4팀

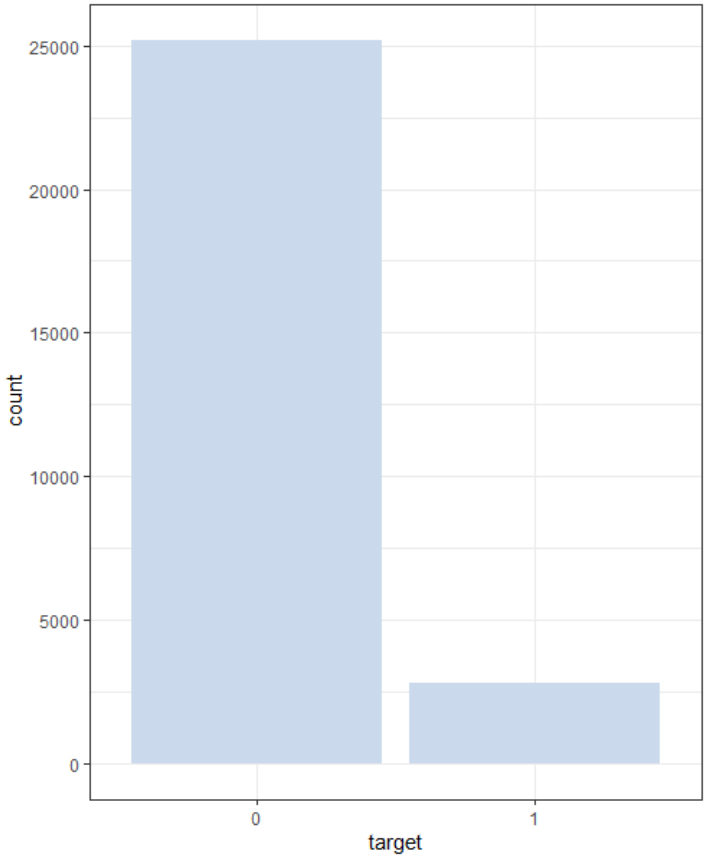
4팀
이재현
위재성
전효림
조장희
홍현경

INDEX

1

EDA

반응변수 Target의 분포



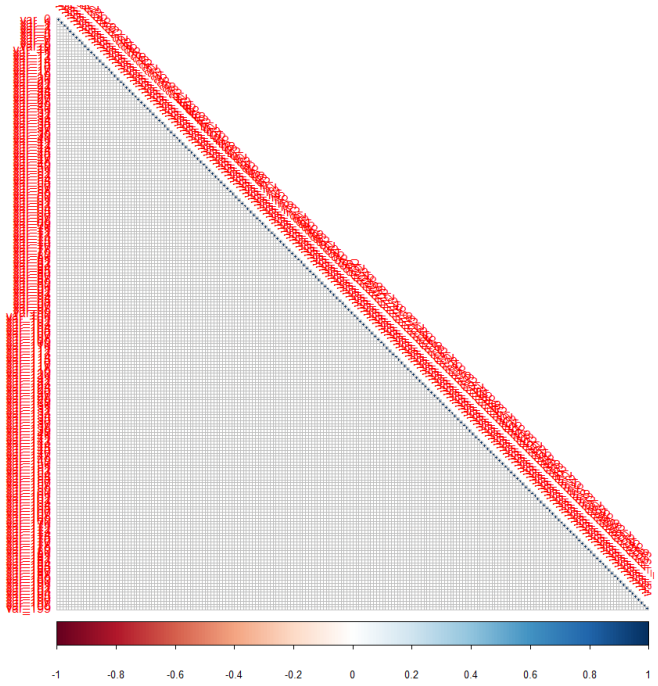
- 클래스 0은 25000개, 클래스 1은 2500개로 약10배 정도 차이
- 클래스 0이 훨씬 많은 imbalanced 데이터!
- 어떤 자료인지는 모르지만 쉽게 일어나지는 않는 사건으로 판단됨

결측치 확인

```
> sum(is.na(train))
[1] 0
```

결측치는 존재하지 않음을 확인.

설명변수 간의 상관관계



남량특집 광기의 그래프

```
> sum(abs(a)>0.03)
[1] 0
```

- 설명변수끼리 상관계수가 0.03이 넘는 경우가 존재하지 않음
- 즉, 서로 정말 관련이 없다.

2

전처리

전처리

차원 축소

- 여러가지 방법을 고민해본 끝에, PCA로 결정!
- 변수끼리 연관이 클 때 효과적인 방법
- 우리 변수들은 연관이 없었지만 후에 모델링 등의 결과가 나쁘진 않아서 고민보다 GO!

