

# 5주차 Ensemble Code Review

## 0. 머신러닝 앙상블

머신러닝의 앙상블에는 bagging, boosting, voting 방법이 있다.

이 중 boosting의 **gradient boosting**은 여러 문제점이 있는데

- 느리다.
- overfitting의 문제가 있다.

따라서 이런 문제를 해결하고자 성능이 더 좋게 만들어진 모델이 여러 등장하게 되는데 그 중 **XGBoost**도 한 가지이다.

## 1. XGBoost(eXtreme Gradient Boosting)

GBM 기반의 XGBoost는 다른 머신러닝보다 뛰어난 예측 성능을 보인다.(그렇지만 여전히 학습 시간은 느려 이를 보완한 LightGBM이 있음)

장점을 소개하자면

- GBM의 단점이었던 느린 수행 시간과  
과적합 규제(Regularization) 부재의 문제 극복
- Tree Pruning이 가능
- 자체적으로 내장된 교차 검증
- 조기종단(Early Stopping)
- 결손값 처리 기능

| 랜덤 포레스트에 비해선 느리다.

## 2. XGBoost의 하이파라미터


XGBoost는 결정트리, 랜덤 포레스트, 등의 기반을 기본적으로 가지고 있다. 그래서 결정트리, 랜덤 포레스트, GBM과 하이퍼 파라미터가 겹치는 것도 있다.

- `n_estimators` (혹은 `num_boost_round`): 결정 트리의 개수, 학습 모델의 수, 많아질수록 성능 향상의 가능성이 있으나, 속도가 느려짐
- `max_depth`: 최대 탐색 깊이, 너무 크면 과적합의 가능성, 너무 작으면 학습 성능 저하
- `min_samples_split`: 분할 종료 최소 샘플 수, 큰 수면 과적합을 막지만 학습 성능 저하 가능성

- `min_samples_leaf` : leaf node가 되기 위한 최소 샘플 수, `min_samples_split`과 비슷한 용도
- `colsample_bytree` : (default=1)GBM의 `max_features`와 유사하다. 트리 생성에 필요한 피처를 임의로 샘플링 하는데 사용, 피처가 매우 많은 경우 과적합을 조절하는데 사용
- `subsample` : weak learner가 학습에 사용하는 데이터 샘플링 비율
- `learning_rate` : 학습률, 너무 크면 gradient 발산의 가능성이 있으며, 너무 작으면 학습이 느림
- `min_split_loss` : 리프 노드를 추가적으로 나눌지 결정하는 값
- `random_state` : 특정 숫자로 고정 시에 실행 시마다 고정된 결과 반환
- `reg_lambda` : L2규제
- `reg_alpha` : L1 규제

이외에도 다양한 파라미터들이 존재한다. 하단의 링크를 참고

XGBoost Parameters — xgboost 2.0.0-dev documentation

 <https://xgboost.readthedocs.io/en/latest/parameter.html>


## 만약 과적합 문제가 심각한 경우

- eta 값을 낮춘다(0.01 ~ 0.1). eta 값을 낮출 경우 `num_boost_rounds`는 반대로 높여줘야 한다.
- `max_depth` 값을 낮춘다.
- `min_child_weight` 값을 높인다.
- `gamma` 값을 높인다.
- `sub_sample`과 `colsample_bytree` 등을 조정한다.

## 3. Dacon Code

[Private 3위] Xgboost + GridSearchCV

유전체 정보 품종 분류 AI 경진대회

 <https://dacon.io/competitions/official/236035/codeshare/7435?page=2&dtype=recent>



```
import xgboost
from xgboost import XGBClassifier
```

```
# 데이콘 코드에선 특정 검색체를 묶어서 진행한 코드와 전체 검색체를 진행한 코드가 있는데
# 앙상블 학습 설명 코드는 하나만 진행하였습니다.
chrom = [chrom_2, chrom_6, chrom_7, chrom_8, chrom_9, chrom_10]

# 검색체 3개 그룹씩 묶기
comb_3_chrom = list(combinations(chrom, 3))
# 검색체 4개 그룹씩 묶기
comb_4_chrom = list(combinations(chrom, 4))
# 전체 그룹 리스트로 생성
comb_chrom = comb_3_chrom + comb_4_chrom

model_list = []
for i in comb_chrom:
    snp_group = list(itertools.chain(*i))
    snp_group.append('trait')
    x = train_x[snp_group]
    x_train, x_test, y_train, y_test = train_test_split(x, train_y,
        test_size=0.2, random_state=42)

    # XGBClassifier() 생성
    # XGBoost의 원하는 파라미터를 dict형태로 설정
    xgb= XGBClassifier()
    xgb_param_grid={
        'n_estimators' : [100,200,300,400,500, 600],
        'learning_rate' : [0.01,0.05,0.1],
        'max_depth' : [2, 3, 4, 5]
    }
    # GridSearchCV로 모델 생성
    xgb_grid=GridSearchCV(xgb, param_grid = xgb_param_grid, scoring="f1_macro", cv=5)
    # 만들어진 모델로 fit
    xgb_grid.fit(x_train, y_train)

    # GridSearchCV의 refit으로 이미 학습된 estimator 반환
    model = xgb_grid.best_estimator_
    # predict로 예측하며 값들을 pred에 저장
    pred = model.predict(x_test)

    # 정확도 계산 f1 score 사용
    # f1 score는 Precision과 Recall의 조화평균으로 주로 분류 클래스 간의 데이터가 불균형이 심할때 사용한다.
    from sklearn import metrics
    print(metrics.f1_score(pred, y_test, average='macro'), snp_group)
    model_list.append(model)
```

- '`n_estimators`' : [100,200,300,400,500, 600]
  - 결정 트리의 갯수, 학습 모델의 수
- '`learning_rate`' : [0.01,0.05,0.1]
  - 학습률

- `'max_depth'` : [2, 3, 4, 5]
  - 최대 탐색 깊이

## GridSearchCV()

앞에서 XGBoost의 하이퍼파라미터와 다양한 파라미터들을 확인할 수 있었는데 파라미터가 굉장히 많아 어떤 값을 넣어줘야 할 지 고민될 수 있다. 그래서 scikit learn의 grid search를 통해 하이퍼파라미터 튜닝을 할 수 있게끔 한다.

```
GridSearchCV( model,          # estimator(classifier, regressor, pipeline가능)
               param_grid=,    # 찾고자하는 파라미터를 dict형태로 만들어 넣는다.
               cv= ,           # 교차 검증에서 몇 개로 분할되는지 지정
               scoring=None,    # 예측 성능을 측정할 평가 방법 넣기
                                # Classification일때 'accuracy', 'f1'
                                # Regression 일때 'neg_mean_squared_error', 'r2'...

               n_jobs= 4,      # 병렬 처리갯수? -1은 전부)
               refit=True      # default가 True.
                                # 최적의 하이퍼 파라미터를 찾아서 재학습 시킴
            )

GCV=GridSearchCV( )          # 옵션.
GCV.fit( )                   # train_X, train_Y
GCV.best_params_              # 좋은 파라미터를 보여줌.
GCV.best_score_               # 0.88 좋은 estimator로
                                # 교차검증된점수를 보여줌.

# 예측방법.1
model=GCV.best_estimator_    # 최적의 파라미터로 모델 생성
model.predict()               # refit=True이기때문에 좋은 estimator로
                                # 수정되어졌으므로 바로 예측에 적용할 수 있다.

# 예측방법2
GCV.predict( )                # test_X, 학습후 최적의 파라미터로 예측한다
```