다변량분석 6번째 과제

Assignment 6

Ensemble

학과 산업경영공학부

학번 2016170863

이름 추창욱

[0] Data preparation

데이터 줄이기

현재 총 26만개가 조금 넘는 데이터가 있다는 것을 확인하였다. 이 데이터를 통으로 사용하게 되면 시간안에 절대 끝낼 수 없을 거라는 판단 하에 데이터의 개수를 줄여서 사용하고자 하였다. 현재 가지고 있는 데이터의 10%만을 이용할 예정이며 현재 데이터 속 총 3개의 범주(class)가 있으므로 오리지널 데이터셋에 있는 범주들의 비율에 맞게 줄여야 큰 차이가 없을 거라 생각하여 다음과 같은 과정을 통해 데이터를 줄여주었다.

<위의 데이터셋으로부터 세번째 시도>

위에서 총 데이터의 10%만을 사용하여 코드를 돌려 보았었다. 하지만 5번 지문을 돌리기 위해 코드를 돌리면 CPU에 과부하가 심하게 걸려 세번의 CPU 터짐이 있었고, 데이터가 너무 무거워서 일어난 일이라 생각하여 Bootstrap수를 줄이거나 ANN의 maxit의 수도 줄여보는 등 여러가지 시도를 해보았다. 결국에 도달한 방법은 분석하는 데이터의 무게를 줄이는 최후의 방법을 쓸 수 없다는 생각을 하게 되어 결국 5%의 데이터만을 사용하여 다시 전체 문제에 대해서 돌려보았다.

원-핫 인코딩 및 데이터 나누기

현재 이 데이터 셋에는 숫자형, character형, numeric이지만 factor의 의미를 가지는 데이터 타입들이 있다. Numeric 형을 가지는 변수는 c(2:8,28)였으며 이는 scaling을 통해 정규화를 시켜 주었다. Character은 c(9:15,27)였으며, class.ind( )를 이용하여 더미 변수를 생성하였고, 이름이 헷갈리지 않도록 해당하는 변수이름을 붙여 주었다. 마지막 데이터 타입을 가지는 변수는 c(16:26,29:39)였으며, 이는 factor로 변환해주었다.

위에서 이뤄진 작업들은 input에 넣어주었고target을 factorize 시킨 후 결합하여 최종적으로 사용할 데이터 Final data를 만들어주었다. 이는 지문에서 요구한 형태의 trn, val, tst로 나누어 주었다.

[Q1] MLR CART ANN 모델을 구축하고 Accuracy 와 BCR 관점에서 분류 정확도를 비교

MLR

<데이터의 10%사용>

Confusion Matrix			
MLR	1	2	3
1	181	357	19
2	113	2985	315
3	21	1634	391

Performace Eval	
ACC	0.5912566
BCR	0.3787048

첫번째로 MLR을 돌려 나온 결과에 대한 분석을 하자면 단순 정확도는 0.6정도에 BCR은 0.38정도의 저조한 결과를 보인다는 것을 확인할 수 있다. 혼동행렬을 보면 알 수 있듯이 2번 범주가 매우 많아서 그런 것인지 2번 범주라고 판단한 케이스가 가장 많았고, 이에 따라서 물론 class 2의 비율이 압도적으로 많기 때문에 단순 정확도는 결과에 비해 높게 나왔으나 BCR을 살펴보면 0.6정도의 정확도 성능에 비해 낮다는 것을 확인할 수 있었다.

<데이터의 5%사용>

Confusion Matrix			
MLR	1	2	3
1	99	201	9
2	67	1570	144
3	8	832	198

Performace Eval	
ACC	0.596867
BCR	0.3776824

단순 정확도는 0.6정도에 BCR은 0.38정도의 저조한 결과를 보인다는 것을 확인할 수 있다. 분석 결과가 전체적으로 10%를 이용하여 돌렸을 때와 매우 유사한 형태로 나왔다는 것을 확인할 수 있었다.

CART

<데이터의 10%사용>

다음으로는 CART 모델을 학습해보았다. 이때 사용한 하이퍼파라미터들은 다음과 같았다. CART를 이용하였기 때문에 강의 자료에 나와있던 코드처럼 ctree()를 이용하여 구축하였다.

Hyper Parameters	
min_criterion	0.8, 0.9, 0.99
min_split	10, 50, 100
max_depth	0, 5, 10, 20

위의 하이퍼 파라미터 조합들을 이용하여 성능을 평가하였다. 이때 최적의 하이퍼 파라미터 조합을 확인하기 위해서 각 조합들에 의해 계산된 BCR들을 기준으로 내림차순 하여 가장 상단에 위치한 조합을 최적의 모델로서 선정하여 이용하였다. 이때 나온 결과들에 대해 일부분만 살펴보도록 하겠다.

min_criteron	min_split	max_depth	ACC	BCR
0.80	50	0	0.6630760	0.5148813
0.80	50	10	0.6630760	0.5148813
0.80	50	20	0.6630760	0.5148813
0.80	10	0	0.6636744	0.5145314
0.80	10	10	0.6636744	0.5145314
0.80	10	20	0.6636744	0.5145314
0.80	100	0	0.6628765	0.5095352
0.80	100	10	0.6628765	0.5095352
0.80	100	20	0.6628765	0.5095352
0.99	10	0	0.6493118	0.4725862
0.90	100	10	0.6582885	0.4624264
0.90	100	20	0.6582885	0.4624264
0.80	50	5	0.6349491	0.3239237
0.80	100	5	0.6349491	0.3239237
0.80	10	5	0.6351486	0.3236207
0.90	50	5	0.6369439	0.2935010
0.90	100	5	0.6369439	0.2935010
0.90	10	5	0.6371434	0.2932254
0.99	10	5	0.6369439	0.2911896
0.99	50	5	0.6369439	0.2911896
0.99	100	5	0.6369439	0.2911896

