인공지능 기초

머신러닝

인공지능_ Day07

2023. 05. 16

CONTENTS

NAVER Cloud x BIT BITCamp

하이퍼파라미터 튜닝

그리드서치 (GridSearchCV)

- 하이퍼파라미터 튜닝 : 임의의 값들을 넣어 더 나은 결과를 찾는 방식
  → 수정 및 재시도하는 단순 작업의 반복

- 그리드 서치 : 수백 가지 하이퍼파라미터값을 한번에 적용 가능

- 그리드 서치의 원리 : 입력할 하이퍼파라미터 후보들을 입력한 후, 각 조합에 대해 모두 모델링해보고 최적의 결과가 나오는 하이퍼파라미터 조합을 확인

예) max_depth = [3, 5, 10]
Learning_rate = [0.01, 0.05, 0.1]

	max_depth	learning_rate
조합 1	3	0.01
조합 2	5	0.01
조합 3	10	0.01
조합 4	3	0.05
조합 5	5	0.05
조합 6	10	0.05
조합 7	3	0.1
조합 8	5	0.1
조합 9	10	0.1

그리드서치 (GridSearchCV)

1. XGBoost 모델의 parmeters

참조 공식문서 https://xgboost.readthedocs.io/en/stable/parameter.html

```
'n_estimators': [100,200,300,400,500,1000]}    #default 100 / 1~inf(무한대) / 정수
'learning_rate' : [0.1, 0.2, 0.3, 0.5, 1, 0.01, 0.001] #default 0.3/ 0~1 / learning_rate는 eta라고 해도 적용됨
'max_depth' : [None, 2,3,4,5,6,7,8,9,10] #default 3/ 0~inf(무한대) / 정수 => 소수점은 정수로 변환하여 적용해야 함
'gamma': [0,1,2,3,4,5,7,10,100] #default 0 / 0~inf
'min_child_weight': [0,0.01,0.01,0.1,0.5,1,5,10,100] #default 1 / 0~inf
'subsample' : [0,0.1,0.2,0.3,0.5,0.7,1] #default 1 / 0~1
'colsample_bytree' : [0,0.1,0.2,0.3,0.5,0.7,1] #default 1 / 0~1
'colsample_bylevel' : [0,0.1,0.2,0.3,0.5,0.7,1] #default 1 / 0~1
'colsample_bynode' : [0,0.1,0.2,0.3,0.5,0.7,1] #default 1 / 0~1
'reg_alpha' : [0, 0.1,0.01,0.001,1,2,10] #default 0 / 0~inf / L1 절대값 가중치 규제 / 그냥 alpha도 적용됨
'reg_lambda' : [0, 0.1,0.01,0.001,1,2,10] #default 1 / 0~inf / L2 제곱 가중치 규제 / 그냥 lambda도 적용됨
```

2. LightGBM 모델의 parmeters

참조 공식문서 https://lightgbm.readthedocs.io/en/latest/Parameters-Tuning.html

대체로 XGBoost와 하이퍼 파라미터들이 비슷하지만 leaf-wise 방식의 하이퍼 파라미터가 존재함

num_leaves : 하나의 트리가 가질 수 있는 최대 리프 개수
min_data_in_leaf : 오버피팅을 방지할 수 있는 파라미터, 큰 데이터셋에서는 100이나 1000 정도로 설정
feature_fraction : 트리를 학습할 때마다 선택하는 feature의 비율
n_estimators : 결정 트리 개수
learning_rate : 학습률
reg_lambda : L2 규제
reg_alpha : L1규제
max_depth : 트리 개수 제한

그리드서치 (GridSearchCV)

