성도 존재한다. 혹은 하이퍼파라미터 조합이 최적이 아니었을 가능성 역시 존재한다.

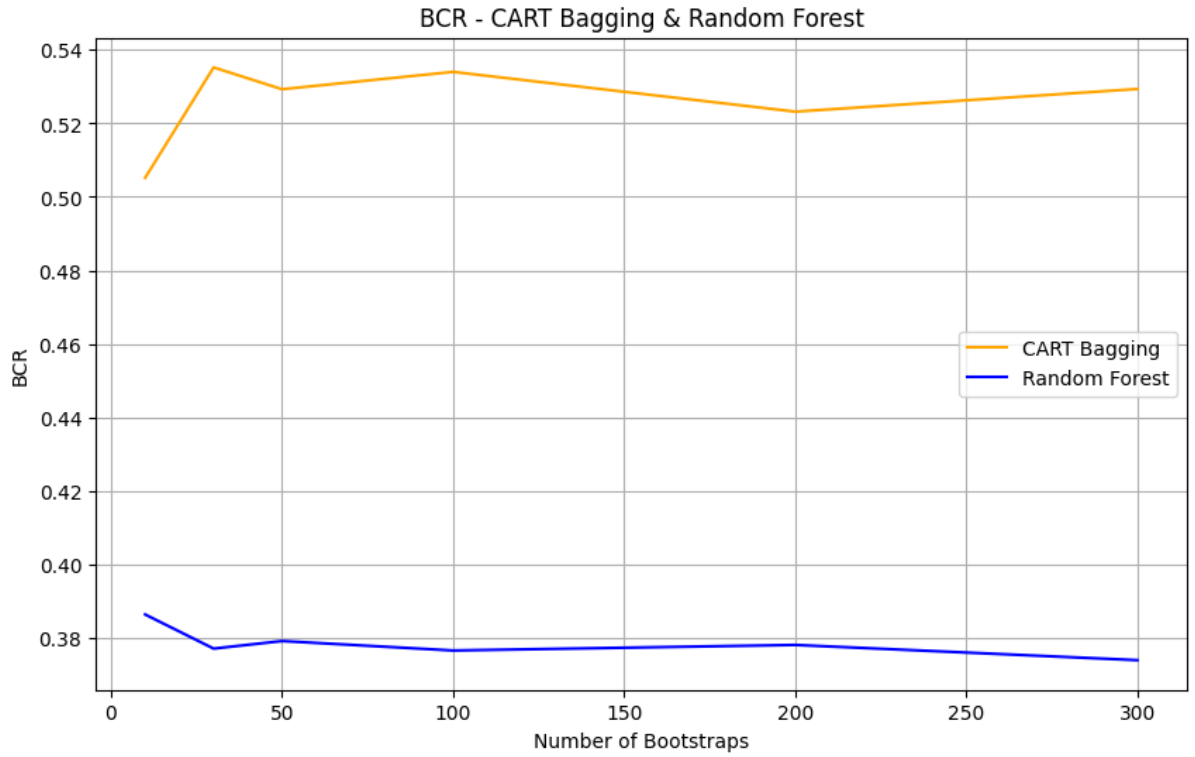
3) 각 Tree의 수(Bootstrap의 수)마다 CART Bagging 모형과의 분류 정확도를 비교할 수 있는 그래프를 도시하시오. Tree의 수는 CART Bagging과 Random Forest의 성능 차이에 영향을 미친다고 볼 수 있는가?

다음은 CART Bagging과 Random Forest의 Accuracy, BCR을 비교한 그래프들이다.



[Figure 1] Accuracy of CART Bagging & Random Forest

CART Bagging의 경우 bootstrap 수가 100이 될 때까지는 증가하는 경향을 보이다 감소, 200 이후 다시 증가하는 모습을 보이고 있다. 반면에 Random Forest는 bootstrap 수가 50이 될 때까지 증가하다가 이후에는 계속 감소하는 것을 알 수 있다. 둘 다 bootstrap이 10인 경우를 제외하면, 0.01 이내의 차이 정도만이 존재하여 Accuracy의 변동성이 크지 않은 것을 볼 수 있다. 또한, bootstrap 수에 따라 성능이 달라지고, bootstrap 수가 큰 것이 높은 성능을 보장하지 않는다는 것을 확인할 수 있다. 그러나 항상 CART Bagging이 Random Forest보다 좋은 성능을 보이고 있음을 알 수 있다.



[Figure 2] BCR of CART Bagging & Random Forest

변동은 Random Forest가 CART Bagging보다 작으나, BCR 역시 CART Bagging에 비해 Random Forest가 낮은 것을 확인할 수 있다. 두 모델의 BCR 차이가 Accuracy 차이보다 큰 것 또한 확인할 수 있다.

Accuracy와 BCR을 통해 bootstrap 수가 개별 모델 내 성능 차이를 만들어내는 것은 맞지만, 본 데이터셋에서는 bootstrap 수에 관계없이 CART Bagging이 더 우수한 모델임을 알 수 있다. 이는 [Q3-2]에서 언급했다시피, 분기점 선택 시 임의의 후보 변수 집합을 사용하는 Random Forest의 특징과 관련이 있을 것으로 예상된다.

