**Intro to Model Tuning: Grid and Random Search**

최적의 하이퍼파라미터는 학습데이터에 따라 달라진다. 따라서 각 문제에 맞게 조정해야 한다. 하이퍼파라미터 조정에는 몇 가지 방법이 있다.

1. 매뉴얼: 직관, 경험, 추측을 기바능로 하이퍼 파라미터를 선택하고 하이퍼 파라미터로 모델을 교육한 후 검증 데이터에 점수를 매긴다. 인내심이 바닥나거나 결과에 만족할 때까지 과정을 반복한다.
2. 그리드 서치: 하이퍼파라미터 값의 그리드를 설정하고 각 조합에 대해 모델을 교육하고 검증 데이터에 대한 점수를 매긴다. 이 접근 방식에서는 모든 하이퍼 파라미터 값의 조합을 시도하지만 매우 비효율적일 수 있다.
3. 랜덤 서치: 하이퍼파라미터 ㄱ밧의 그리드를 설정하고 랜덤 조합을 선택하여 모델과 점수를 학습한다. 검색 반복 횟수는 시간, 리소스에 따라 설정된다.
4. 자동 하이퍼파라미터 조정: 그레디언트 강하, 베이지안 최적화 또는 진화 알고리즘과 같은 방법을 사용하여 최상의 하이퍼파라미터에 대한 검색을 수행한다.

이 노트에서는 graident boosting모델에 대한 접근법을 그리드 서치와 랜덤 서치를 사용해서 구현할 것이다. 향후 노트북에서는 베이지안 최적화, 특히 Hyperopt 라이브러리를 사용하여 자동화된 하이퍼 파라미터 튜닝을 구현할 예정이다.

**Gradient Boosting**

Gbm은 최근 최고의 기계 학습 모델 중 하나로 떠오르고 있다. Gbm은 정보가 행과 열에 있는 구조화된 데이터와 수백만 개의 관측치가 있는 중간 크기의 데이터 세트에 매우 효과적이다. 이 모델은 현재 캐글의 대부분의 경쟁에서 최고의 성능을 발휘하는 방법이고 성능이 하이퍼 매개 변수 선택에 크게 의존하기 때문에 우리는 이 모델에 초점을 맞출 것이다. Gbm에 대해 당신이 알아야 할 기본은 거의 항상 많은 개별 학습자를 훈련시켜 작동하는 앙상블 방식이라는 것이다. 그러나 나무가 병렬로 훈련되는 랜덤 포레스트와 달리 gbm에서는 각 트리가 이전 트리의 실수로부터 학습하면서 순차적으로 훈련된다. 수백 또는 수천 명의 약한 학습자가 겷삽되어 경사하강법을 사용하여 교육 중에 학습한 각 개인의 기여도를 바탕으로 하나의 강력한 앙상블 학습자가 된다.

gbm에는 전체 앙상블과 개별 의사결정트리를 제어하는 많은 하이퍼 파라미터가 있다. 하이퍼 파라미터 간에는 복잡한 상호 작용이 있기 때문에 이론만으로 어떤 하이퍼 파라미터 조합이 가장 적합한지 알기는 어렵다. 최적의 하이퍼 파라미터 값을 찾는 유일한 방법은 데이터 세트에서 다양한 조합을 시도하는 것이다.

우리는 lgbm을 사용해서 gbm을 구현할 것이다. 이 방법은 사이킷런에서 사용할 수 있는 것보다 훨씬 빠르다(일부는 더 정확하다고 말한다).

여기서 사용할 데이터는 1만개의 행으로 구성된 데이터의 하위 집합으로 작업할 것이다. 하이퍼파라미터 조정은 계산 비용이 매우 많이 들고 캐글 커널에서 전체 데이터 세트를 사용하는 작업은 몇 번 이상의 검색 반복으로는 가능하지 않다. 그러나 여기서 구현할 동일한 아이디어를 전체 데이터 세트에 적용할 수 있으며 이 노트는 gbm을 대상으로 하지만, 모든 머신러닝 모델에 동일하게 적용할 수 있다.

튜닝 결과를 테스트하기 위해 교육 데이터 중 일부인 6천 행을 별도의 테스트 세트로 저장할 것이다. 하이퍼 파라미터 튜닝을 수행할 때 테스트 데이터의 하이퍼 파라미터를 조정하지 않는 것이 중요하다. 우리는 검증 데이터에 대해 조정된 최종 모델을 평가할 때 테스트 데이터를 한 번만 사용할 수 있다. 이 노트에서 실제로 우리의 방법을 테스트하려면 모든 교육 데이터에 대해 최상의 모델을 교육하고 실제 테스트 데이터를 예측한 다음 제출해야 한다.

*# Data manipulation*

import pandas as pd

import numpy as np

*# Modeling*

import lightgbm as lgb

*# Splitting data*

from sklearn.model\_selection import train\_test\_split

N\_FOLDS = 5

MAX\_EVALS = 5

features = pd.read\_csv('../input/home-credit-default-risk/application\_train.csv')

*# Sample 16000 rows (10000 for training, 6000 for testing)*

features = features.sample(n = 16000, random\_state = 42)

*# Only numeric features*

features = features.select\_dtypes('number')

*# Extract the labels*

labels = np.array(features['TARGET'].astype(np.int32)).reshape((-1, ))

features = features.drop(columns = ['TARGET', 'SK\_ID\_CURR'])

*# Split into training and testing data*

train\_features, test\_features, train\_labels, test\_labels = train\_test\_split(features, labels, test\_size = 6000, random\_state = 50)

제공된 훈련 데이터를 자체적으로 훈련데이터와 테스트데이터로 나눈다.