빨간 박스 속에 있는 값은 상단의 BCR이 높은 상위 10개의 조합들에 대한 결과를 보여주며 파란 박스 속은 하위 10개를 보여주고 있다. 가장 높았던 하이퍼 파라미터 조합은 가장 위에 있는 조합이 될 것이다. 이를 이용하여 다시 한번 성능을 확인하기 위해 CART_trn과 CART_val을 결합한 후 Cart_data를 생성하여 다시 학습 시킨후 CART_tst에 대하여 다음을 구해 보았다

Confusion Matrix			
CART	1	2	3
1	245	295	17
2	187	2686	540
3	26	1109	911

Performace Eval	
ACC	0.6386303
BCR	0.5361642

위에서도 확인을 할 수 있었지만 BCR값이 MLR보다 더 높게 나온다. 이 말은 범주를 예측하는

데 있어 더 골고루 예측을 하고 있다는 의미가 될 것이다. 물론 2번 범주의 비율이 높기 때문에 혼동행렬에서 보이는 것과 같이 2번 범주를 예측하는 비율 또한 높을 것이다. 하지만 더 골고루 MLR보다 예측률이 더 높아졌다는 점은 확실하게 볼 수 있었다.

<데이터의 5%사용>

하이퍼 파라미터는 10%의 데이터를 돌려을 때와 동일하게 넣어주었다. 하지만 선정된 하이퍼 파라미터는 10%때와는 다르게 나왔다는 것을 확인할 수 있었다. 하위의 결과들은 더 이상 첨부하지 않고 가장 높은 BCR을 나타낸 결과창만 보도록 하겠다.

```
      min_criteron min_split max_depth       ACC       BCR
[1,]         0.90        10         0 0.6475575 0.4102989
```

기존에는 위의 표에 나와있는 순서대로 0.8, 50, 0의 하이퍼 파라미터 조합이 선정되었는데 현재는 0.9, 10, 0이 선정 되었고, BCR과 ACC는 데이터가 10%였을 때보다는 더 낮은 결과가 나왔다.

Confusion Matrix			
CART	**1**	**2**	**3**
1	**131**	**177**	**1**
2	**119**	**1524**	**138**
3	**12**	**706**	**320**

Performace Eval	
ACC	**0.6313939**
BCR	**0.4817949**

하지만 그럼에도 불구하고 ACC와 BCR을 살펴보면 MLR을 수행하였을 때보다는 더 높은 값이 나왔다는 것을 확인할 수 있다. 그 이유에 대해서 생각을 해보자면 역시나 3번 범주와 1번 범주에서 올바르게 맞춘 결과의 빈도수가 더 올라갔기 때문이라고 생각해볼 수 있을 것이다.

ANN

<데이터의 10%사용>

Hyper Parameters	
Hidden node 수	**10, 20, 30, 40**
maxit	**100, 150, 200, 250, 300**
rang	**0.3, 0.5, 0.7, 0.9**

ANN의 경우에는 위와 같은 하이퍼 파라미터들을 이용하여 보았다. 총 80가지의 모델이 생성되었고, 그 중에서 최적의 하이퍼 파라미터 조합을 찾아보고자 노력하였다. CART와 동일하게 상위 열가지와 하위 열 가지 조합들에 대해서 살펴보자면 다음과 같다.

```
      nH Maxit rang       ACC       BCR
 [1,] 40   300  0.7 0.7277080 0.6454369
 [2,] 40   250  0.5 0.7289048 0.6436603
 [3,] 40   250  0.3 0.7203271 0.6340381
 [4,] 40   300  0.3 0.7079593 0.6303193
 [5,] 40   250  0.7 0.7271095 0.6299825
 [6,] 40   200  0.7 0.7189308 0.6278055
 [7,] 40   150  0.5 0.7153401 0.6253849
 [8,] 40   200  0.3 0.7123479 0.6229619
 [9,] 30   250  0.3 0.7109515 0.6226740
[10,] 40   150  0.3 0.7195292 0.6217633
```

```
[70,] 40   150  0.7 0.5745063 0.0000000
[71,] 30   200  0.7 0.5745063 0.0000000
[72,] 20   250  0.7 0.5745063 0.0000000
[73,] 20   300  0.7 0.5745063 0.0000000
[74,] 30   100  0.9 0.5745063 0.0000000
[75,] 10   200  0.9 0.5745063 0.0000000
[76,] 40   200  0.9 0.5745063 0.0000000
[77,] 10   250  0.9 0.5745063 0.0000000
[78,] 40   250  0.9 0.5810892 0.0000000
[79,] 10   300  0.9 0.5745063 0.0000000
[80,] 20   300  0.9 0.5745063 0.0000000
```

ANN의 경우 가장 최적의 하이퍼 파라미터의 조합으로 나온 ACC와 BCR은 MLR과 CART중에서 가장 높은 값을 가지고 있다는 것을 확인할 수 있었다. 하지만 반대로 하위 조합들을 살펴보면 BCR이 0값이 나오게 되는 경우가 상당히 많았다는 것 또한 확인할 수 있었다. 최적의 하이퍼 파라미터 조합은 nH가 40, Maxit이 300, rang이 0.7이였을 때 였다는 것을 확인하였고, 이 조합을 가지고 Confusion Matrix와 성능을 확인해보았다.