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

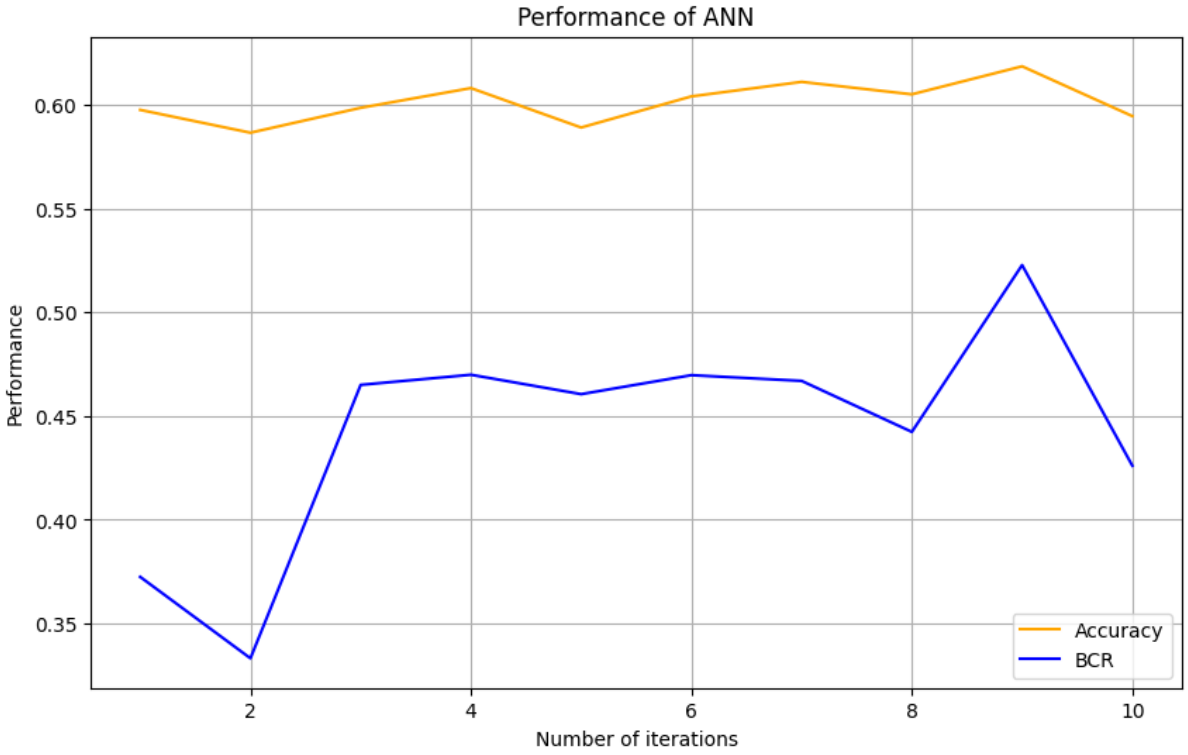
반복 실험을 위해 이전에 분할해둔 학습/검증/테스트 데이터셋 대신 매 실험마다 새롭게 학습/검증/테스트 데이터셋을 분할하여 사용하였다. 이때, 전체 데이터는 사전에 샘플링했던 10,000개의 instance만을 사용하였으며, StandardScaler가 적용된 상태로 진행하였다. [Q1]과 동일하게 6,000개의 학습 데이터로만 학습을 진행하였으며, 본 실험에서 검증 데이터셋은 분할만 되어있을 뿐 실질적으로 사용되지는 않았다. 아래는 단일 ANN의 성능 평가 결과이며, 평균(표준편차)의 형태로 작성되어있다.

[Table 15] (10 회 반복) 단일 ANN 성능 평가 결과

Model	Accuracy	BCR
(10회 반복) 단일 ANN	0.601(0.010)	0.443(0.054)

Accuracy에 비해 BCR의 평균이 낮고, 표준편차가 큰 것을 확인할 수 있다. 이는 대부분의 객체를 비율이 가장 높은 2번 범주로 예측하였기 때문일 것이다.

앞서 [Q1]에서보다 Accuracy는 조금 더 높아지고, BCR은 낮아진 것을 확인할 수 있다. 따라서 보다 정확히 원인을 파악하고자 매 반복시의 성능 평가 결과를 그래프로 나타내었다.



[Figure 3] Performance of 10 iterated ANN - Accuracy & BCR

BCR의 경우, Accuracy에 비해 변동이 매우 심한 것을 확인할 수 있다. 이는 데이터셋에 따른 성능 차이가 존재하기 때문으로 볼 수 있다. iteration이 9인 경우, [Q1]에서와 비슷한, 가장 높은 성능을 보였다. 따라서 [Q1]에서의 모델이 우연히 데이터와 초기값의 조합이 좋았던 것으로 보인다. 따라서 더 많은 반복시행을 통해 보다 정확한 평균 및 표준편차를 구할 필요가 있어 보인다.

Accuracy는 대개 비슷한 정도를 유지하는 반면, BCR은 크게 변화하는 구간들이 존재한다. 이러한 경우는 전체 정답 개수는 비슷하나, 비율이 적은 1번 범주와 3번 범주를 얼마나 맞추었느냐에 따라 달라졌을 것이다.