아미 은주야 잘지내니,,

200개의 설명변수

PCA

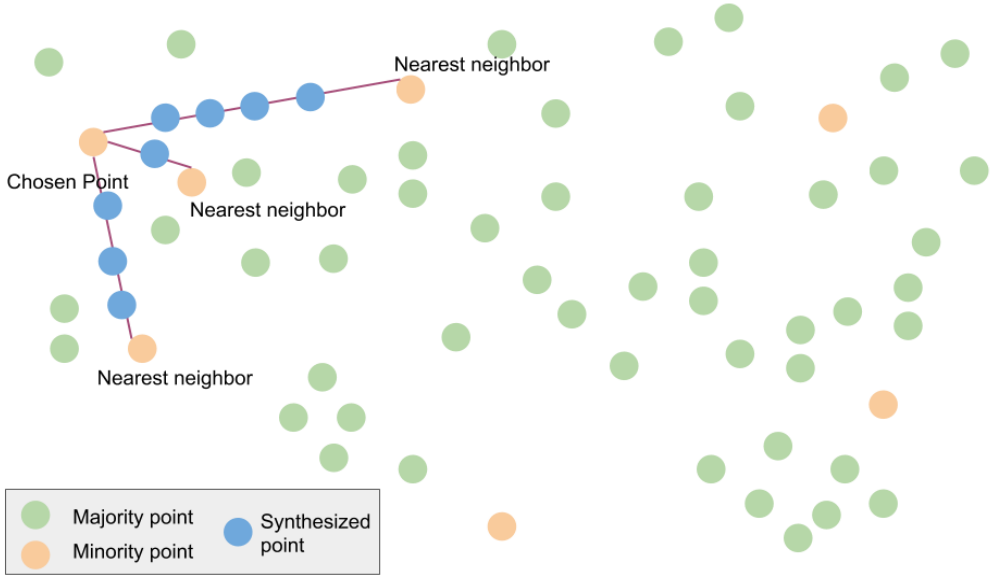
- 전체 분산의 95% 차지하는 변수 선택

189개의 설명변수

차원 축소 좀 더 공들여서 더 많이 해 볼 걸 하는 아쉬움,,ㅠ

불균형 처리

- 다양한 샘플링 방법 중에서 오버 샘플링을 하기로 결정!
- 그 중에서도 SMOTE를 사용!



2 전처리

불균형 처리

- 다양한 샘플링 방법 중에서 오버 샘플링을 하[illegible]
- 그 중에서도 SMOTE를 사용!

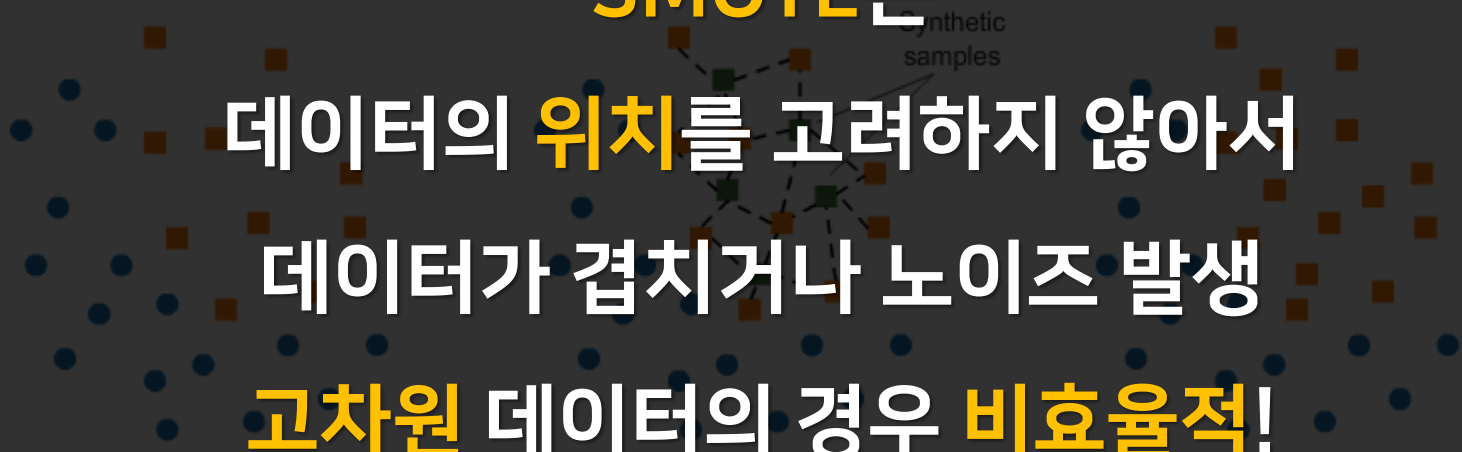
하지만,

SMOTE는

데이터의 위치를 고려하지 않아서

데이터가 겹치거나 노이즈 발생

고차원 데이터의 경우 비효율적!