Confusion Matrix			
ANN	1	2	3
1	285	225	47
2	88	2818	507
3	24	779	1243

Performace Eval	
ACC	0.7224069
BCR	0.6355064

가장 두드러지게 보였던 점은 3번 범주를 예측하여 맞춘 경우가 MLR과 CART에 비해 매우 올랐다는 것을 확인할 수 있었다. 이에 따라 ACC와 BCR도 높아졌고 확실히 이전 모델들에 비해 성능이 향상되었음을 확인할 수 있었다. 하지만, 위에서도 언급하였듯, 하이퍼 파라미터의 조합에 따라서 BCR의 값이 천차만별 이였다는 것을 확인할 수 있었고, 하이리스크가 따르는 모델이라는 것을 확인할 수 있었다.

<데이터의 5%사용>

ANN의 경우 다음과 같은 하이퍼 파라미터 조합이 가장 높은 BCR값을 도출하였다.

```
     nH Maxit rang       ACC       BCR
[1,] 40   200  0.9 0.7424304 0.6698273
```

이 역시도 nH를 제외한 나머지 하이퍼 파라미터들의 값이 10%의 데이터를 사용하였을 때와 다른 값을 도출하였다.

Confusion Matrix			
ANN	1	2	3
1	159	112	38
2	40	1448	293
3	6	374	658

Performace Eval	
ACC	0.7241039
BCR	0.6424765

혼동행렬을 살펴보자면 역시나 ANN이 10%에서도 그랬듯이 위의 두가지 분석 방법을 사용하였을 때보다 1번 범주와 3번 범주를 제대로 예측한 빈도수가 더 높아졌고 ACC와 BCR 또한 가장 높은 값을 산출해냈다는 것을 확인할 수 있었다.

<! 2번 문제부터는 10%의 데이터에 대한 자료를 첨부하는 것이 더 이상 의미가 없다고 판다하여 5%의 데이터를 사용하였을 때 나온 결과 값들에 대한 분석만을 진행하도록 하겠습니다 !>

[Q2] CART의 Bagging 모델을 Bootstrap의 수를 증가시키면서 분류 정확도를 평가

Bagging reuslt in order		
Bootstrap	Acc	BCR
60	0.713363	0.550942
270	0.704885	0.537048
120	0.706096	0.537013
180	0.70973	0.535125
90	0.706904	0.535121
30	0.708922	0.533889
300	0.708518	0.533405
150	0.705289	0.532873
210	0.7065	0.532271
240	0.705692	0.524787

Question 1에서 선정된 CART의 하이퍼 파라미터를 이용하여 30부터 300까지 30씩 증가시켜가며 분류 정확도를 파악해보았다. 이때에 사용한 코드는 cforest() 함수를 사용하였고, 성명 변수 후보 개수를 0으로 설정하여 변수가 임의로 선택되는 것을 방지해 주었다. 10개의 Boostrap별로 도출된 결과값을 BCR을 기준으로 내림차순 해볼 결과는 다음과 같았다. 현재 Bootstrap이 **60개**일 때 BCR의 값이 가장 높게 나온 것을 확인할 수 있었으며 동시에 ACC의 값도 가장 높은 값을 내고 있다는 것을 확인할 수 있다. 여기서 한가지 알 수 있는 점은 항상 ACC가 BCR보다 높게 나오는 것을 확인하였다. 이 데이터 셋 자체의 특징 중 하나가 class 2의 비율이 압도적으로 높다는 것이었는데 당연히 2번 범주를 많이 선택할수록 단순 정확도는 오를 수 밖에 없는 구조이기 때문에 너무나도 당연한 결과라는 것을 인지할 수 있었다. 그렇다면 최적의 Bootstrap 개수를 가지고 모델을 돌려 본 후, 단일 모델을 돌렸을 때와 어떤 식으로 다른 지를 비교하는 과정을 거쳐 보도록 하겠다.

Confusion Matrix			
CART_B	**1**	**2**	**3**
1	**122**	**186**	**1**
2	**44**	**1636**	**101**
3	**7**	**569**	**462**

Performace Eval	
ACC	**0.7097187**
BCR	**0.5444882**

혼동행렬과 성능 평가를 해본 결과는 위와 같이 산출 되었다는 것을 확인해 볼 수 있었다. 우선 정확도와 BCR은 전부 기존의 단일 모델에 비해 성능이 상향되었다는 것을 확인해볼 수 있었다. 혼동행렬에서 이를 살펴보자면 기존의 단일 모델에 비해서 1번 범주에 대한 예측률을 9개 정도 덜 맞췄다는 것을 확인할 수 있었다. 하지만 2번 범주와 3번 범주에서 100개가 넘는 개수의 데이터를 올바르게 맞춤으로써 성능이 향상되었다는 점을 확인해볼 수 있었다.

