2주차 Classification

3조: 안현준 이재현 강태환 이미르 이규민

Week 1 Abstract

기업 파산 확률 예측

```
X1 net profit / total assets
X2 total liabilities / total assets
X3 working capital / total assets
X4 current assets / short-term liabilities
X5 [(cash + short-term securities + receivables - short-term liabilities) / (operating
expenses - depreciation)] * 365
X6 retained earnings / total assets
X7 EBIT / total assets
X8 book value of equity / total liabilities

...

class bankruptcy
```

NA Imputation

- NA값이 데이터의 절반 이상이었던 `Attr37` 제거
- 각 열의 평균값으로 NA값 대체

Outlier

- min-max scaling → 값이 1인 행 제거

Feature Selection

- 역수 관계, 선형 관계에 있는 변수 제거
- 변수 추가
  `Attr65` ROI (투자자본수익률) = `Attr1` / `Attr17` → 투자 효율성 측정

`Attr66` ROE (자기자본수익률) = `Attr1` / `Attr10` → 자본 활용 이익 평가

`Attr67` 신생기업 여부

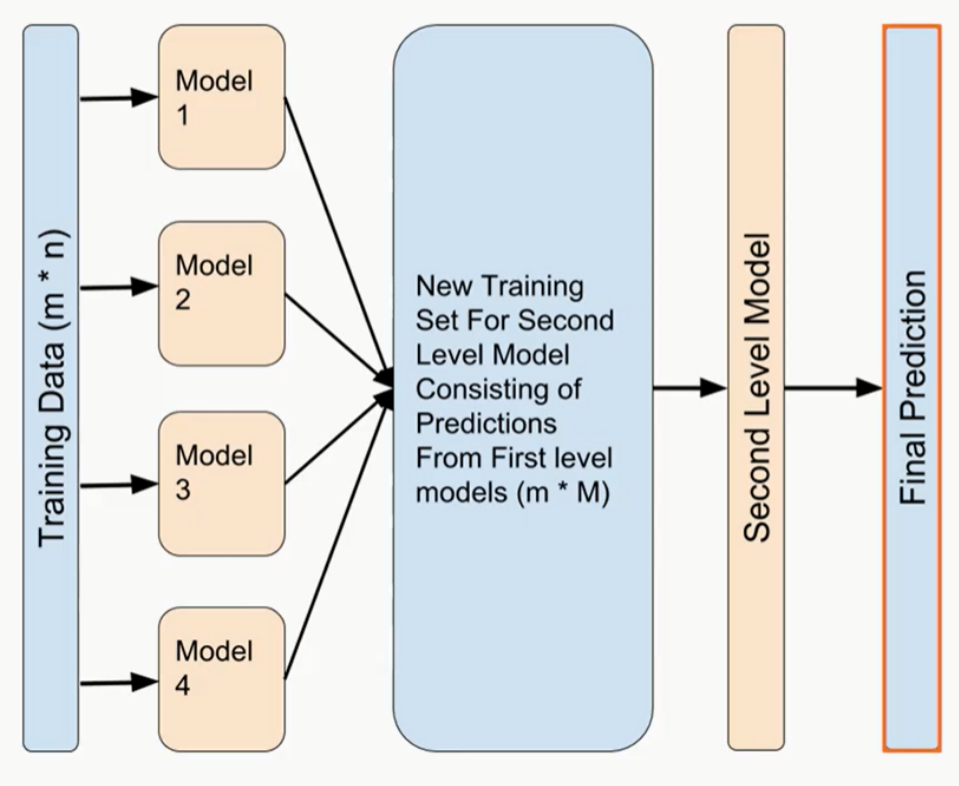
- 상관계수 행렬과 VIF값 통해 최종 변수 선택 → 총 **26개** 변수 선택!