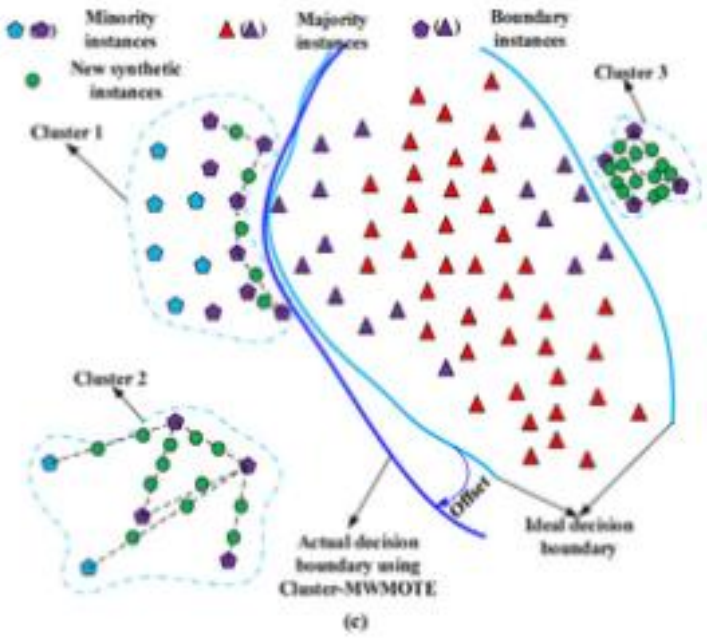


자세한 내용은 다음학기 범주팀의 클린업을 기대해주세용~>_<

불균형 처리

MWMOTE

- Majority weighted minority oversampling technique
- SMOTE의 단점을 보완한 여러 샘플링 방법 중 하나