또한, ANN의 early stopping을 True로 설정해두었기 때문에, local optimum에 빠진 상태로 학습이 종료되는 경우가 있었을 것으로 보인다. 추가적으로 early stopping을 False로 설정하고 학습을 진행했을 때는 성능이 어떻게 변화하는지 알아볼 필요가 있어 보인다.

[Q5] ANN Bagging 모델에 대해 다음 물음에 답하시오.

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

[다변량데이터분석][과제4]

[Q1]에서 구한 최적 하이퍼파라미터 조합을 적용하여 진행하였다. 최적의 bootstrap 수를 구하기 위해 각 경우의 검증 데이터셋에 대한 성능을 구하였다. 데이터셋에 불균형이 존재하므로 최적의 bootstrap 수를 결정할 때에는 Accuracy보다 BCR의 값을 고려하였다.

bootstrap별 검증 데이터셋에 대한 성능 평가 결과는 다음과 같다.

[Table 16] (ANN Bagging) 검증 데이터셋 성능 평가 결과

# of bootstrap	Accuracy	BCR
10	0.607	0.468
30	0.618	0.473
50	0.618	0.476
100	0.618	0.471
200	0.615	0.474
300	0.620	0.477

앞서 보았던 CART Bagging과 동일하게 bootstrap 수가 증가한다고 해서 Accuracy와 BCR이 무조건 증가하지는 않는다는 것을 확인할 수 있다.

2) 최적의 Bootstrap 수는 몇으로 확인되는가?

[Table 16]에 따르면, bootstrap 수가 300일 때 BCR이 가장 크므로 최적의 bootstrap 수는 300으로 확인된다. bootstrap 수와 성능이 무조건 비례하는 것은 아니나, bootstrap 수가 300일 경우 다양성이 높아 좋은 성능을 보였을 것으로 추정된다.

3) 이 모델은 최적의 단일 모형과 비교했을 때 어떠한 성능의 차이가 있는가?

다음은 단일 ANN과 ANN Bagging 모델의 테스트 데이터셋에 대한 성능 평가 결과를 나타낸 표이다.

[Table 17] 단일 ANN & ANN Bagging 성능 평가 결과

Model	Accuracy	BCR
단일 ANN	0.590	0.520
ANN Bagging	0.598	0.453

Bagging은 분산이 높고 편향이 작은 ANN에 적절한 방법이며, voting과 stacking을 통해 모델의 성능을 안정적으로 만들어줄 것으로 기대되어 ANN Bagging이 단일 ANN보다 높은 성능을 낼 것이라 예상했었다. 그러나 단일 ANN에 비해 ANN Bagging이 Accuracy는 조금 더 높지만, BCR이 낮은 것을 확인할 수 있다. 이는 [Q4]에서 확인한 것처럼, 단일 ANN이 우연히 좋은 조건 하에서 높은 성능을 보여주었기 때문으로 생각된다. 혹은 early stopping condition을 True로 설정해둔 것으로 인해 local optimum에 빠지는 경우가 많아 ANN Bagging의 성능이 낮아졌을 가능성도 있어 보인다.

우연히 얻어진 단일 ANN의 결과 대신, [Q4]의 결과를 생각해본다면, ANN Bagging이 BCR 측면에서 조금 더 나은 성능을 보여준 것을 확인할 수 있다. ANN Bagging은 반복실험을 하지 않았지만, 300개의 모델을 통해 얻어진 결과이므로 다른 데이터셋을 사용하더라도 그 변동성이 크지 않을 것이라 예측할 수 있다. 이를 통해 Bagging을 적용하는 것이 단일 모델을 사용하는 것보다 안정적인 성능을 보장할 수 있다는 것을 알 수 있다.

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

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

AdaBoost의 최적 하이퍼파라미터 조합을 찾기 위해 아래와 같이 하이퍼파라미터 후보를 설정하였다. base learner는 [Q1]에서 찾은 최적의 하이퍼파라미터 조합을 지닌 decision tree로 설정하였다.

[Table 18] (AdaBoost) 사용된 하이퍼파라미터 종류와 범위

종류	범위	설명
n_estimators	[10, 30, 50, 100, 200, 300]	boosting이 종료될 때까지 생성할 수 있는 최대 estimator의 수.
learning_rate	[0.01, 0.1, 1, 10, 100]	boosting 알고리즘의 각 반복 과정에서 각각의 분류기에 적용되는 가중치. 높은 learning rate는 각 분류기의 기여도를 증가시킨다. n_estimators와 trade-off 관계가 존재한다.

각 하이퍼파라미터의 범위 선정 이유는 다음과 같다.

1) n_estimators
   이전 bagging 모델들과 동일하게 다양한 범위의 최대 estimator의 수를 실험해볼 수 있도록 설정하였다.

2) learning_rate
   n_estimators의 후보가 작게는 10부터 크게는 300까지 존재하므로, n_estimators와 trade-off 관계를 가진 learning rate 역시 매우 작은 경우와 큰 경우 모두 다룰 필요가 있다고 생각하였다.