[Q3] Random Forest 모델의 Tree의 수를 증가시키면서 분류 정확도를 평가

Bagging reuslt in order		
Tree	Acc	BCR
30	0.809851	0.725472
240	0.80864	0.721577
300	0.813888	0.721515
210	0.811869	0.719774
270	0.810254	0.716545
180	0.807025	0.715114
120	0.80864	0.714456
60	0.806621	0.711314
150	0.807025	0.710056
90	0.803795	0.709446

문제의 지문에서 타겟 변수에 대한 설명 변수의 중요도를 산출할 수 있도록 코드를 수행하라는 말이 있어 importance=TRUE라는 코드를 삽입하여 돌려 주었다. Bootstrap의 수는 문제 2번 과 동일한 수치로 설정하여 주었다. 또한 분류 예측이므로 타입은 class로 해주었고 하이퍼 파라미터는 문제 1에서 CART에 대해 찾은 최적의 조합을 대입하였다. 이때 가장 높은 BCR 값을 산출해낸 Tree의 개수는 **30개**였으며 정확도와 BCR은 왼쪽 표에 보이는 대로이다. 현재까지 진행한 모델들 중에서는 가장 높은 값을 산출해내고 있다는 것을 확인하였다. 하지만 아직까지도 ACC의 값이 BCR보다 높다는 점을 미루어 보아 어느 한 범주에 몰리는 현상이 아직 일어나고 있다는 것을 유추해볼 수 있었다.

현재 최적의 Bootstrap수를 구하였기 때문에 랜덤포레스트 모델에 적용을 시켜 모델을 돌려보았다. 변수의 중요

도에 대해서 먼저 어떤 결과가 나왔는지 확인해보도록 하겠다. 데이터의 중요도를 살펴보기 위해 중점으로 둔 수치는 Mean Decrease Accuracy 였으며 이는 어느 한 변수가 정확도 개선에 영향을 준 정도라고 생각하였다. 각 변수들의 중요들 살펴보자면 다음과 같다.

변수명	MeanDecreaseAccuracy
geo_level_1_id	19.93742426
geo_level_2_id	14.08435936
count_floors_pre_eq	13.56716169
height_percentage	12.06217348
age	11.91223366
has_superstructure_timber	11.11781907
geo_level_3_id	10.61273795
area_percentage	9.913614769
other_floor_type_q	7.87037555
foundation_type_r	7.614237686
has_superstructure_mud_mortar_stone	7.241455566
has_superstructure_cement_mortar_brick	6.568978402
roof_type_q	6.218173499
has_secondary_use	6.181696378
has_superstructure_adobe_mud	5.720715432
has_superstructure_rc_non_engineered	5.691228272
foundation_type_w	5.61347002
other_floor_type_x	5.559950253
position_t	5.536705245
position_s	5.345941125
count_families	5.234471849
has_superstructure_stone_flag	5.218407258
ground_floor_type_f	5.172152986
roof_type_n	4.98686941
plan_configuration_q	4.928824309
land_surface_condition_t	4.885153326
ground_floor_type_v	4.81638958
has_superstructure_mud_mortar_brick	4.753221729
has_secondary_use_agriculture	4.716562684
roof_type_x	4.600645906
plan_configuration_d	4.466076417
land_surface_condition_n	4.317288646
has_superstructure_other	4.2407047
legal_ownership_status_a	4.189294534
other_floor_type_j	4.10417921
has_superstructure_bamboo	4.093258643
foundation_type_i	3.681491969
has_superstructure_cement_mortar_stone	3.417653976
foundation_type_u	3.292544243
position_j	3.258269004
other_floor_type_s	3.116263018
position_o	3.072550588
legal_ownership_status_r	3.019535009
ground_floor_type_x	2.95723986
has_secondary_use_rental	2.899811136
has_superstructure_rc_engineered	2.783981236
foundation_type_h	2.783745706
legal_ownership_status_v	2.67912477
plan_configuration_u	2.621663681
has_secondary_use_hotel	2.487655833
has_secondary_use_other	2.360000265
land_surface_condition_o	2.189093211
has_secondary_use_school	1.463810841
plan_configuration_m	1.463642872
plan_configuration_a	1.383957992
plan_configuration_c	1.148359137
legal_ownership_status_w	0.955619038
ground_floor_type_z	0.539220273
plan_configuration_f	0
plan_configuration_n	0
plan_configuration_o	0
has_secondary_use_health_post	0
has_secondary_use_gov_office	0
has_secondary_use_use_police	0
plan_configuration_s	-0.006237809
ground_floor_type_m	-0.920609531
has_secondary_use_institution	-1.017095255
has_secondary_use_industry	-1.017095255

가장 영향을 많이 준 변수는 geo_level_1_id였다는 것을 확인할 수 있었고, -의 값이 나오며 전혀 유의미하지 중요도를 가진 변수들도 다수 있었다. 중요도를 평가하는 지표는 Mean Decrease Accuracy를 이용하였다.

다음으로 위에서 언급한 최적의 Tree 수를 기준으로 만든 모델을 CARt_Bagging과 비교를 해보도록 하겠다. 바로 혼동 행렬과 산출된 성능을 살펴보도록 하겠다.