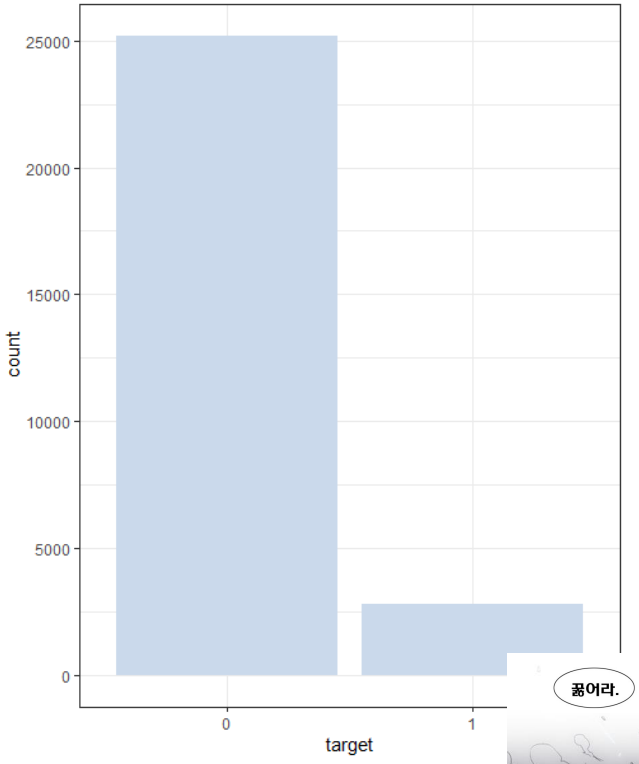


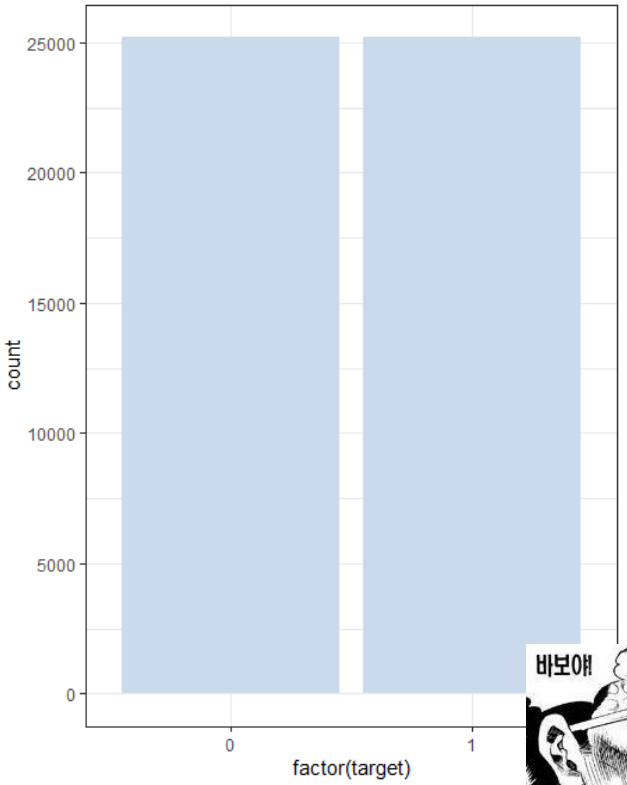
1. 소수 클래스 데이터와 가장 가까운 k개의 데이터와의 유클리드 거리를 구해 가중치 부여!
   (주위에 다수 클래스 데이터 많을 수록 많은 가중치 부여)
2. 소수 클래스 데이터 주위로 클러스터를 만들어서 가중치 큰 소수 클래스 데이터 중심으로 클러스터 안에 새 데이터 생성

전처리

불균형 처리

MWMOTE 적용 결과





전처리

최종 데이터셋


와~!!!
4팀이
쓰기로 한
데이타셋이
다!
데이터
"셋"?
21
원본
데이터에
요!
일단
아무것도
안 건드리고
한번 해볼래요
PCA만
적용한
불균형
데이터에
요!
차원축소
제대로 된 거
맞는거죠?
PCA와
MWMOTE
모두 적용
한 데이터
에요!
4팀의
야심작!
나도 답이
없으면
4팀은
물거에요

2 전처리

최종 데이터셋

과연 데이터 " 셋"의 운명은?