학습 데이터로 각 경우를 학습하고, 검증 데이터를 통해 BCR이 가장 높은 경우를 찾아보았다. 그 결과 얻어진 최적의 하이퍼파라미터 조합은 다음과 같다.

[Table 19] (AdaBoost) 최적의 하이퍼파라미터 조합

종류	최적 값
n_estimators	10
learning_rate	100

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

다음은 AdaBoost의 Confusion Matrix와 각 모델들의 성능 평가 결과를 나타낸 표이다.

[Table 20] AdaBoost 의 Confusion Matrix

Confusion Matrix		Predicted		
		1	2	3
Actual	1	72	116	5
	2	82	838	218
	3	9	316	344

[Table 21] 모델별 성능 평가 결과

Model	Accuracy	Accuracy 순위	BCR	BCR 순위
MLR	0.584	7	0.372	7
CART	0.610	4	0.523	2
ANN	0.590	6	0.520	4
CART Bagging	0.678	1	0.528	1
Random Forest	0.626	3	0.399	6
ANN Bagging	0.598	5	0.453	5
AdaBoost	0.627	2	0.521	3

AdaBoost는 Accuracy와 BCR 모두 나쁘지 않았으나, BCR의 경우 1등~4등까지 차이가 크지 않아 데이터셋이 바뀐다면 순위 역시 바뀔 수 있을 것으로 보인다.

먼저 구축된 모델들 중 가장 높은 BCR 성능을 보인 CART Bagging 모델의 Confusion Matrix와 AdaBoost의 Confusion Matrix를 비교해보면, AdaBoost가 1번 범주와 3번 범주를 더 잘 맞히고, 2번 범주는 더 많이 틀린 것을 알 수 있다. 이는 Accuracy에서도 확인할 수 있는데, 절대적인 숫자로 따지면 새롭게 맞히게 된 객체 수보다 틀리게 된 객체 수가 많기 때문이다.

AdaBoost 역시 base learner로 decision tree를 사용했는데, 이를 통해 전반적으로 이 데이터셋에서는 Random Forest를 제외하고, decision tree를 사용한 모델들이 그나마 좋은 성능을 보여주고 있음을 알 수 있다. 이는 base learner로 사용되었던 decision tree의 하이퍼파라미터 조합이 최적이었기 때문으로 추정되며, ANN Bagging이 다른 모델들보다 성능이 조금 떨어지는 것 또한 하이퍼파라미터 조합 때문인 것으로 추정된다.

AdaBoost는 단일 CART와 비교했을 때 Accuracy는 조금 더 높지만, BCR이 약간 낮은 것을 확인할 수 있다. 이는 AdaBoost가 상대적으로 비율이 높은 2번 범주를 더 많이 맞추었기 때문으로 볼 수 있다.

AdaBoost는 틀린 데이터에 대해 더 높은 가중치를 설정하여 학습을 진행한다. 그러나 현 데이터셋에서는 1번 범주에 속하는 객체는 절대적인 숫자가 많지 않으므로, 1번 범주에 속하는 객체의 선택 확률이 증가하더라도 다른 범주에 속하는 객체를 틀린 경우가 더 많을 수밖에 없다. 따라서 처음부터 적은 비율의 범주에 대한 가중치를 높게 설정한 후 Boosting을 진행하면 더 높은 성능

을 보일 수 있을 것이라 생각한다.

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

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

GBM의 최적 하이퍼파라미터 조합을 찾기 위해 아래와 같이 하이퍼파라미터 후보를 설정하였다.

[Table 22] (GBM) 사용된 하이퍼파라미터 종류와 범위

종류	범위	설명
n_estimators	[50, 100, 150]	수행할 boosting 단계 수. GBM은 과적합에 robust해, 대개 큰 수를 사용하면 더 좋은 결과를 보인다.
learning_rate	[0.01, 0.1, 1]	각 트리의 기여도를 learning_rate만큼 줄인다. learning_rate와 n_estimators 사이에 trade-off 관계가 존재한다.
subsample	[0.7, 1.0]	각 base learner를 학습시키는데 사용할 sample의 비율. 1.0보다 작으면 Stochastic Gradient Boosting이 된다. 1.0보다 작게 설정하면 분산은 줄어드나 편향이 높아진다.
max_depth	[10, 15, 20, None]	각 estimator의 최대 깊이.
min_samples_leaf	[5, 10, 20]	leaf node에 필요한 최소 샘플 수.

각 하이퍼파라미터의 범위 선정 이유는 다음과 같다.

1) n_estimators
   Bagging의 경우에는 estimator의 수와 성능이 비례하지 않는 것을 알 수 있었다. boosting 역시 그럴 가능성이 있다고 생각하여 default 값인 100을 기준으로 더 작은 값인 50과 더 큰 값인 150을 후보로 선정하였다.

2) learning_rate
   learning_rate는 n_estimators와 trade-off 관계가 있으므로, 역시 동일하게 default 값인 0.1을 기준으로 더 작은 값인 0.01과 더 큰 값인 1을 후보로 선정하였다.

3) subsample
   각 base learner를 학습시키는데 사용하는 sample의 비율이 줄어들면 각 estimator의 다양성이 높아질 가능성이 존재하므로 default 값인 1.0과 0.7을 후보로 설정하였다.