Pre-Modeling

- 불균형 `class`
  - 파산 : 비파산 = 6494 : 361
- Stacking
  - Logistic Regression
  - SVM
  - Boosting(GBM)
  - Random Forest

📊 Model Selection & Modeling

Stacking



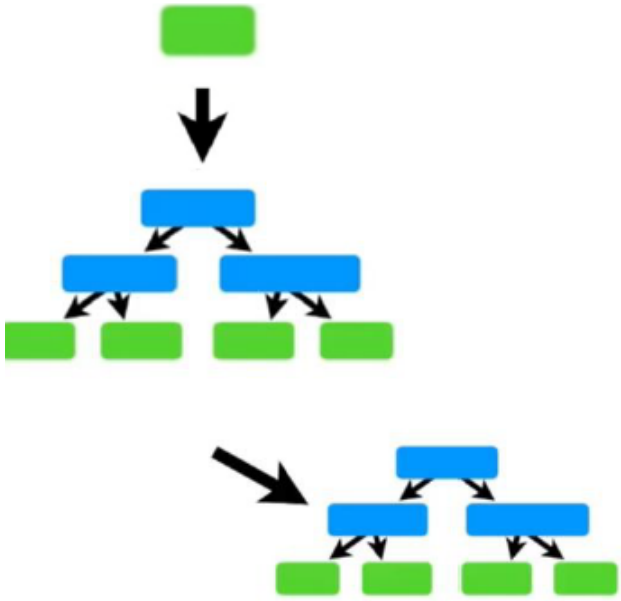
여러 가지 분류 모델들을 학습하여 예측한 값을 모아서 새로운 train set으로 활용

사용한 submodels: Boosting, Logistic regression, SVM, Random Forest

최종 model: XGboost

Sub Models

Boosting

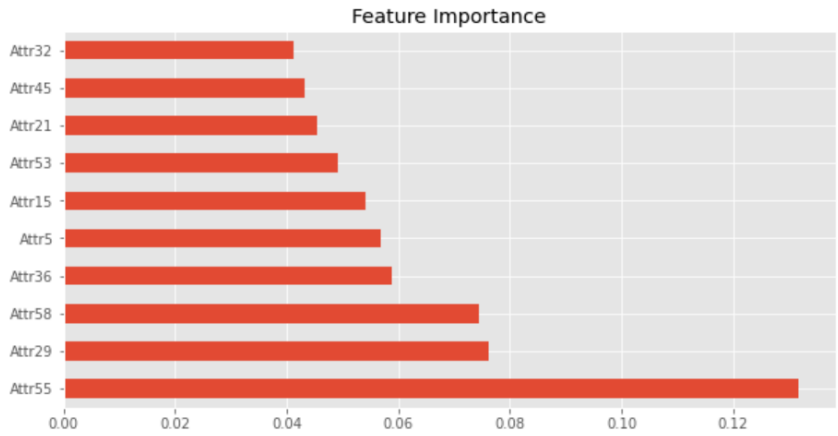


- Hyper-Parameter
  - n_estimators : Base tree의 개수
  - max_depth : 각 base tree의 depth
  - learning_rate

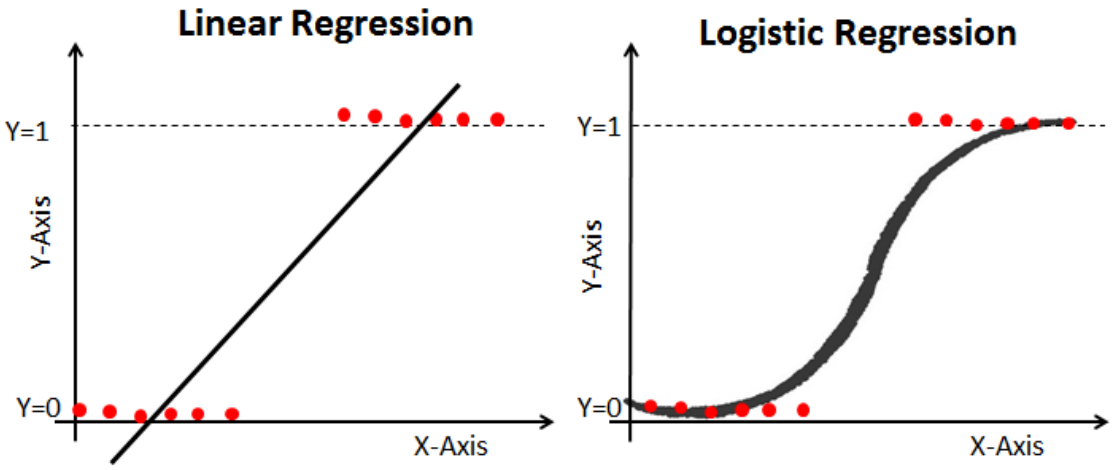
    Grid-Search 결과, n_estimators = 50, max_depth = 32, learning_rate=1로 설정