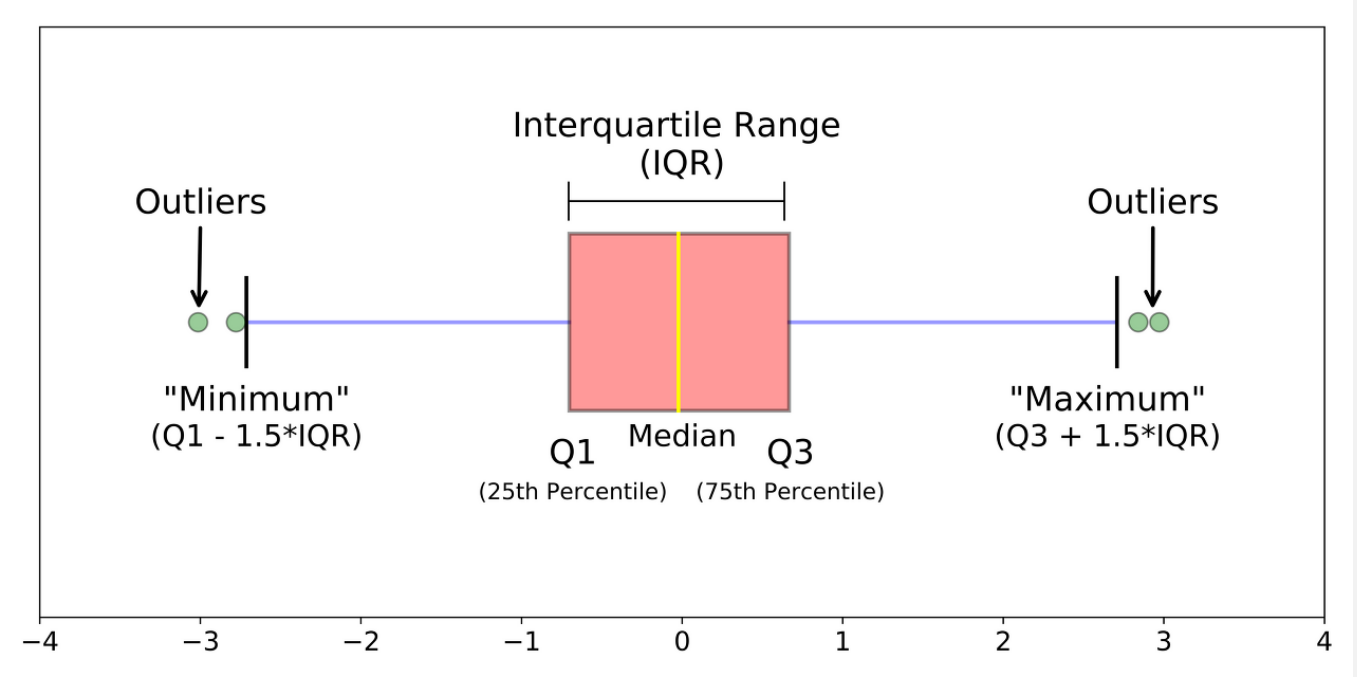
3. Catboost 모델의 parmeters

참조 공식문서 https://catboost.ai/en/docs/concepts/python-reference_catboost_grid_search

CatBoost 평가 지표 최적화에 가장 큰 영향을 미치는 하이퍼파라미터는 learning_rate, depth, l2_leaf_reg 및 random_strength

learning_rate: 학습률
depth: 각 트리의 최대 깊이로 과적합을 제어
l2_leaf_reg: L2 정규화(regularization) 강도로, 과적합을 제어
colsample_bylevel: 각 트리 레벨에서의 피처 샘플링 비율
n_estimators: 생성할 트리의 개수
subsample: 각 트리를 학습할 때 사용할 샘플링 비율
border_count: 수치형 특성 처리 방법
ctr_border_count: 범주형 특성 처리 방법

아웃라이어 (Outliers)