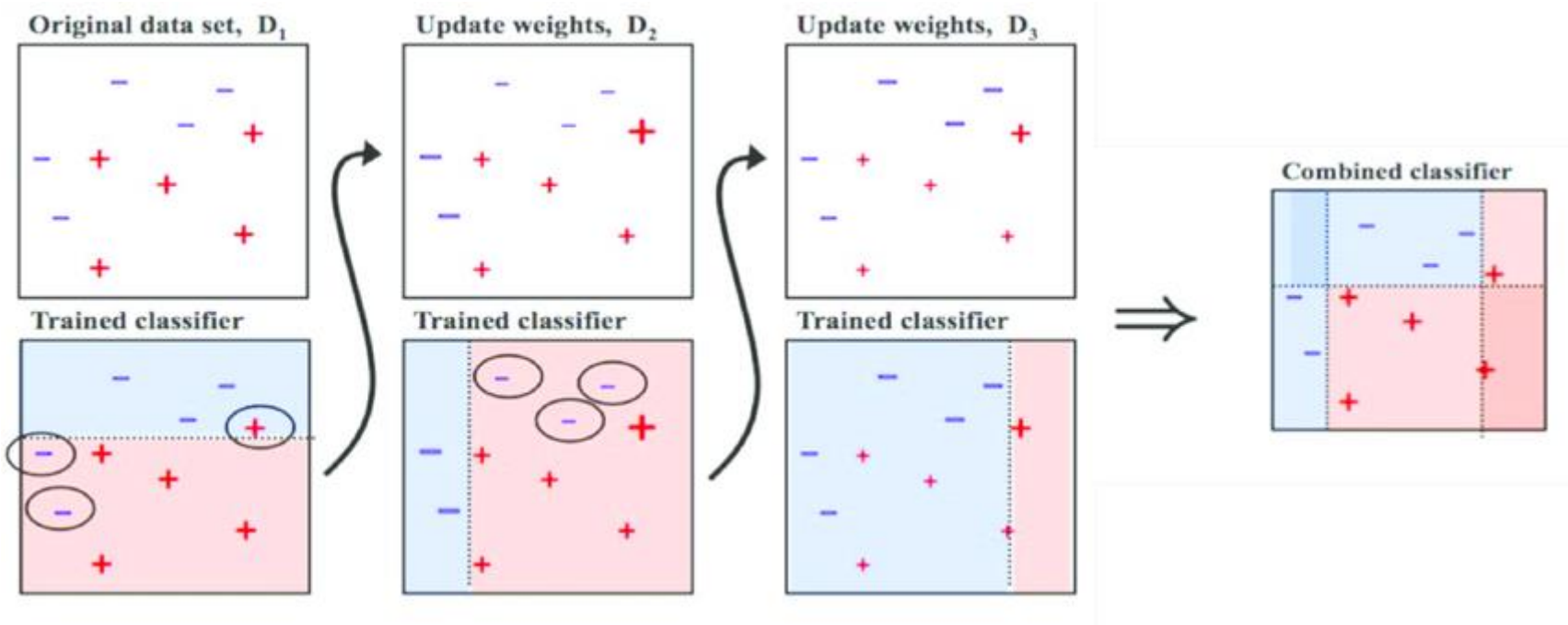
BGM♬
인간극장 끝나는 노래

3

모델링

AdaBoost

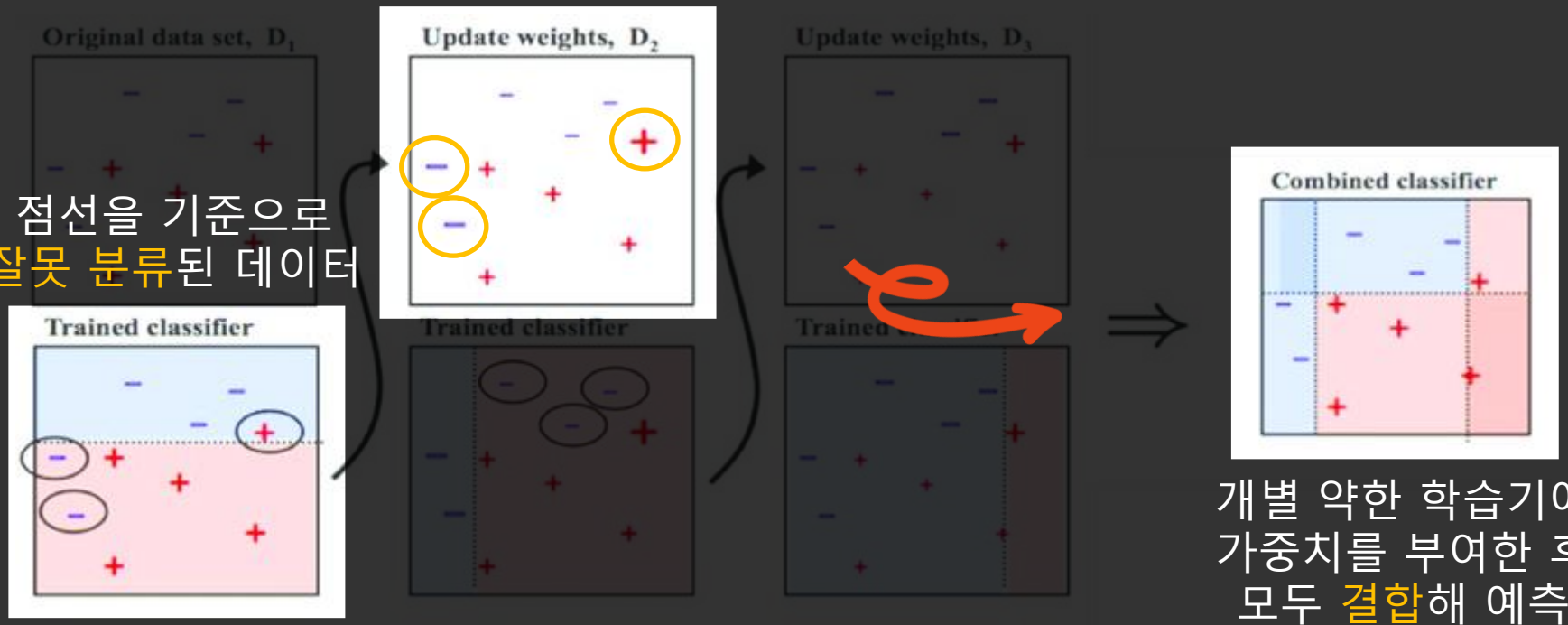
여러 개의 약한 학습기를 순차적으로 학습 / 예측
잘못 예측한 데이터에 가중치를 부여하여 오류 개선



AdaBoost

여러 개의 약한 학습기를 순차적으로 학습 / 예측
잘못 분류된 데이터의 가중치를 부여하여 오류 개선

가중치 업데이트



점선을 기준으로
잘못 분류된 데이터

개별 약한 학습기에
가중치를 부여한 후
모두 결합해 예측

AdaBoost

AdaBoost 하이퍼파라미터 튜닝

best_estimators

학습에 사용할 알고리즘

↓

default 사용

n_estimators

약한 학습기의 최대 개수

↓

n_estimators = 600

learning_rate

각 부스팅 iteration에서 분류기에 적용되는 가중치

↓

learning_rate= 0.05

AdaBoost

파라미터 튜닝

best_estimators / n_estimators / learning_rate

최종 결과

parameter

최적의 파라미터 탐색

best_estimators=default
n_estimators=600
learning_rate=0.05

AdaBoost

AdaBoost 하이퍼파라미터 튜닝 결과

- PCA만 적용한 train 데이터를 train : val = 7 : 3으로 나누어 학습
- train set에 오버샘플링 적용, validation set에는 적용하지 않음