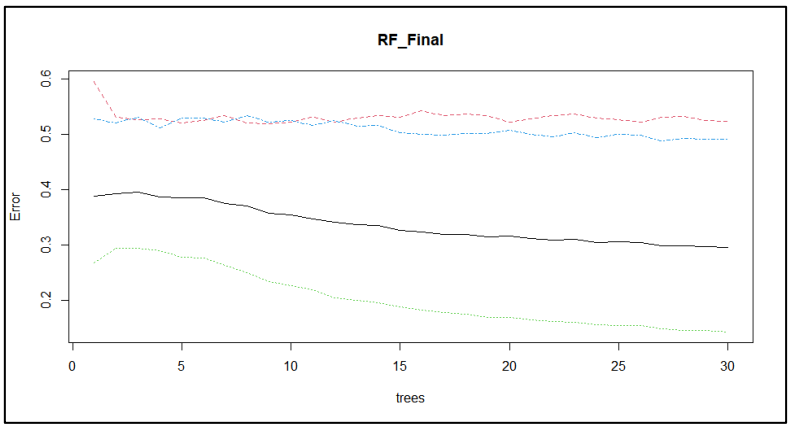
Confusion Matrix			
RF_model	**1**	**2**	**3**
1	**198**	**109**	**2**
2	**24**	**1680**	**77**
3	**5**	**348**	**685**

Performace Eval	
ACC	**0.8193734**
BCR	**0.7361197**

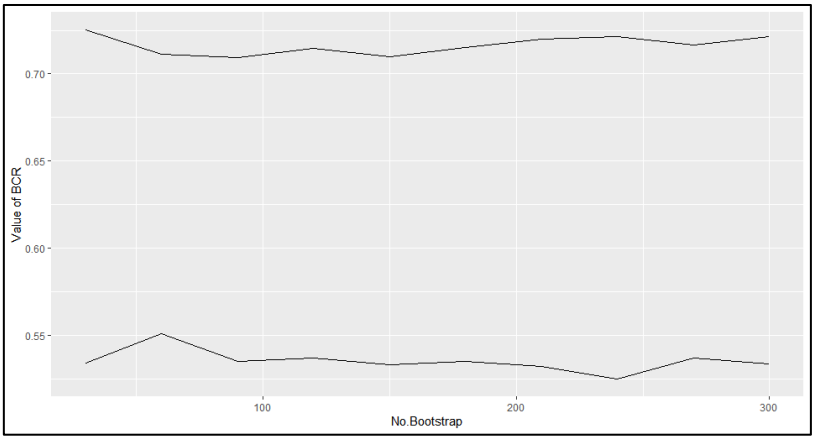
우선 CART_Bagging에 비해서 성능이 훨씬 상승하였다는 것을 확인할 수 있었다. CART_Bagging의 ACC보다 대략 0.19 정도 성능이 향상 되었고 BCR은 대략 0.2 정도 성능이 향상되었음을 확인하였다. 혼동 행렬도 살펴보자면 역시나 1범주, 2범주, 3범주 어느 것 하나 빠짐 없이 올바르게

예측한 빈도수가 다 높아졌다는 것을 확인할 수 있었다. 특히나 흥미로운 점은 1범주와 3범주에 대해 위의 모델들은 2범주를 더 많이 예측하였는데, 랜덤포레스트 모델은 각자 자신의 범주를 예측한 결과가 가장 높게 나왔다는 것을 알 수 있었다.

다음으로 살펴볼 것은 각 나무의 수마다 CART_Bagging과의 분류 정확도를 비교해보고자 한다. 이에 대한 그래프를 도시해보자면 다음과 같다.



현재 위의 그래프는 tree의 수에 따른 에러가 어떤 식으로 변하는지를 보여주는 그래프이다. 이를 먼저 확인하였다면 다음의 그래프르 살펴보도록 하겠다.



현재 위의 그래프는 X축은 나무의 수를 Y축은 BCR 값을 나타내는 그래프이다. 물론 위에서도 언급하였다시피 현재 랜덤 포레스트의 BCR값이 CART_Bagging 보다 높기 때문에 아래에 있는 선이 CART_B를 나타내고 있다는 것을 확인할 수 있다. 현재 바로 위의 그래프 말고도 tree의 개수마다 error가 어떤 식으로 변하는지에 대한 그래프만 보아도 큰 차이는 없지만 들쑥날쑥한 모양을 하고 있다는 것을 확인할 수 있다. 즉, 나무의 개수(Bootstrap의 개수)는 성능에 큰 영향을 미치고 있지는 않다는 것을 확인할 수 있는 매우 좋은 시각 자료임을 확인할 수 있다.

[Q4] [Q1]에서 찾은 최적의 ANN 단일모형을 30번 반복하여 테스트 정확도를 평가 및 Accuracy와 BCR의 평균 및 표준편차를 기록

우선 1번에서 선정된 ANN의 최적 하이퍼 파라미터는 nH: 40, Maxit: 200, rang: 0.9였으며 이를 30번 돌린 후 정확도와 BCR의 평균 및 표준편차를 기록하였다.

	ACC	BCR
[1,]	0.6902174	0.6037332
[2,]	0.7205882	0.6440591
[3,]	0.3487852	0.3493211
[4,]	0.3318414	0.0000000
[5,]	0.7404092	0.6478510
[6,]	0.7391304	0.6520557
[7,]	0.5693734	0.0000000
[8,]	0.3212916	0.3129090
[9,]	0.3372762	0.3428108
[10,]	0.7317775	0.6427212
[11,]	0.7231458	0.6352400
[12,]	0.5693734	0.0000000
[13,]	0.3395141	0.3250831
[14,]	0.5693734	0.0000000
[15,]	0.7225064	0.6295719
[16,]	0.7432864	0.6530854
[17,]	0.5693734	0.0000000
[18,]	0.7103581	0.3611627
[19,]	0.7605499	0.6723831
[20,]	0.3318414	0.3239853
[21,]	0.7461637	0.6669717
[22,]	0.5693734	0.0000000
[23,]	0.5693734	0.0000000
[24,]	0.5693734	0.0000000
[25,]	0.3395141	0.3243741
[26,]	0.3388747	0.3369630
[27,]	0.4475703	0.0000000
[28,]	0.7515985	0.6561313
[29,]	0.7304987	0.6304661
[30,]	0.5693734	0.0000000

30번을 돌린 것에 대한 결과는 위와 같았고, 평균과 표준편차를 기록한 결과는 아래와 같았다.

mean_ACC	sd_ACC	mean_BCR	sd_BCR
0.5733909	0.1656931	0.3470293	0.279967

생각보다 표준편차가 크다는 점에서 상당히 흥미롭다는 생각을 할 수 있었다.