**교차검증**

각 하이퍼파라미터 값의 조합을 평가하려면 검증 세트에서 점수를 매겨야 한다. 테스트 데이터에서 하이퍼 파라미터를 조정할 수 없다. 우리는 최종 모델을 평가할 때 테스트 데이터를 한 번만 사용할 수 있다. 테스트 데이터는 실제 데이터에 배치될 때 모델 성능의 추정치 역할을 하도록 되어 있으므로 실제 성능에 대한 공정한 추정치가 제공되지 않기 때문에 테스트 데이터에 모델을 최적화하면 안된다. 따라서 검증 세트를 사용하는 것이 올바른 방법이다. 그러나 중요한 교육 데이터를 별도의 교육 및 검증 세트로 분할하는 대신 Kfold 교차 검증을 사용한다. 교육 데이터를 보전하는 것 외에도 단일 검증 세트를 사용하는 것보다 테스트 세트의 일반화 성능을 더 잘 추정할 수 있따. 각 하이퍼 파라미터 세트의 성능은 교차 검증을 통해 곡선 아래의 영역(ROC\_AUC)에 의해 결정된다.

이 예에서는 성능을 평가하기 위해 각 하이퍼 파라미터 값 집합을 사용하여 모형을 5회 교육하고 테스트하는 5배 교차 검증을 사용한다. 하이퍼 파라미터 튜닝에 시간이 많이 걸리는 이유 중 하나는 교차 검증을 사용하기 때문이다. 교육 세트가 충분히 크면 별도의 단일 검증 세트를 사용하는 것으로도 해결할 수 있지만 교차 검증이 과적합을 방지하는 더 안전한 방법이다.

Kfold 교차 검증을 구현하기 위해 lightgbm 교차 검증 기능 (cv)를 사용할 것이다. 조기 정지를 사용할 필요가 없는 다른 머신러닝 모델의 경우 사이킷런의 GridSearchCV나 RandomSearchCV를 사용할 수 있다.

**Early stopping**

gbm에서 가장 중요한 하이퍼 파라미터 중 하나는 추정기 수이다. 우리는 이것을 우리의 검색에서 또 다른 하이퍼 파라미터로 설정할 수 있지만, 더 좋은 방법이 있다. 조기중지는 지정된 반복 횟수 동안 유효성 검사 오류가 줄어들지 않을 때까지 훈련하는 것을 의미한다. Gbm의 경우 더 많은 의사 결정 트리를 교육하는 것을 의미하며, 이 예에서는 100라운드로 조기 중지를 사용하므로 검증 오류가 100라운드로 줄어들지 않을 때까지 교육이 계속된다. 그런 다음 검증 데이터에서 가장 높은 점수르 ㄹ얻은 추정치의 수가 최종 모형에 사용할 추정기의 수로 선택된다.

조기 정지 개념은 일반적으로 gbm과 심층 신경망에 적용되므로 이해하기 좋은 기술이다. 이것은 교육 데이터에 과적합하지 않음으로써 검정 세트의 일반화 성능을 향상시키는 것을 목표로 하는 많은 형태의 정규화 중 하나이다. 추정기를 계속 추가하면 모델의 용량이 증가하기 때문에 교육 오차는 항상 줄어들 것이다. 비록 이것이 긍정적으로 보일 수 있지만, 이것은 모형이 훈련 데이터를 암기하기 시작하고 나서 새로운 시험 데이터에 대해 잘 수행되지 않을 것이라는 것을 의미한다. 모형이 교육 데이터에 너무 많이 의존하기 시작하므로 추정기를 계속 추가할수록 모형의 분산이 증가한다.

초기 중지는 교차 검증 기능에서 lightgbm 라이브러리로 구현하기 쉽다. 조기 정지 횟수만 넘기면 된다. Cv 기능을 사용하려면 먼저 lightgbm 데이터 세트를 만들어야 한다.

*# Create a training and testing dataset*

train\_set = lgb.Dataset(data = train\_features, label = train\_labels)

test\_set = lgb.Dataset(data = test\_features, label = test\_labels)

일련의 하이퍼 파라미터를 교차 검증에 전달해야 하므로 lightgbm의 기본 하이퍼 파라미터를 사용한다. Cv 호출에서 num\_boost\_round는 1만 이상으로 설정되지만 조기 중지를 사용하고 있으므로 이 숫자에 실제로 도달하지 않는다. 우리가 사용하고 있는 지표는 곡선 아래의 수신기 작동 특성 영역(roc\_auc)이다. 아래 코드는 5 kfold 및 100회 조기 정지 라운드로 교차 검증을 수행한다.

*# Get default hyperparameters*

model = lgb.LGBMClassifier()

default\_params = model.get\_params()

*# Remove the number of estimators because we set this to 10000 in the cv call*

del default\_params['n\_estimators']

*# Cross validation with early stopping*

cv\_results = lgb.cv(default\_params, train\_set, num\_boost\_round = 10000, early\_stopping\_rounds = 100,

metrics = 'auc', nfold = N\_FOLDS, seed = 42)