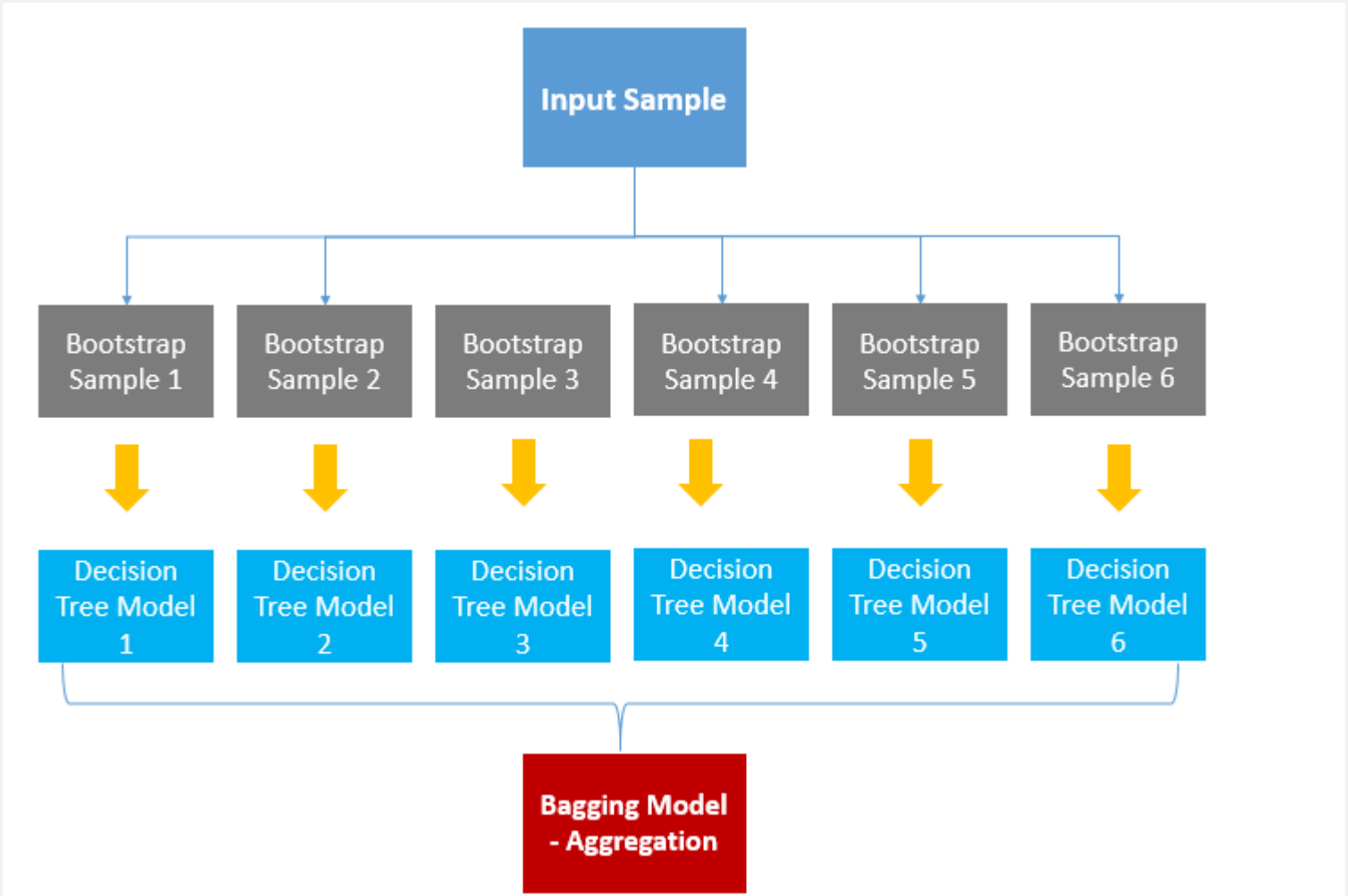
출처 https://hellojaehoon.tistory.com/149

Figure 1. Interquartile Range(IOR) Outliers

- IQR은 사분위 값의 편차를 이용하여 이상치를 걸러내는 방법
- 전체 데이터를 정렬하여 이를 4등분하여 Q1(25%), Q2(50%), Q3(75%), Q4(100%) 중 IQR는 Q3 - Q1 가 됨

배깅 (Bagging)

배깅 (Bagging)



출처 swallow.github.io

Figure 2. 배깅 Bagging

- Bagging은 Bootstrap Aggregation의 약자
- 배깅은 샘플을 여러 번 뽑아 (Bootstrap) 각 모델을 학습시켜 결과물을 집계 (Aggregration) 하는 방법
- 즉, 데이터로부터 부트스트랩 한 데이터로 모델을 학습시키고 학습된 모델의 결과를 집계하여 최종 결과값을 도출