F1 Score = 0.38905

3일동안 튜닝만 했지만^^;;

CatBoost

수치형 변수밖에 없지만..우선 해보자^^,,

범주형 변수가 많은 데이터셋에서 우수한 예측 성능을 보이는 모델

→ XGBoost, LGBM과 달리 파라미터에 덜 민감

→ Ordered Boosting / Random Permutation 이용
: 순서에 따라 모델을 만들고 예측

→ 효과적인 범주형 변수 처리

✓ Ordered Target Encoding

✓ Categorical Feature Combinations

CatBoost

CatBoost 하이퍼파라미터 튜닝

iterations

트리의 최대 개수를 규정

iterations = 83

max_depth

트리의 최대 깊이를 규정

max_depth = 13

learning_rate

gradient step 조정에 적용되는 계수

learning_rate= 0.15

CatBoost

파라미터 튜닝

iterations/ max_depth / learning_rate

최종 결과

parameter

최적의 파라미터 탐색

iterations=83
max_depth=13
learning_rate=0.15

CatBoost

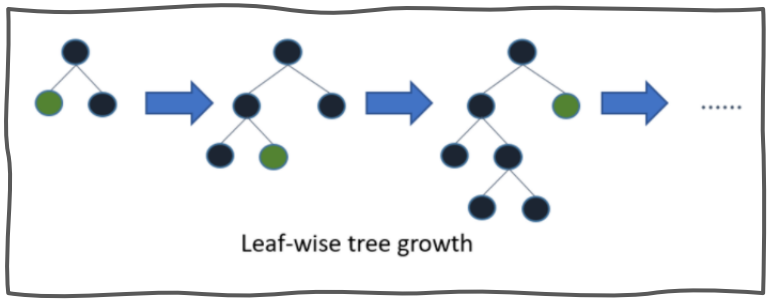
CatBoost 하이퍼파라미터 튜닝 결과

✓PCA + mwmote를 적용한 train 데이터를 이용하여 학습

F1 Score = 0.36231

LGBM

- Light Gradient Boosting Model
- **최대** delta loss**가 증가하도록 잎의 개수를 정함.**



LGBM 장점

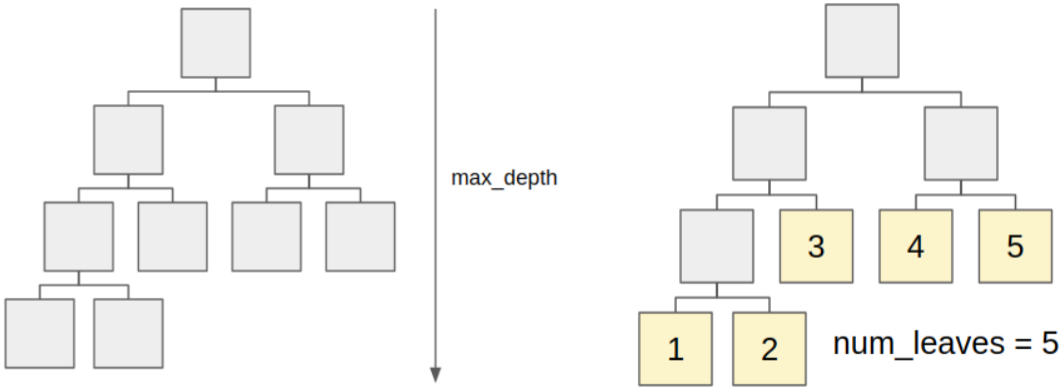
1. **적은 메모리**
2. **엄청난 속도**
3. **높은 정확도**
4. GPU **활용 가능**

LGBM

파라미터 튜닝

무수히 많은 파라미터..
그러나! Ensemble형식!!
구조자체가 파라미터에 맞게 큰 그림에서 맞춰져,
정확도면에서 큰 차이X

특정 파라미터에 초점



LGBM

LGBM 하이퍼파라미터 튜닝

num_leaves

전체 Tree의 leave 수
default : 31

↓ 2^max_depth 보다 작은 값

iterations = 36

max_depth

트리의 최대 깊이를 규정
default: -1

↓

max_depth = 6

learning_rate

학습률
default: 0.1

↓

learning_rate= 0.05

LGBM

파라미터 튜닝

ax_depth / num_leaves / feature_fraction
learning_rate / min_data_in_leaf

최종 결과

parameter

최적의 파라미터 탐색

max_depth=6
num_leaves=36
feature_fraction=0.7
learning_rate=0.05
min_data_in_leaf = 100

LGBM

LGBM **파라미터 튜닝 결과**

✓ PCA + MWMOTE**를 적용한** train **데이터를 이용하여 학습**

↓

F1 Score = 0.36065

튜닝할 수록 점점 낮아지는 score..