[Q5] ANN Bagging

절대적인 시간의 문제와 컴퓨터의 CPU가 버티지 못하고 다수 터져 bootstrap의 수마다 30회 반복을 해야 하나 10회로 줄여 코드를 수행하였다. 그 결과 나온 각각의 Accuracy와 표준편차를 저장해주었다. 그 결과는 다음과 같이 나왔다는 것을 확인 할 수 있었다.

	Bootstrap	mean_ACC	sd_ACC	mean_BCR	sd_BCR
[1,]	30	0.6049872	0.03998887	0.10504941	0.1203452
[2,]	60	0.5957481	0.03034082	0.08781144	0.1064217
[3,]	90	0.6048167	0.03582130	0.12105302	0.1304035
[4,]	120	0.6096707	0.03661016	0.14681375	0.1341394
[5,]	150	0.6114962	0.03637054	0.15517142	0.1399637
[6,]	180	0.6079497	0.03643750	0.14215654	0.1399884
[7,]	210	0.6105499	0.03471888	0.15401314	0.1344620
[8,]	240	0.6091752	0.03385675	0.14974810	0.1325339
[9,]	270	0.6115729	0.03325332	0.16141167	0.1311212
[10,]	300	0.6090601	0.03318919	0.15177221	0.1309887

이렇게 저장한 결과값들을 가지고 최적의 bootstrap을 찾아보기 위해 아래와 같이 정리를 해보았는데, 우선 최적의 값을 찾기 위해 몇가지의 조건에서 생각을 해보았다. 현재 위에서 보면 알 수 있다시피 평균 BCR이 가장 높은 결과는 270일때라는 것을 확인할 수 있었다. 하지만 평균이 높다고 해서 좋기만 하지는 못할 것이다. 평균이 높다 한들 표준편차가 크면 의미가 매우 불안한

모델이라고 판단하였다. 그에 따라 물론 평균 BCR은 가장 낮아도 표준편차가 가장 적은 모델을 사용하는 것이 맞는 선택일 것이라는 생각을 하였고 그에 따라 최적의 bootstrap을 60을 살펴보았다. 하지만 전체적으로 바라본다면 데이터들의 평균 정확도와 BCR이 너무 낮다. 그러므로 그 다음으로 표준편차가 작은 **30이 최적일 것**이라는 결론을 내렸다.

우선 위에서 이미 한번 보여준 바 있지만 단일 모델을 30회 반복 하였을 때와, 비교를 해보자면 다음과 같다.

mean_ACC	sd_ACC	mean_BCR	sd_BCR
0.5733909	0.1656931	0.3470293	0.279967

Bootstrap	mean_ACC	sd_ACC	mean_BCR	sd_BCR
30	0.6049872	0.03998887	0.10504941	0.1203452

확실히 표준편차들은 많이 줄었다는 것을 확인할 수 있었으나, 평균 BCR은 오히려 좀 낮아진 듯한 모습을 보였다. 왜 이렇게 되었을까 에 대해 생각을 해보자면, 우선 반복 횟수가 같지 않다는 점에서가 첫번째 이유가 될 수 있을 것이다. 또한 데이터 히스토리가 계속 날아가 그때마다 같은 seed를 사용하였지만 random 하게 뽑혔기 때문에 데이터 자체의 특성이 이런 걸 수도 있다는 생각을 하였다.

[Q6] Adaptive Boosting(AdaBoost)

가장 먼저 사용한 하이퍼 파라미터들은 다음과 같다.

Hyper Parameters	
iteration	**50, 100, 200**
bag.frac	**0.1, 0.25, 0.5**

Ada boost는 개별적 모델을 순차적으로 학습하고 오분류된 샘플에 대해 더 많이 찾도록 가중치를 부여하는 방식이다. 그렇기에 이러한 모델을 구축하기 위한 하이퍼파라미터는 population size를 의미하는 iteration과 샘플링의 비율 bag.frac이 있다. 총 9가지 조합이 나오는데 adaboost는 CART를 따르므로 CART의 최적 파라미터를 할당한다.

위의 하이퍼 파라미터 조합들을 가지고 각각에 대한 정확도와 BCR을 측정하여 결과를 저장해주었는데 다음과 같이 나왔다는 것을 확인할 수 있었다.

iteration	bag.frac	ACC	BCR
50	0.50	0.6386758	0.4046823
200	0.25	0.6366572	0.3980511
50	0.10	0.6378684	0.3972885
50	0.25	0.6334275	0.3961522
100	0.50	0.6374647	0.3956551
200	0.50	0.6334275	0.3943421
100	0.25	0.6354461	0.3919435
200	0.10	0.6330238	0.3896935
100	0.10	0.6330238	0.3846187

위의 결과에서 알 수 있듯 최적의 하이퍼 파라미터 조합은 가장 첫 줄에 나온 것과 같이 iteration: 50 bag.frac:0.5 일 때 였다. 이를 가지고 모델을 돌려보았고 그에 따른 혼동 행렬과 성능 평가 지표는 다음과 같이 나왔다.