4) max_depth
   [Q1]에서 decision tree의 최적 하이퍼파라미터 조합을 찾을 때와 동일하게 설정하였다. default 값이 3으로 설정되어 있어, underfitting의 위험이 있다고 생각하였고, full tree의 경우보다는 작은 값을 가져야할 필요가 있다고 생각하였다.

5) min_samples_leaf

[Q1]과 동일하게 설정하려 했으나, min_samples_leaf를 30, 50으로 설정할 경우 sample 수 부족으로 학습이 잘 되지 않을 수 있다고 생각하여 위와 같은 범위로 설정하였다.

학습 데이터로 각 경우를 학습하고, 검증 데이터를 통해 BCR이 가장 높은 경우를 찾아보았다. 그 결과 얻어진 최적의 하이퍼파라미터 조합은 다음과 같다.

[Table 23] (GBM) 최적의 하이퍼파라미터 조합

종류	최적 값
n_estimators	50
learning_rate	0.1
subsample	1.0
max_depth	None
min_samples_leaf	20

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

다음은 GBM의 Confusion Matrix와 각 모델들의 성능 평가 결과를 나타낸 표이다.

[Table 24] GBM 의 Confusion Matrix

Confusion Matrix		Predicted		
		1	2	3
Actual	1	75	115	3
	2	53	888	197
	3	3	299	367

[Table 25] 모델별 성능 평가 결과

Model	Accuracy	Accuracy 순위	BCR	BCR 순위
MLR	0.584	8	0.372	8
CART	0.610	5	0.523	3
ANN	0.590	7	0.520	5
CART Bagging	0.678	1	0.528	2
Random Forest	0.626	4	0.399	7
ANN Bagging	0.598	6	0.453	6
AdaBoost	0.627	3	0.521	4
GBM	0.665	2	0.55	1

GBM의 BCR 값이 가장 높았으며, Accuracy는 CART Bagging이 가장 높은 것을 확인할 수 있다.

마찬가지로 먼저 구축된 모델들 중 가장 높은 BCR 성능을 보인 CART Bagging 모델의 Confusion Matrix와 GBM의 Confusion Matrix를 비교해보면, GBM이 1번 범주와 3번 범주를 더 잘 맞히고, 2번 범주는 더 많이 틀린 것을 알 수 있다. 그러나 Accuracy에서도 확인할 수 있다시피, 절대적인 숫자로 따지면 새롭게 맞히게 된 객체 수보다 틀리게 된 객체 수가 많아 GBM보다 CART Bagging 모델의 Accuracy가 더 높다.

[다변량데이터분석][과제4]

[Q6]에서 전반적으로 이 데이터셋에서는 Random Forest를 제외하고, decision tree를 사용한 모델들이 그나마 좋은 성능을 보여주고 있었다는 것을 알았기 때문에, GBM 역시 CART 모델을 학습할 때 조절했던 하이퍼파라미터들을 변화시켜보는 과정을 거쳤다.

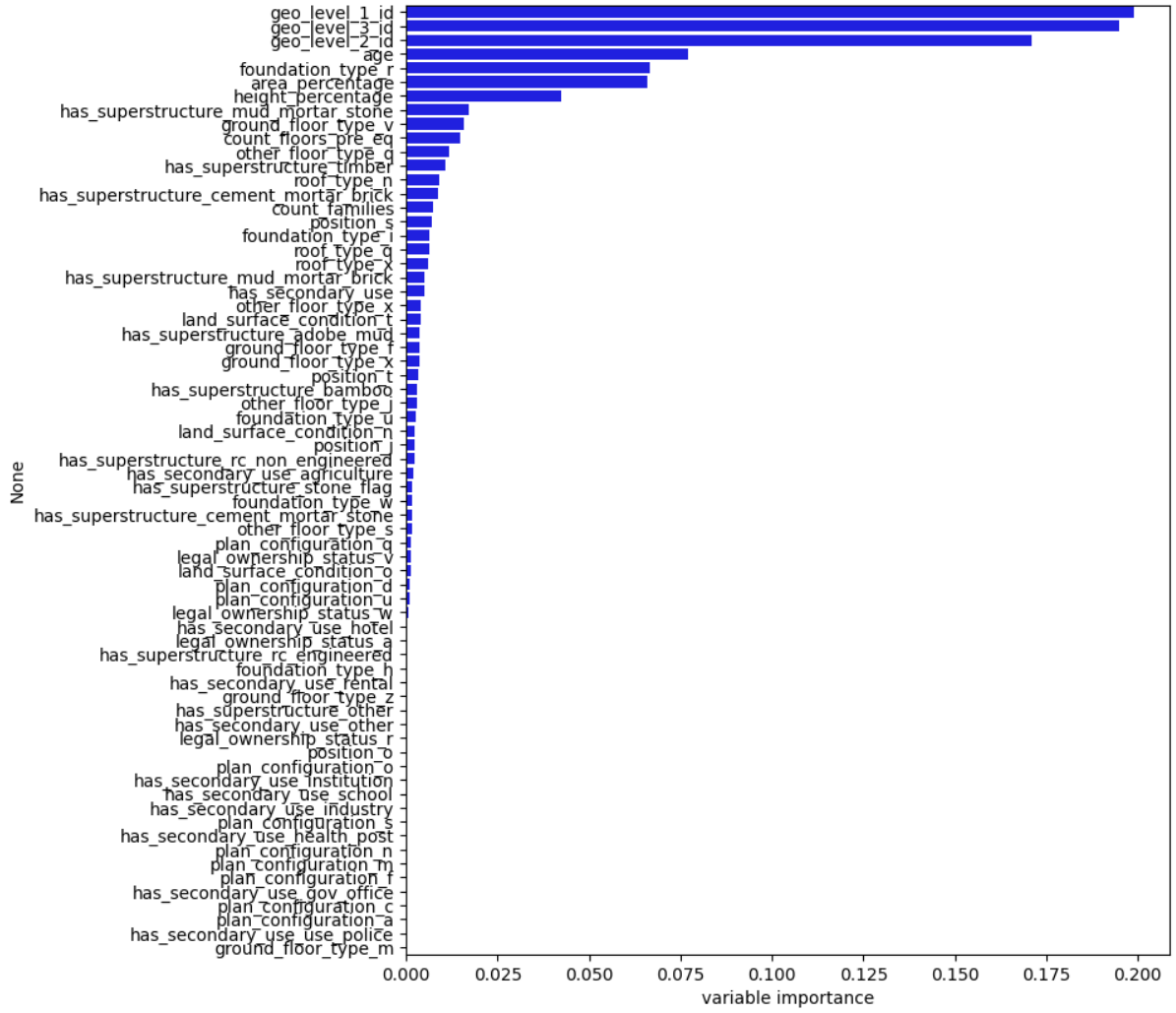
GBM은 noise 이외의 부분 중 이전 모델이 학습하지 못한 부분을 추가적으로 학습하는 알고리즘이므로, 틀린 데이터의 선택 확률을 높여주는 AdaBoost보다 좀 더 높은 성능을 보여줄 수 있었을 것으로 보인다.