- Score
  - Label 데이터가 불균형하기 때문에 F1 score를 성능 측정 지표로 활용
  - Validation data에 대한 F1 Score: 0.17266

- Feature-Importance



Logistic Regression



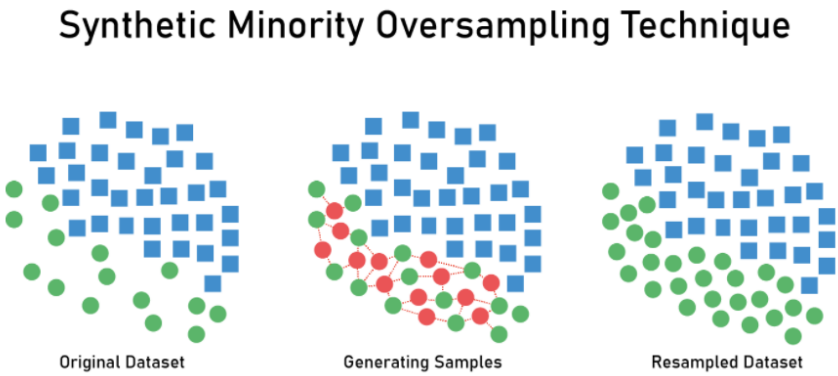
Pre-modeling

- train데이터에서 각 변수의 분포를 보고 outlier를 제거한 후 정규분포 형태로 변환 시도

  =⇒ 너무 낮은 f1-score가 나오는 관계로 원래 train_clean 데이터로 작업

- 파산과 비파산이 train data기준 매우 편차가 크기 때문에 Smote 패키지로 둘의 균형을 맞춰줌

Smote



- 데이터의 개수가 적은 클래스의 표본을 가져온 뒤 임의의 값을 추가하여 새로운 샘플을 만들어 데이터에 추가하는 오버샘플링 방식

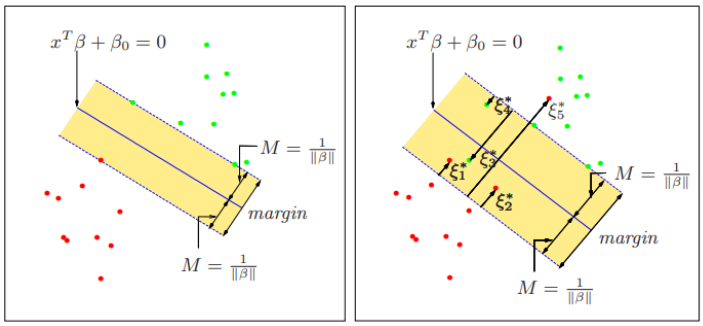
Hyper-Parameter Optimization

- Penalty: 'L1', 'L2'
- C: 0.01, 0.1, 1, 5, 10
- Grid Search로 C = 10, Penalty = L2 선택

Model Summary

- Accuracy: 0.584
- **F1-Score: 0.17**

Support Vector Machine



$$\max_{\beta,\beta_0,||\beta||=1} M$$

$$subject\ to\ y_i(x_i^T\beta + \beta_0) \geq M - \xi_i,$$

$$\xi_i \geq 0, \sum_{i=1}^{N} \xi_i \leq constant$$

Pre-Modeling & Smote

- Logistic Regression과 내용 공유

Kernel Candidates

✔️ Linear

□ Radial basis

□ 3rd-degree polynomial

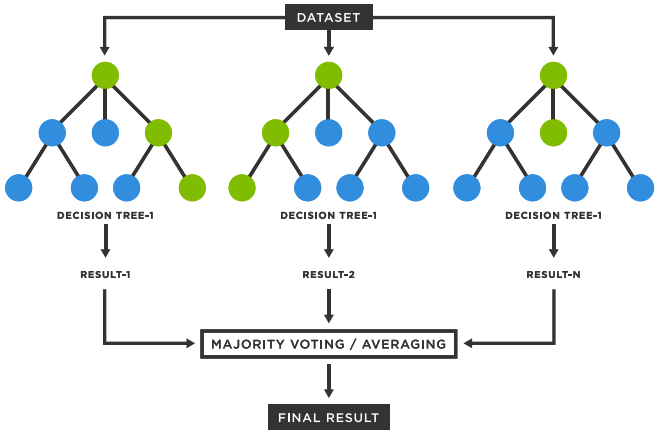
Parameter C

- The cost of slack variables ξ
- 0.5, 1, 5, 10, 50, ... C를 옮겨 가며 Training Data에서 F1 Score 비교
- 최종적으로 C=10,000 선택

Model Summary

- Accuracy: 0.7973
- **F1 Score: 0.1796**

Random Forest



substantial modification of bagging that builds a large collection of de-correlated trees and average them.