Confusion Matrix			
ADA_boost	1	2	3
1	79	221	2
2	53	1611	135
3	8	633	386

Performace Eval	
ACC	0.6636829
BCR	0.4448707

역시나 초반에 수행한 MLR에 비해서는 더 상향된 성능을 보여준다, 하지만 단순 CART, ANN 및 CART_Bagging, RF에 비해서는 성능이 비교적 좋지 못하다는 것을 확인할 수 있다. 그 이유에 대해서 살펴보자면 역시나 2번 범주를 가장 많이 선택하였고, 1범주와 3범주를 고르게 예측하지 못했다는 점에서 성능이 상승하지 않은 것으로 보인다. 또한 CART가 base learner이였는데, 부스팅의 목적 즉 bias를 줄이는 것을 원활하게 수행하지 못한 것으로 보인다. 그렇기 때문에 되려 BCR이 낮아졌다는 추측 할 수 있을 것이다.

[Q7] Gradient Boosting Machine(GBM)

Hyper Parameters	
tree_num	400, 500, 600, 700
shrinkage	0.05, 0.1, 0.15, 0.2

Gradient boost Machine 또한 ada boost와 같이 부스팅이다. 이 때 선정해줄 하이퍼 파라미터는 위에서 보는 것과 같다. 이때, tree_num 이라고 해놓은 것은 population의 수를 의미하며 shrinkage는 말그대로 줄임양을 의미하는 것이다. 여기서 유심히 볼 점은 여태까지 설정한 population 수보다 후보군으로 선정한 하이퍼 파라미터들의 값이 크다는 것을 확인할 수 있다. 단순히 랜덤포레스트와 ADA Boost 보다 더 많은 population 설정을 해주는 것이 좋을 것이라 판단하였기 때문이다.

tree_num	shrinkage	ACC	BCR
700	0.2	0.690755	0.546653
700	0.15	0.687122	0.545653
600	0.2	0.691562	0.544091
400	0.2	0.677432	0.539768
600	0.15	0.682277	0.539089
500	0.15	0.681066	0.53096
400	0.15	0.679047	0.530272
500	0.2	0.682277	0.52007
700	0.1	0.679855	0.516008
600	0.1	0.676221	0.514766
500	0.1	0.677432	0.514615
700	0.05	0.672588	0.503273
400	0.1	0.668551	0.50219
600	0.05	0.671377	0.500027
500	0.05	0.664514	0.491266
400	0.05	0.66088	0.468441

위의 결과를 보면 확인할 수 있다시피 현재 가장 BCR이 높게 나온 하이퍼 파라미터는 population이 700일때와 shrinkage가 0.2 일 때 라는 것을 알 수 있다. 이렇게 산출해낸 최적의 하이퍼 파라미터를 이용하여 모델을 돌려본 결과 다음과 같은 결과가 나옴을 확인할 수 있다.

Confusion Matrix			
GBM	**1**	**2**	**3**
1	**104**	**61**	**3**
2	**191**	**1518**	**473**
3	**8**	**220**	**551**

Performace Eval	
ACC	**0.6946931**
BCR	**0.6731384**

위의 결과를 살펴보자면 확실히 지금까지 돌린 모델 들 중에서 BCR의 값이 두번째로 높아졌다는 것을 확인할 수 있다. 한가지 확인할 수 있는 점은 ACC와 BCR이 얼마 차이가 나지 않는 다는 점은 각 범주별로 골고루 예측을 잘 했다고 볼 수 있을 것이다. GBM 모델을 통해 각 변수 별 중요도를 살펴보면 다음과 같은 결과가 나온다. 비록 ACC가 가장 좋지는 않지만 확실히 BCR이 많이 높아짐으로써 상위권의 성능을 보여주고 있다고 판단된다.

변수명	중요도
geo_level_1_id	29.66215697
geo_level_2_id	13.29448333
geo_level_3_id	12.04715514
foundation_type_r	7.37847329
age	4.55498236
area_percentage	3.99395022
ground_floor_type_v	3.2265808
height_percentage	2.84070967
foundation_type_i	1.89158094
count_floors_pre_eq	1.55997849
has_superstructure_mud_mortar_stone	1.25425585
roof_type_x	1.24292315
has_superstructure_cement_mortar_brick	0.91456315
has_secondary_use_other	0.86823663
other_floor_type_s	0.78027886
legal_ownership_status_a	0.71747527
foundation_type_h	0.71701105
foundation_type_w	0.68893135
has_superstructure_mud_mortar_brick	0.68645554
has_superstructure_rc_engineered	0.65866545
count_families	0.62942466
plan_configuration_q	0.57993482
other_floor_type_q	0.5571996
has_superstructure_other	0.54643519
has_secondary_use_rental	0.53030021
ground_floor_type_z	0.51782418
position_t	0.47042362
has_secondary_use	0.46900691
legal_ownership_status_r	0.42589371
has_superstructure_cement_mortar_stone	0.42220705
has_superstructure_bamboo	0.40329551
plan_configuration_u	0.37774021
position_s	0.35143582
has_superstructure_adobe_mud	0.34650362
has_superstructure_timber	0.34188361
foundation_type_u	0.32242712
has_superstructure_stone_flag	0.31982998
has_secondary_use_agriculture	0.28910927
land_surface_condition_n	0.28788531
ground_floor_type_m	0.27558485
roof_type_q	0.27506905
has_superstructure_rc_non_engineered	0.2574817
has_secondary_use_hotel	0.23946634
position_o	0.23829614
plan_configuration_d	0.23019654
legal_ownership_status_w	0.17855241
legal_ownership_status_v	0.16106183
land_surface_condition_t	0.15800812
position_j	0.15385997
ground_floor_type_x	0.1412937
land_surface_condition_o	0.13163852
ground_floor_type_f	0.12510059
roof_type_n	0.0977556
other_floor_type_j	0.08258579
other_floor_type_x	0.05152973
plan_configuration_s	0.03491119
plan_configuration_a	0
plan_configuration_c	0
plan_configuration_f	0
plan_configuration_m	0
plan_configuration_n	0
plan_configuration_o	0
has_secondary_use_institution	0
has_secondary_use_school	0
has_secondary_use_industry	0
has_secondary_use_health_post	0
has_secondary_use_gov_office	0
has_secondary_use_use_police	0