다른 Bagging 모델들과 비교하여도, 각 base learner들을 학습시켜 voting을 통해 예측하는 것보다 학습하지 못한 부분을 추가로 학습하는 것이 본 데이터셋에서는 더 적합한 방법이었던 것으로 보인다. 이는 데이터셋이 매우 복잡해, 비슷한 모델 여러 개를 가지고 하는 예측은 한계가 있었기 때문으로 추측된다.

추가적으로, GBM이 과적합에 robust하기 때문에, 다른 모델들과는 달리 일반화 성능이 더 높았을 가능성도 존재한다.

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

다음은 GBM 모델에서 산출된 변수의 중요도를 나타낸 그래프이다.



[Figure 4] Variable Importance of GBM

geo_level_1_id, geo_level_3_id, geo_level_2_id 순으로 높은 중요도를 보이고 있음을 알 수 있다. age 중요도는 geo_level_2_id의 중요도보다 절반 이상 감소한 것을 알 수 있다. 따라서 geo_level_1_id, geo_level_3_id, geo_level_2_id가 예측에 있어 매우 중요한 역할을 했음을 알 수 있다. 해당 세 변수는 split에 많이 사용되었고, split에 사용되었을 때의 효과가 컸을 것으로 보인다.

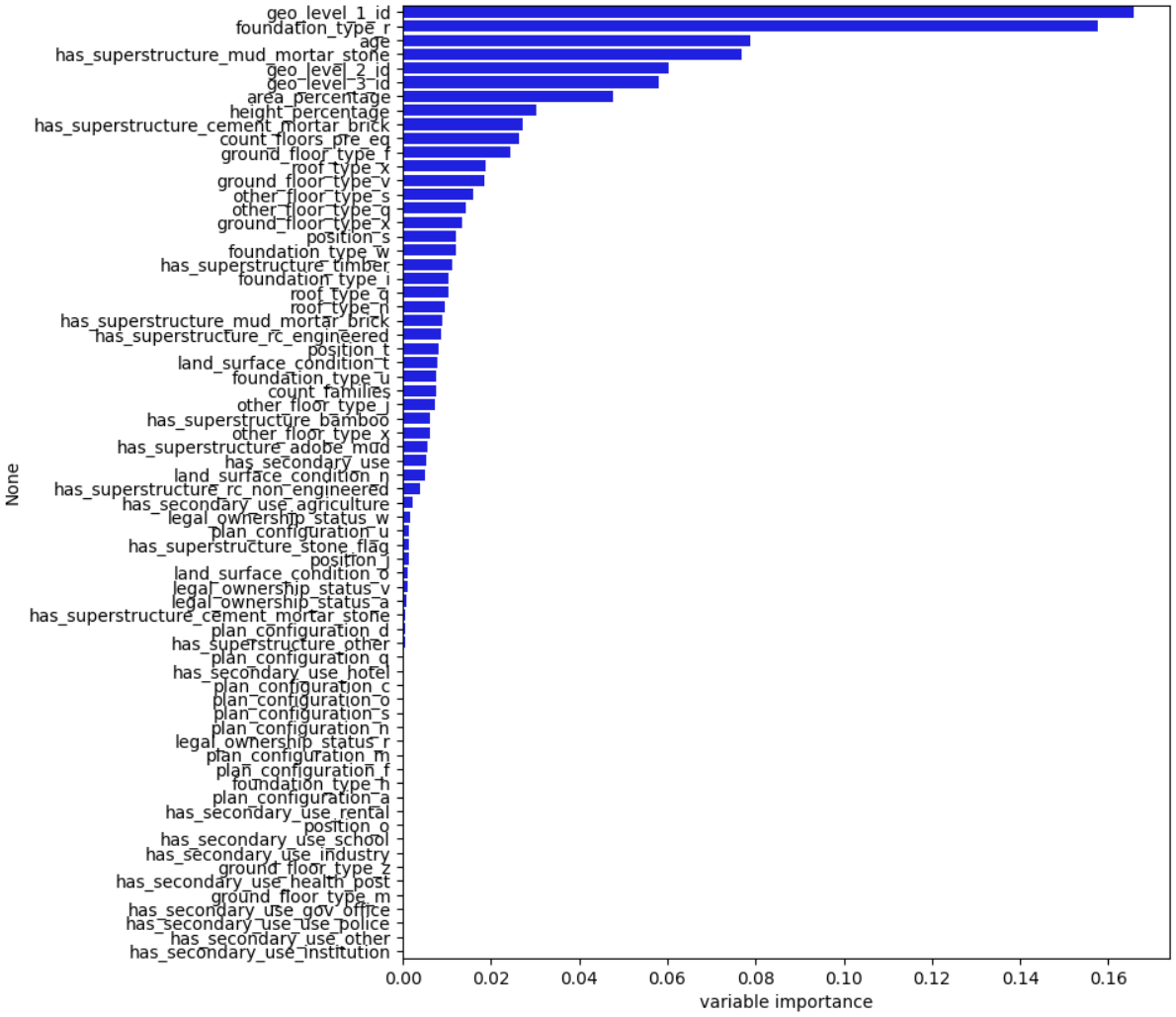
반면에 ground_floor_type_m[2], has_secondary_use_use_police, plan_configuration_a[3] 등은 중요도가 매우 낮은 것을 확인할 수 있다. 이는 변수가 split에 사용된 적이 없거나, split에 사용되었다고 하더라도 그 효과가 크지 않았음을 의미한다.

다음은 Random Forest 모델에서 산출된 변수의 중요도를 나타낸 그래프이다.

---

[2] ground_floor_type이 m일 때의 dummy variable.

[3] plan_configuration이 a일 때의 dummy variable.



[Figure 5] Variable Importance of Random Forest

Random Forest에서는 geo_level_1_id, foundation_type_r[4]의 중요도가 매우 높았으며, age의 중요도가 foundation_type_r의 중요도의 절반 수준에 그치는 것을 확인할 수 있다.