XGBoost

Extreme Gradient Boosting

Boosting기반 ensemble 모델

$$y'_i = \sum_{k=1}^{K} f_k(x_i)\ , f_k\ \in F$$

$$\text{Obj} = \sum_{i=1}^{n} l(y_i, y'_i) + \sum_{k=1}^{K} \Omega(f_k)$$

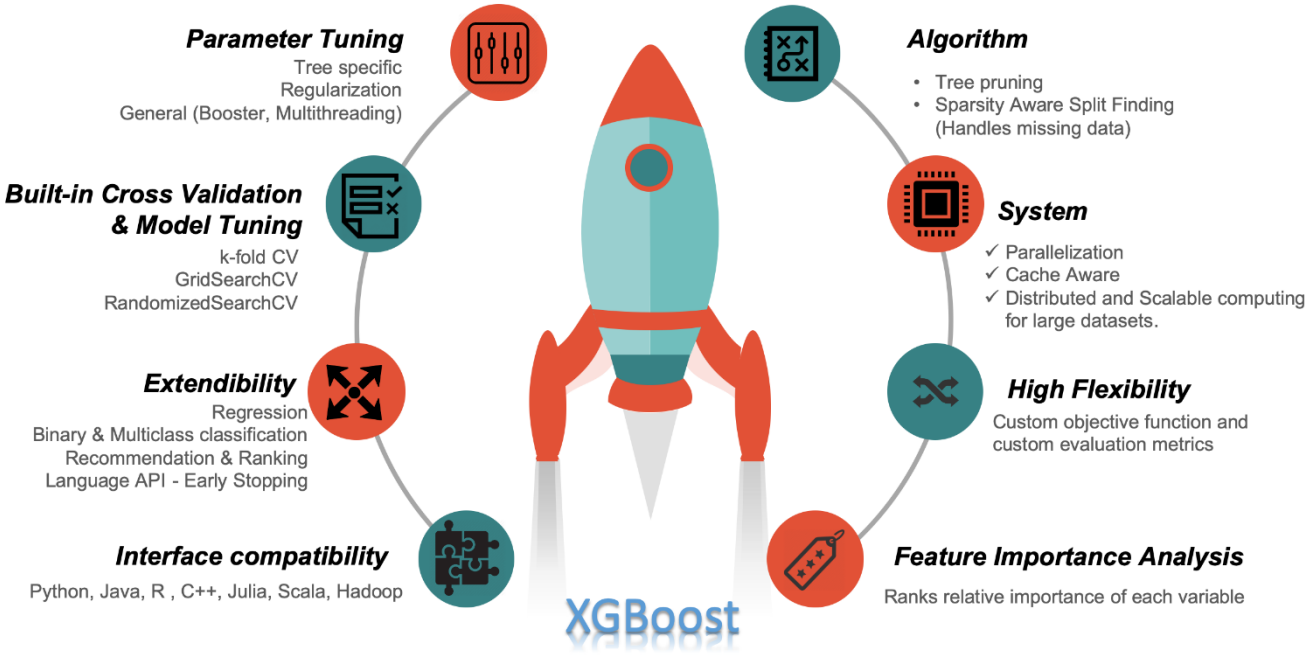
$* \ where\ y'_i = \text{predict score corresponding to } x_i,$

$f_k = k\ th\ decision\ tree\ in\ function\ space\ F,$

$l = loss\ function,$

$\Omega = regularization\ term(= complexity\ of\ Trees)$

XGBoost


Parameter Tuning
Tree specific
Regularization
General (Booster, Multithreading)
Built-in Cross Validation & Model Tuning
k-fold CV
GridSearchCV
RandomizedSearchCV
Extendibility
Regression
Binary & Multiclass classification
Recommendation & Ranking
Language API - Early Stopping
Interface compatibility
Python, Java, R , C++, Julia, Scala, Hadoop
Algorithm
• Tree pruning
• Sparsity Aware Split Finding (Handles missing data)
System
✓ Parallelization
✓ Cache Aware
✓ Distributed and Scalable computing for large datasets.
High Flexibility
Custom objective function and custom evaluation metrics
Feature Importance Analysis
Ranks relative importance of each variable
XGBoost