- tree를 만들때, 전체 변수를 고려하는 것이 아니라 m개의 변수를 랜덤으로 뽑아 트리를 구성함

- 이를 통해 de-correlated tree를 만들 수 있고, 결과적으로 예측의 분산을 더 효과적으로 낮출 수 있음.

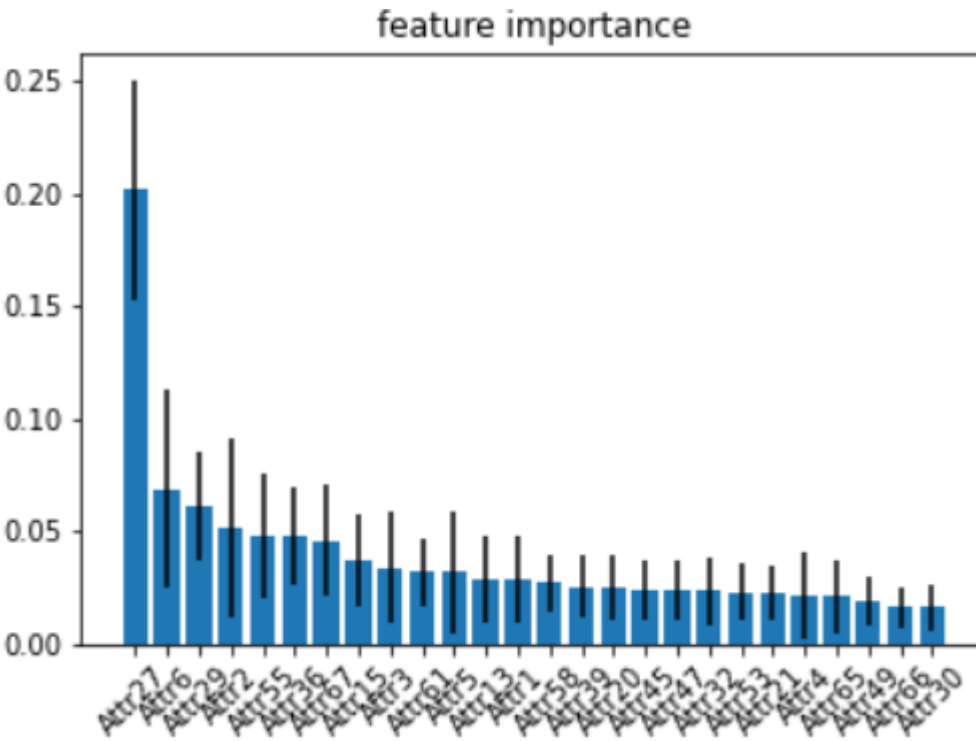
SMOTE

- imbalance data이기 때문에, logistic regression, SVM과 마찬가지로 SMOTE를 통해 over-sampling해주는 과정을 거침.

Parameters

- bootstrap = True
- max_features # 트리를 구성할 때 최대 몇개의 변수를 랜덤으로 뽑을 것인지
- n_estimators # 트리의 개수
- max_depth # 트리의 최대 깊이

grid search 결과, { bootstrap = True, max_features = 'sqrt', n_estimators = 1000, max_depth = 9 }로 결정