geo_level_1_id는 GBM과 Random Forest 모두에서 가장 중요한 변수로 사용되었으며, 그 중요도에 차이는 있을지라도 geo_level_2_id, geo_level_3_id, foundation_type_r, age 등의 변수는 두 모델 모두에서 어느 정도 중요한 위치에 있었음을 확인할 수 있다. GBM에서 중요하지 않다고 판단된 변수들, 예를 들어 ground_floor_type_m, has_secondary_use_use_police, plan_configuration_a 등도 Random Forest에서 중요하지 않은 경우가 많았다. 따라서 이렇게 중요하지 않다고 판단된 변수들을 제거한 후 모델을 새롭게 학습시키면 성능이 개선될 수 있을 것으로 보인다.

Random Forest가 임의로 몇몇 변수들을 분기 시에 사용하지 않기 때문에 중요한 변수일지라도 Random Forest의 특성에 의해 실제보다 중요도가 낮게 판별되었을 가능성이 존재한다.

---

[4] foundation_type이 r일 때의 dummy variable.

[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?

다음은 총 8가지의 모델의 성능 평가 결과를 종합한 표이다.

[Table 26] 모델별 성능 평가 결과

Model	Accuracy	BCR	BCR 순위
MLR	**0.584**	**0.372**	**8**
CART	**0.610**	**0.523**	**3**
ANN	**0.590**	**0.520**	**5**
CART Bagging	**0.678**	**0.528**	**2**
Random Forest	**0.626**	**0.399**	**7**
ANN Bagging	**0.598**	**0.453**	**6**
AdaBoost	**0.627**	**0.521**	**4**
GBM	**0.665**	**0.55**	**1**

BCR은 GBM, CART Bagging, CART, AdaBoost, ANN, ANN Bagging, Random Forest, MLR 순으로 높은 것으로 나타났다. CART, AdaBoost, ANN은 그 차이가 매우 작아 데이터셋이 변할 경우 순위가 뒤바뀔 수 있을 것으로 보인다.

MLR의 성능이 가장 좋지 않은 것을 통해 데이터셋이 선형으로 분류하기에는 부적절함을 알 수 있다. 따라서 비선형으로 분류할 수 있는 다른 모델들의 성능이 더 높았으며, 복잡한 데이터를 더 잘 처리할 수 있는 앙상블 계열의 모델이 가장 좋은 성능을 보여주었을 것으로 예상된다.