XGBoost

XGBoost 하이퍼파라미터 튜닝

max_depth

Decision Tree의 깊이 한도
커질수록 모델이 복잡해져
오버피팅 가능성 증가

↓

max_depth = 3

min_child_weight

관측치에 대한 가중치 합의 최소
노드가 더 이상 분할되는 것을 막음

값이 높을수록 과적합 방지

↓

min_child_weight = 5

scales_pos_weight

불균형 데이터에서
유용한 샘플 스케일링

↓

scales_pos_weight = $\sqrt{9}$

XGBoost

파라미터 튜닝

eta / max_depth / min_child_weight / subsample /
colsample_bytree / scale_pos_weight

최종 결과

parameter

최적의 파라미터 탐색

Eta = 0.1
Max_depth = 3
Min_child_weight = 5
Subsample = 0.2
Colsample_bytree = 0.6,
Scale_pos_weight = √9

XGBoost

XGBoost **파라미터 튜닝 결과**

PCA**만 적용한** train **데이터를 이용하여 학습**

F1 Score = 0.42553

로지스틱 회귀

- 선형 회귀 분석과는 다르게 종속 변수가 범주형 데이터를 대상
- 결과가 특정 분류로 나뉘기 때문에 일종의 분류(Classification) 기법으로 칭함
- 일반적으로 가장 간단하고 빠르게 수행되는 알고리즘이지만 최상의 결과를 기대하기는 어려움

로지스틱 회귀

로지스틱 회귀 결과

PCA**와** MWMOTE**를 적용한** train **데이터를 이용하여 학습**

F1 Score = 0.36464

RandomForest

- 앙상블 학습 방법의 일종으로, training 과정에서 다수의 결정 트리를 구성해서 분류 및 예측하는 모델
- 트리 기반 모델인 랜덤 포레스트는 불균형 데이터에서 준수한 성능을 내는 것으로 알려져 있음
- Python에서는 랜덤 포레스트가 불균형 데이터를 다룰 때 여러가지 옵션을 부여할 수 있게 라이브러리 제공

RandomForest

- 일반 랜덤 포레스트
- 균형 랜덤 포레스트

- RandomForestClassifier
- BalancedRandomForest Classifier

RandomForest

RandomForest 하이퍼파라미터 튜닝

max_features

무작위로 선택할 feature의 개수

max_features = 'log2'

min_samples_leaf

노드를 분할한 후 리프 노드에
있어야 하는 최소 샘플 수

min_samples_leaf= 1

n_estimators

만들 나무의 수 지정
클수록 더 일반화 된 결과 생성
하지만 시간 복잡도 증가

n_estimators = 500

RandomForest

파라미터 튜닝

max_features / min_samples_leaf / n_estimators

최종 결과

parameter

최적의 파라미터 탐색

max_features='log2'
min_samples_leaf=1
n_estimators=500

RandomForest

RandomForest 파라미터 튜닝 결과

PCA를 적용한 train 데이터를 이용하여 학습

F1 Score = 0.36666

BalancedRandomForest

BalancedRandomForest 하이퍼파라미터 튜닝

max_features

무작위로 선택할 feature의 개수

max_features = 'log2'

max_depth

트리의 깊이
클수록 복잡한 트리 생성
너무 크면 오버피팅 가능성 증가

max_depth= 15

n_estimators

만들 나무의 수 지정
클수록 더 일반화 된 결과 생성
하지만 시간 복잡도 증가

n_estimators = 10000

BalancedRandomForest

파라미터 튜닝

max_depth /max_features / min_samples_leaf /
n_estimators

최종 결과

parameter

최적의 파라미터 탐색

max_depth=15
max_features='log2'
min_samples_leaf=1
n_estimators=10000

BalancedRandomForest

BalancedRandomForest **파라미터 튜닝 결과**

원본 데이터를 적용한 train **데이터를 이용하여 학습**

F1 Score = 0.40740

4

결론

4 결론

전처리 및 샘플링

PCA 처리 데이터

PCA + MWMOTE 처리 데이터

원본 데이터

사용한 모델

AdaBoost
CatBoost
LGBM
XGBoost
Logistic Regression
RandomForest
BalancedRandomForest

4 결론

전처리 및 샘플링

PCA 처리 데이터

PCA + MWMOTE 처리 데이터

원본 데이터

사용한 모델

AdaBoost
CatBoost
LGBM
XGBoost
Logistic Regression
RandomForest
BalancedRandomForest

4 결론

최종 결과

XGBoost

F1 Score : 0.42553

THANK YOU