각 변수들의 중요도를 산출하여 살펴본 결과 위와 같은 결과가 나왔다는 것을 확인할 수 있다. 이를 살펴보기 위해 이전에도 수행한 랜덤포레스트와 비교를 해보자면 우선 가장 중요도가 높은 변수는 geo_level_1_id로 같았다는 것을 확인할 수 있다. 비록 중요도를 매기는 기준은 다르지만

중요도가 높은 1등과 2등의 변수가 같고 크게 어긋나지 않게 비슷한 결과를 볼 수 있다.

더 자세하게 살펴보자면 랜덤 포레스트에서 중요하다고 매겨진 변수의 상위 세개는 geo_level_1 , 2, count_floor_pre_eq였다. GBM에서는 geo_level_1 , 2, 3이였는데, 여기서 공통으로 들어간 두개의 변수를 보아 이 변수들이 가장 중요하게 영향을 미치고 있다는 것을 확인할 수 있었다. 물론 3등이 다르지만 랜덤포레스트 3등 변수가 GBM에서 나름 상위권에 있으며 GBM의 3등 변수 또한 랜덤 포레스트에서 상위권에 있는 것을 보아 큰 차이가 있지는 않다고 볼 수 있다.

〔Q8〕 총 여덟 가지의 모델 중 가장 우수한 모델

모델	ACC	BCR
RF_model	0.81937	0.73612
GBM	0.69469	0.67314
ANN	0.7241	0.64248
CART_B	0.70972	0.54449
CART	0.63139	0.48179
ADA_boost	0.66368	0.44487
MLR	0.59687	0.37768
ANN_B	0.60499	0.10505

지금까지 구해본 모델들의 성능을 정리해보면 위와 같이 내림차순으로 정리를 할 수 있다. 물론 지금까지 계속 그래왔던 것처럼 BCR을 기준으로 보았는데 그 이유는 2범주의 비율이 압도적으로 많았기 때문에 정말 자신의 범주를 얼만큼 정확히 예측했느냐를 보는 것이 맞다고 판단하였기 때문이다. 우선 결과적으로 보면 위의 모델들 중에서 압도적으로 높은 성능을 내고 있는 것은 Random forest임을 확인할 수 있다. ACC의 관점으로 BCR의 관점으로나 가장 높은 성능을 내고 있다는 것을 확인할 수 있다. 랜덤 포레스트는 복잡한 형태의 트리의 높은 분산을 낮추는 것이 목표이다. 랜덤 포레스트의 특징을 보면 하이퍼 파라미터와 변수의 중요도를 통해서도 알 수 있다시피 변수의 범주를 세분화하여 분류하게 된다. 이러한 면이 한쪽 범주로 치우쳐져 있는 데이터에 대해서 좋은 성능을 낼 수 있게끔 한 특징이 아닐까 라고 추측해본다.

〔Extra Question〕

가장 먼저 떠오른 방법은 샘플들의 개수를 전부 맞춰주는 것인데, 즉 각 범주에 해당하는 sample의 양을 맞춰 주는 것이다. 두가지의 방법이 있을 수가 있을 것이다. 가장 많은 샘플이 들어있는 범주의 샘플 양에 맞게 다른 범주 데이터를 늘리는 방법과 가장 샘플의 개수가 적은 범주의 샘플 양에 맞춰 나머지를 적게 추출하든가 하는 방법이다. 우선 교수님께서 제시하신 데이터에서 10%로만 사용하여 실험해보았다. 위에서 가장 좋은 성능으로 뽑힌 모델이 랜덤 포레스트였기 때문에 가장 샘플의 개수가 적었던 1번 범주(2520개)에 맞춰 나머지 범주들을 동일하게 추출하고 결과를 확인해보니 다음과 같았다.

Confusion Matrix(under sampling)			
RF_model	1	2	3
1	547	34	38
2	46	465	85
3	20	69	510

Performace Eval	
ACC	0.8390298
BCR	0.8373022

범주들 간의 데이터 양이 불균형 했을 때와는 확연하게 차이가 있다는 것이 확인 된다. 우선 BCR의 수치가 0.1이나 늘어났고 그에 따라 ACC 또한 당연히 상승하였다. 당연히 각 범주에 해당하는 데이터의 개수가 같기 때문에 골고루 예측을 더 잘하는 결과가 나올 수 밖에 없을 것이다.