전반적으로 Bagging 계열의 모델보다 Boosting 계열의 모델의 성능이 좋은 것으로 보아, 분산을 줄이는 것보다 편향을 줄이는 것이 모델의 성능 개선에 도움이 될 것으로 보인다.

앞서 언급했다시피, base learner로 decision tree를 사용한 모델들이 비교적 좋은 성능을 보여주었다. 이는 base learner로 사용되었던 decision tree의 하이퍼파라미터 조합이 최적이었기 때문으로 추정되며, ANN Bagging이 다른 모델들보다 성능이 조금 떨어지는 것 또한 하이퍼파라미터 조합 때문인 것으로 추정된다. GBM 역시 decision tree에서 사용되는 하이퍼파라미터를 변경해준 것이 좋은 성능을 보이는데 영향을 미쳤을 것으로 예상된다.

[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 hyperparamter)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%

데이터 전처리 관점에서 불균형을 해소하기 위해 시도할 수 있는 방법으로는 resampling이 있다.

resampling에서도 under sampling과 over sampling 방법이 존재하며, over sampling에는 단순 over sampling, 알고리즘을 이용한 over sampling이 존재한다.

under sampling은 모든 범주의 데이터 수가, 가장 적은 수의 데이터를 가진 1번 범주의 데이터 수만큼으로 맞춰지는 방법이다. 데이터의 수를 줄이는 방법이므로 noise뿐만 아니라 정상 데이터 역시 삭제될 가능성이 존재하며, 데이터 수가 적어져 under fitting의 위험성이 높아진다.

over sampling은 모든 범주의 데이터 수가, 가장 많은 수의 데이터를 가진 2번 범주의 데이터 수만큼으로 맞춰지는 방법이다. 이때, 동일한 데이터를 늘릴 수도 있지만, 알고리즘을 통해 새로운 데이터를 생성할 수도 있다. 또한, 모든 클래스의 데이터 수를 더 늘릴 수도 있다. 단순 over sampling의 경우 over fitting의 위험성이 있다. 그러나 under sampling보다는 over sampling이 선호된다.

resampling 기법을 적용하기 위한 데이터로는 이전 모델들과 동일하게 사전에 샘플링했던 10,000개의 instance만을 사용하였으며, StandardScaler가 적용된 상태로 진행하였다. 먼저 10,000개의 데이터를 가지고 resampling 기법을 적용한 뒤, 6:2:2의 비율로 학습/검증/테스트 데이터셋을 분할하였다.

[Q8]에서 보았다시피, 가장 높은 성능을 보인 GBM의 하이퍼파라미터 조합을 그대로 사용하여 불균형을 해소한 데이터셋을 학습해본 결과는 다음과 같다.

[Table 27] 알고리즘별 성능 평가 결과

Algorithm	Accuracy	BCR
Under Sampling	**0.615**	**0.604**
Over Sampling	**0.819**	**0.807**
SMOTE	**0.776**	**0.773**
ADASYN	**0.781**	**0.779**

resampling을 통해 데이터셋의 불균형을 해소한 결과, 모든 방법이 resampling을 적용하지 않은 데이터셋에 비해 높은 성능을 보여주고 있음을 알 수 있다. 또한, 이전 모델들과는 다르게 Accuracy와 BCR이 비슷한 점수를 보이고 있음을 알 수 있다.

Under sampling을 적용했을 때 유난히 낮은 성능이 나왔는데, 이는 데이터셋의 크기가 가장 작았기 때문으로 보인다. sampling 후 각 범주의 데이터 개수는 다음과 같았다.

[Table 28] Resampling 후 각 범주의 데이터 개수

Algorithm	Class 1	Class 2	Class 3
Under Sampling	**964**	**964**	**964**
Over Sampling	**5689**	**5689**	**5689**
SMOTE	**5689**	**5689**	**5689**
ADASYN	**6048**	**5689**	**5575**

Under sampling의 경우 다른 방법론보다 약 14,000개 정도 데이터가 적었기 때문에 under fitting이 일어났을 것으로 짐작할 수 있다. over sampling의 경우, 알고리즘을 통한 over sampling보다 단순 over sampling이 더 높은 성능을 보여주었는데, 이는 오히려 동일한 데이터를 생성함으로써

[다변량데이터분석][과제4]

1번 범주와 3번 범주를 더 잘 예측할 수 있도록 만들었기 때문으로 보인다. 다만, over sampling을 통해 데이터 수를 늘린 후 학습을 진행하였기 때문에, 총 6,000개의 학습 데이터만을 사용한 이전 모델들보다 over sampling 후의 데이터셋을 사용한 모델이 성능이 좋을 수밖에 없었을 것이라 생각된다. 그러나 under sampling 후의 데이터셋을 사용한 모델은 이전 모델들보다 더 적은 수의 데이터를 가지고 학습하였음에도 불구하고 더 좋은 성능을 나타냈기에, over sampling을 진행한 데이터셋 역시 이전 모델들과 동일한 수의 학습 데이터를 사용하더라도 더 좋은 성능을 보일 것으로 기대할 수 있다.