배깅 (Bagging)

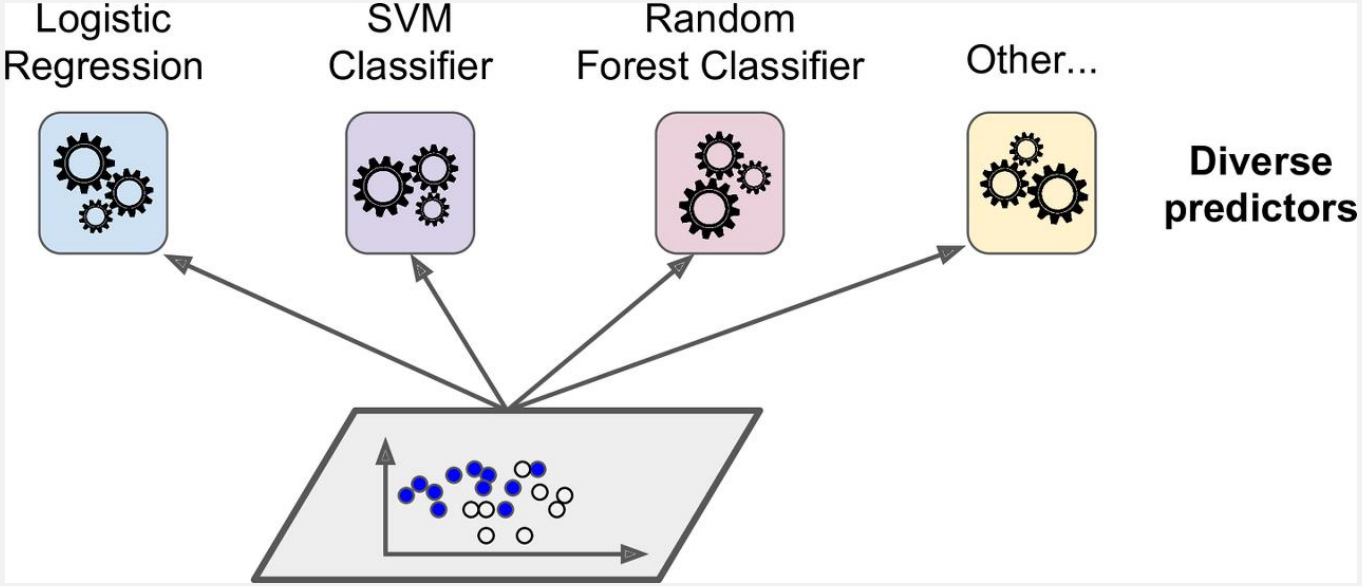
참조 코드

```
#2. 모델
from sklearn.ensemble import BaggingRegressor
from xgboost import XGBRegressor

model = BaggingRegressor(XGBRegressor(),
                         n_estimators=100,
                         n_jobs=-1,
                         random_state=72
                         )
```

보팅 (Voting)

보팅 (Voting)



출처 https://nicola-ml.tistory.com/95

Figure 3. 보팅 Voting

- Voting은 일반적으로 서로 다른 알고리즘을 가진 분류기를 결합하는 것 (참고 : 배깅의 경우 각각의 분류기가 모두 같은 유형의 알고리즘을 기반으로 함)
- 하드 보팅: 각 분류기의 예측 결과를 단순히 다수결(majority voting)로 결정
- 소프트 보팅: 각 분류기의 예측 확률을 평균하여 예측을 수행

보팅 (Voting)

참조 코드

```
#2. 모델
lr = LogisticRegression()
knn = KNeighborsClassifier(n_neighbors=8)
rfc = RandomForestClassifier()
xgb = XGBClassifier()

model = VotingClassifier(
    estimators=[('LR', lr), ('KNN', knn), ('RFC', rfc), ('XGB', xgb)],
    voting='soft',
    n_jobs=-1
)
```

실 습

1. 그리드서치
2. 배깅
3. 보팅
4. 아웃라이어
5. 데이터 전처리

수고하셨습니다.