Model Summary

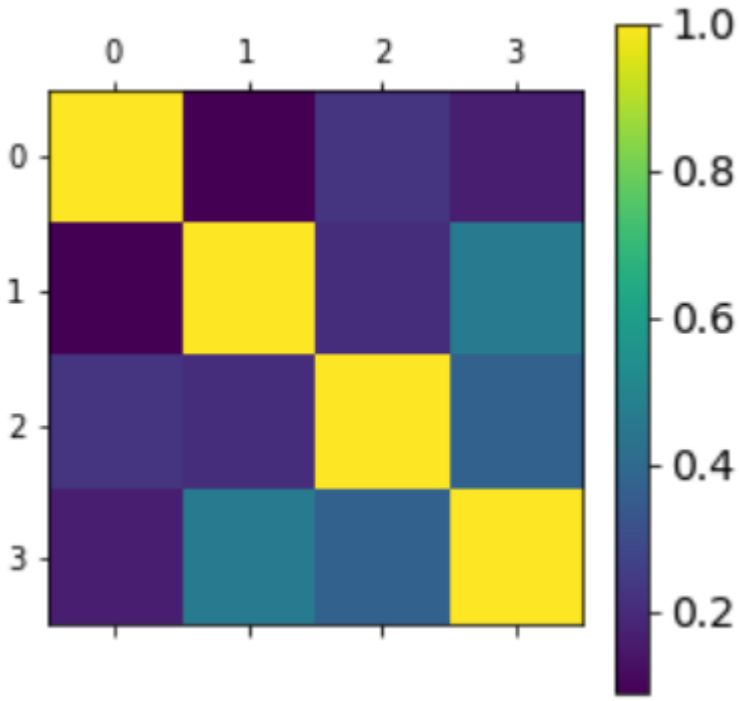
- out of bag accuracy - 약 0.88
- validation set f1 score - 약 0.20

📈 Prediction

- `train data`
  - 각 모델을 통해 나온 y hat

	boost	logistic	rf	svm
0	0	0.0	0	0
1	0	1.0	0	0
2	0	1.0	0	1
3	0	0.0	0	0
4	0	0.0	0	0
5	0	0.0	0	0
6	0	0.0	0	0
7	0	0.0	0	0
8	0	0.0	0	0
9	0	1.0	1	1
10	0	0.0	0	0
11	0	0.0	1	0
12	0	0.0	0	0

  - correlation plot



- Model summary

  Accuracy - 0.8919

  f1 score - 0.3877381938690969

	boost	logistic	rf	svm
0	0	0.0	0	0
1	0	1.0	0	0
2	0	1.0	0	1
3	0	0.0	0	0
4	0	0.0	0	0
5	0	0.0	0	0
6	0	0.0	0	0
7	0	0.0	0	0
8	0	0.0	0	0
9	0	1.0	1	1
10	0	0.0	0	0
11	0	0.0	1	0
12	0	0.0	0	0

0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	1
10	0
11	1
12	0

Boosting : 5955 vs 884

Logisitc Regression : 5234 vs 1605

Random Forest : 6838 vs 1

SVM : 5377 vs 1416

- `test data`

	boost	logistic	rf	svm		
0	0	0	0	0	**0**	0
1	0	0	0	0	**1**	0
2	0	0	0	0	**2**	0
3	1	0	1	0	**3**	1
4	0	0	0	0	**4**	0
5	0	0	0	0	**5**	0
6	0	0	0	0	**6**	0
7	0	0	0	0	**7**	0
8	0	0	0	0	**8**	0
9	0	0	0	0	**9**	0
10	0	0	0	0	**10**	0
11	0	0	0	0	**11**	0
12	0	0	0	0	**12**	0

- model

Boosting : 2599 vs 338

Logisitc Regression : 2136 vs 801

Random Forest : 2937 vs 0

SVM : 2375 vs 562

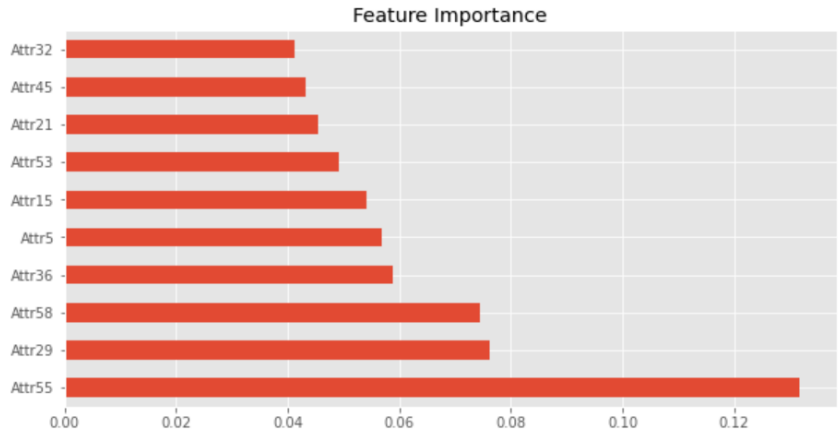
💸 Lend or Not?

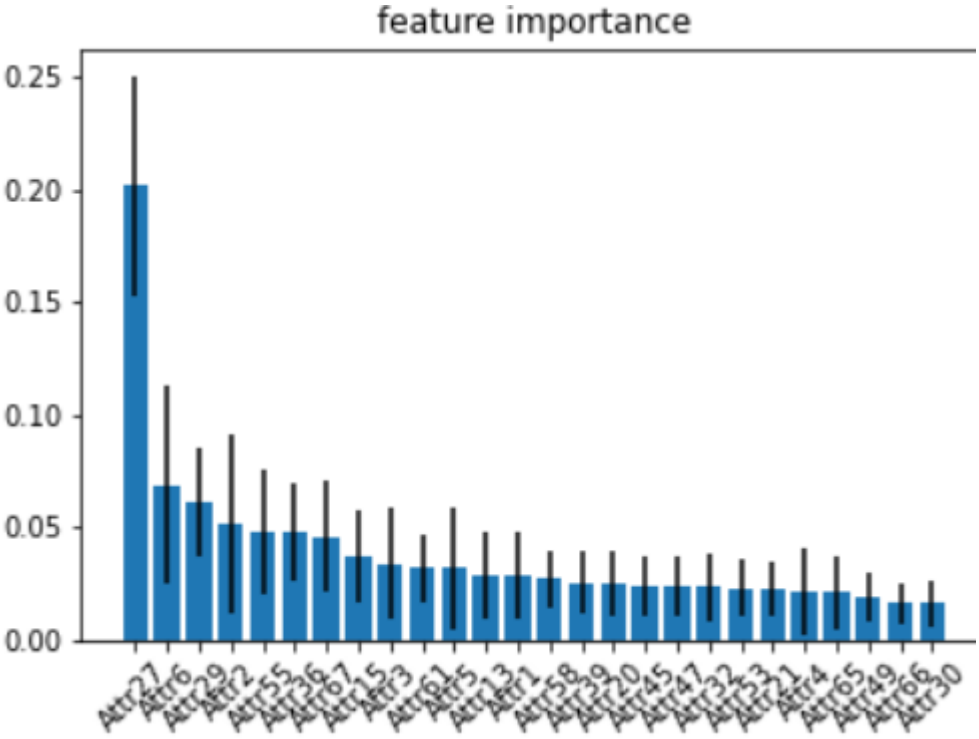
여신 여부 결정

0: 2643

1: 294

- 중요한 Feature





예측과 설명

현실 금융업계에서는 logistic regression을 많이 쓴다!

이유는 고객들에게 분류기준을 **설명**해야 하기 때문이다.

민원에 필요한 비용을 줄이기 위해서 다음의 조건을 갖춘 모델이 선호될 것이다.

- 쉽고, 여신 결격 사유가 무엇 때문인지 분명하게 알 수 있을 것
  - Neural Network 비교적 설명이 어렵다.
  - Logistic Regression Odds Ratio를 통해 X가 한 단위 변화할 때의 부도율 변화를 계산할 수 있다.
- 모형이 확정적 (deterministic) 일 것
  - 여신 여부를 '운'에 맡긴다고 오해할 수 있다.
  - Bagging, Random Forest 난수에 따라 결과값이 다르게 나올 수 있다.
  - Logistic Regression Gradient Descent를 통해 Loss Function의 최저점을 계산할 수 있다.

대신 예측력이 뛰어난 머신러닝 모델을 따로 관리하여 기업 판단에 충분히 반영할 수 있다!

❗ Conclusion & Limitation

의의

- 파산의 위험이 큰 기업을 사전에 확인하고 관리할 수 있다.
- 파산 위험에 따라 여신 여부를 합리적으로 결정할 수 있다.
- 여러 분류 모형을 적용하며 `21-SUMMER`, `21-FALL` 에서 배운 내용을 복습할 수 있었다.

한계점

- 변수 선택단계에서 너무 많은 observation을 제거했다.
- NA imputation
  - missing value가 많은 dataset이었기 때문에 이를 처리하기 위해서 모든 NA값을 평균으로 대체하여 사용했음.
  - 하지만 mice 패키지 등을 사용해 multiple imputation을 수행했다면 더 좋았을 것.